

ADVANCED  
COURSES+  
SCIENTIFIC  
CONFERENCES

WELLCOME GENOME CAMPUS

NEXT GENERATION SEQUENCING BIOINFORMATICS  
27 JANUARY-1 FEBRUARY 2019  
UNIVERSITY OF WITWATERSRAND, JOHANNESBURG, SOUTH AFRICA



COURSE MANUAL

IN COLLABORATION WITH:



UNIVERSITY OF THE  
WITWATERSRAND,  
JOHANNESBURG



**H3ABioNet**

Pan African Bioinformatics Network for H3Africa

CONNECTING  
SCIENCE



**Wellcome Genome Campus Advanced Course**

**Next Generation Sequencing  
Bioinformatics**

**27 January - 01 February 2019**

*Held at*

*University of Witwatersrand, Johannesburg, South Africa*

*Organised by:*

WGC Advanced Courses

**Wellcome Genome Campus, Hinxton, Cambridge, UK**

## STAFF

**Advanced Courses Manager**      **Rebecca Twells** (email: [advancedcourses@wellcomegenomecampus.org](mailto:advancedcourses@wellcomegenomecampus.org))  
Wellcome Genome Campus, Hinxton, Cambridge, UK

**Course Instructors**

**Thomas Keane** EMBL-EBI, UK

**Jacqui Keane** Wellcome Trust Sanger Institute, UK

**Shaun Aron** University of the Witwatersrand, South Africa

**Gerrit Botha** University of Cape Town, South Africa

**Petr Danecek** Wellcome Trust Sanger Institute, UK

**Amel Ghouila** Institut Pasteur de Tunis, Tunisia

**Fatma Guerfali** Institut Pasteur de Tunis, Tunisia

**Phelelani Mpangase** University of the Witwatersrand, South Africa

**Victoria Offord** Wellcome Trust Sanger Institute, UK

**Sumir Panji** University of Cape Town, South Africa

<b>Advanced Courses Team</b>	<b>Yvonne Thornton</b>	Advanced Courses Event Manager
	<b>Julie Ormond</b>	Advanced Courses Laboratory Manager
	<b>Darren Hughes</b>	Advanced Courses Programme Manager
	<b>Nicola Stevens</b>	Advanced Courses Event Organiser
	<b>Kate Waite</b>	Advanced Courses Assistant Lab Manager
	<b>Pamela Black</b>	Advanced Courses Education Officer
	<b>Martin Aslett</b>	Advanced Courses IT Manager
	<b>Alice Matimba</b>	Advanced Courses Overseas Development Officer
	<b>Karon Chappell</b>	Advanced Courses Event Organiser
	<b>Adam Crewdson</b>	Advanced Courses Laboratory Assistant

**Wellcome Genome Campus**      **URL:** [www.wellcomegenomecampus.org/coursesandconferences](http://www.wellcomegenomecampus.org/coursesandconferences)  
**Advanced Courses**              **email:** [advancedcourses@wellcomegenomecampus.org](mailto:advancedcourses@wellcomegenomecampus.org)

## ACKNOWLEDGEMENTS

With grateful thanks to the staff at the University of Witwatersrand, Johannesburg, South Africa for their invaluable assistance in organising this Workshop.

**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Instructors**

**Thomas Keane EMBL-EBI**

Dr Thomas Keane completed his PhD degree in the area of distributed computing and high-throughput phylogenomics from NUI Maynooth (Ireland) in 2006. He worked at the Wellcome Trust Sanger Institute from 2006-16, co-founding the Vertebrate Resequencing Informatics team, and lead the Sequence Variation Infrastructure team (2014-16). In these roles, he managed the production for the 1000 Genomes Project, the UK10K project, and the Mouse Genomes Project. In 2016, he joined EMBL-EBI as head of the European Genome-phenome Archive (EGA) and the European Variation Archive (EVA). His research interests include mouse genetics and genomics, methods for structural variation, and genome assembly. He leads the Large Scale Genomics workstream of the Global Alliance for Genomics and Health (GA4GH).



**Jacqui Keane Wellcome Sanger Institute**

Jacqueline McQuillan has a PhD in Software Engineering. She joined the Pathogen Genomics group at the Wellcome Trust Sanger Institute as a postdoctoral fellow in 2008 where she worked on RNA-Seq gene finding in helminth genomes. From 2010, she has managed the Pathogen Informatics team at Sanger. The Pathogen Informatics team is responsible for providing pipelines and analysis support to the Pathogen Genomics group along with developing and maintaining key pieces of software such as Artemis, ACT and GeneDB.



**Shaun Aron University of the Witwatersrand, South Africa**

Shaun Aron is currently a bioinformatics consultant and lecturer at the Sydney Brenner Institute for Molecular Bioscience (SBIMB) at the University of the Witwatersrand in Johannesburg, South Africa. After pursuing an undergraduate degree in Genetics and Microbiology followed by an Honours degree in Human Genetics, he handed over the pipettes to the experts and entered the then still developing field of bioinformatics pursuing an MSc degree. Currently he is a member of the H3Africa Pan African Bioinformatics Network (H3ABioNet), which is a network consisting of 28 research institutes in 18 countries, tasked with developing and supporting informatics and genomics research in Africa. His research interests include GWAS of complex diseases in African populations, exploring population diversity, structure and admixture in Africa and bioinformatics education and training.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Instructors**

**Gerrit Botha University of Cape Town, South Africa**

Gerrit Botha is a Bioinformatics engineer at the Computational Biology division of the University of Cape Town. His background is electronic engineering but gained experience in bioinformatics whilst working as a software developer at the division. His area of expertise is in human variant analysis, microbial analysis and bioinformatics compute environment and pipeline design.



**Petr Danecek Wellcome Sanger Institute**

Petr Danecek has a background in computer science and PhD in biophysics.

After his PhD he worked at the University of Texas Medical Branch in Galveston, focusing on development of statistical methods for flavivirus vaccines. Petr joined WTSI in 2009 to participate in the Mouse Genomes Project where he was responsible for variant calling, data analysis and data processing. His focus is methods development and is one of the lead SAMtools, BCFtools and HTSlib developers. He has participated in the large-scale sequencing projects such as 1000 Genomes Project or UK10k, most recently he has been involved in the HipSci and DDD projects.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Instructors**

**Amel Ghouila Institut Pasteur de Tunis, Tunisia**

Amel Ghouila is currently a bioinformatician at Institut Pasteur de Tunis, where she works on the frame of the pan-African bioinformatics network H3ABionet supporting researchers and their projects while developing bioinformatics capacity throughout Africa. She has a background in computer sciences with a PhD in Bioinformatics from the Laboratory of Informatics, Robotics and Microelectronics, Montpellier, France.

Within H3ABionet, Amel is involved in postgraduate courses organisation and teaching.

Amel is also part of the Institut Pasteur IGDA network core teaching team for the hands-on NGS courses and introductory Bioinformatics courses organized across the Pasteur International Network.

Her main research interests are in DataScience and Bioinformatics of Pathogens. She works mainly on NGS analysis pipelines for pathogens genomics and transcriptomics data analysis, combining different data sources to enhance functional annotation and perform comparative studies.

She is a Mozilla Science fellow -- an Open Science and Open Education advocate. She was nominated by Mozilla as one of "50 People Who Are Making the Internet a Better Place."

She is the general secretary for the African Society of Bioinformatics and Computational Biology (ASBCB) and is a regional ambassador for the Technovation program: the global tech entrepreneurship for young girls.



**Fatma Guerfali Institut Pasteur de Tunis, Tunisia**

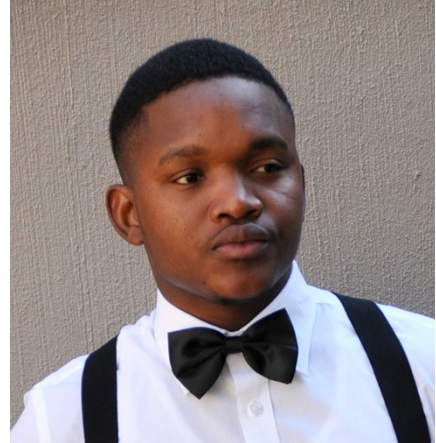
Fatma Guerfali is member of the Institut Pasteur de Tunis (IPT), Laboratory of Transmission, Control and Immunobiology of Infections. Fatma worked on high-throughput approaches applied to analyze dual host and pathogen interactions. She is currently involved in the generation, analysis, and data mining of high throughput DNAseq and RNAseq data from non-model pathogens showing differential disease manifestation in human patients and classified according to their associated virulence. Fatma is actively involved in organizing/providing bioinformatics trainings adapted to LMIC context and has been invited as a trainer/speaker in several local and international courses and workshops for post-graduates in the field of NGS.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Instructors**

**Phelelani Mpangase – University of the Witwatersrand, South Africa**

Phelelani Mpangase joined the Sydney Brenner Institute for Molecular Bioscience at University of the Witwatersrand as a Bioinformaticist. He has since provided Bioinformatics support to students and researchers in the University. He has worked on projects involving protein structure and function, next generation sequencing data analysis, pipeline design, metagenomics and transcriptomics. Apart from being a Bioinformaticist, he is also in the 3rd year of his PhD in Bioinformatics. His PhD project is mainly focused at analysing transcriptomic data from black South African patients with the Systemic Sclerosis disease and designing reproducible pipelines in Nextflow for analysing transcriptomic data and metagenomics.



**Victoria Offord Wellcome Sanger Institute**

Victoria is a bioinformatician in the Pathogen Informatics team, providing support to pathogen groups at the Wellcome Trust Sanger Institute. She studied biological sciences as an undergraduate at the University of Exeter where she also completed her Masters in bioinformatics. She then joined the Royal Veterinary College (RVC) as part of the BioChip consortia, developing tools and resources for the pan-viral array. This led to a passion for all things pathogenic, her doctorate and a fellowship in marine mammal innate immunity, during which she provided core support in bioinformatics analysis to RVC researchers.



**Sumir Panji University of Cape Town, South Africa**

Sumir Panji is a bioinformatician within the H3ABioNet consortium working on the implementation and interpretation of bioinformatics solutions and new technologies to diverse biological problems, and is actively involved in providing bioinformatics support and training to the H3Africa projects. Sumir's main interests lie in high quality bioinformatics education, creating and implementing computational and analyses workflows, biological algorithms, high performance computing, the overall application of bioinformatics and genomics to better understand complex biological systems and is passionate about developing bioinformatics capacity on the African continent.





**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

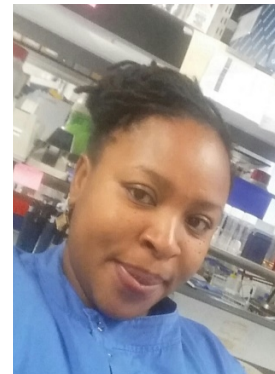
**Abdalla Munir Khalid Abdalla    National University Research Institute,  
SUDAN**

I'm interested in tackling biological problems in a broad view, more in the realms of systems biology type of view. NGS technology offers this possibility by providing high throughput data at the whole genome and transcriptome level. Currently my research interests include: WGS of bacterial isolates, genome evolution and dynamics, metagenomics of human microbiome. Our ongoing research is WGS of 250 clinical isolates of MRSA from Sudan hospitals to determine antibiotic resistance and virulence properties using illumina platform. The Course will help to acquire the skill to analyze the NGS data.



**Chioma Achi    Usmanu Danfodiyo University, NIGERIA**

My current research is on the genomic analysis of antimicrobial resistant Salmonella recovered from humans and animals from Nigeria. I am a Researcher at the Centre for Advanced Molecular Research and Training (CAMRET) and a Lecturer of Veterinary Public Health and preventive Medicine in Usmanu Danfodiyo University Sokoto, Nigeria. I am a 2nd year PhD student at the University of Cambridge with strong interest in molecular biology. It is my hope that by attending this course, I will be able to improve my bioinformatics skills on Next Generation sequencing and be able to transfer the knowledge and skills learnt.



**Abdul-Rahman Adamu Bukari    Kumasi Centre for Collaborative  
Research, KNUST, GHANA**

I have a research interest in infectious disease epidemiology, pathogenesis and hostpathogen interaction. Specifically, I am keen on employing genomics and other molecular-based techniques in understanding the (re)emergence, spread and pathogenesis of infections of zoonotic origins. Currently, I seek to employ Next Generation Sequencing (NGS) Bioinformatics tools to identify zoonotic (viral and bacterial) pathogens reserved in livestock and their keepers across Ghana. This hands-on introduction to NGS Bioinformatics course comes at a perfect time as my research will heavily rely on generating and analysing large amount of NGS sequence data. I join this course with the



objective of obtaining the requisite practical and theoretical underpinning of operation the NGS technology and the accompanying post-sequencing analysis. But also this will be a great opportunity for networking and exposure to the cutting-edge work of invited speakers.

**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Samah Ahmed    Centre of Bioinformatics and Systems Biology, SUDAN**

Currently, I am working on discovery of genetic variation in pharmacokinetic i.e., absorption distribution, metabolism and excretion (ADME) genes among African; particularly Sudanese patients with aggressive prostate cancer and their influence on response to anti-neoplastic chemotherapy.

Data of this project will be generated from Whole Genome Sequencing (WGS) for selected Sudanese patients and their controls; data will be subjected for downstream analysis for variant calling, structural analysis, copy number variation analysis and pathway analysis.

These information will be tailored to develop precision medicine plan for African populations.



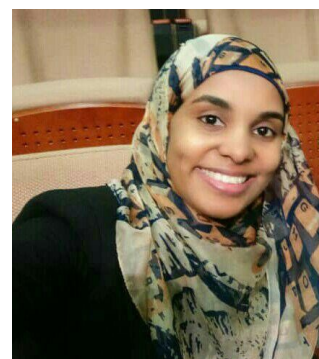
**Samuel Ahuno    Kwame Nkrumah University of Science and Technology, GHANA**

Samuel is a Research Assistant/Student in Alexander Kwarteng (CIFAR-Azrieli Global Scholar | Humans and Microbiome) research group. His interest extends to Cancer Genomics and is part of the Ghanaian research team studying liquid biopsies for early breast cancer in collaboration with American Cancer Genomics experts Prof. Will Foulkes, McGill University and Dr. Paz Polak, Icahn School of Medicine at Mount Sinai. Following this course he aims to be able to establish the bioinformatics pipelines after Ultra-Low Pass Sequencing (ULPS) and Whole Genome Sequencing (WGS) of cell-free DNA samples to understand mutational signatures of Ghanaian breast cancer patients.



**Rayan Ali    University of Khartoum, SUDAN**

My name is Rayan Ali from Sudan. I am a PhD student at Brighton and Sussex Medical School, UK based at the Mycetoma Research Centre, University of Khartoum, Sudan. My research focuses on studying genetic susceptibility to mycetoma which is a slow-growing, destructive infection of the skin and underlying tissues that is endemic in impoverished Sudanese communities. I joined this course to learn more about the pipelines used to analyse NGS data and WES analysis to be more specific, since part of my PhD project will be done using this technology.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Olaitan Awe    University of Ibadan, NIGERIA**

Olaitan's research involves the analysis of molecular sequencing data, including the development of tools to analyze Next-Generation Sequencing datasets. He develops a computational model for the identification of protein-coding and non-coding regions in transcriptomes. He uses comparative genomics techniques to investigate the genetic relationships between pathogenic viruses like Ebola and other filoviruses and also between Zika virus and its related arboviruses. He also investigates gene expression and novel viruses in RNA sequencing datasets for cancer research to see if there is a correlation between an unusual expression of these genes and cancer e.g. Human Endogenous Retroviruses in Acute Myeloid Leukemia.



**Odunayo Azeez    University of Ibadan, NIGERIA**

Dr Odunayo Azeez is a Lecturer in Veterinary Physiology in the Department of Veterinary Physiology and Biochemistry, University of Ibadan, Nigeria. He has a PhD in Veterinary Anatomy and Physiology from the University of Pretoria, South Africa.

He is currently studying Transcriptomic characterization of non-model animals and the effects of pollution, inflammation and stress on non-model organisms, using NGS by RNASeq.

The training on NGS and Bioinformatics shall broaden his horizon and provide the requisite knowledge needed for analysis of NGS data using Bioinformatics on currently available data and new data to be generated in future projects and studies.

**Shakuntala Baichoo    University of Mauritius, MAURITIUS**

My current and ongoing research consists of performing the computational analysis of cancer datasets, in order to advance knowledge discovery from various angles including pan-genome perspectives, workflow development and predictive modeling. The datasets to be analysed will come from public data banks such as ICGC, AACR and TCGA; these will be mostly in the form of NGS datasets such as WGS, RNA-Seq and WES. Hence this course is of utmost relevance to my ongoing research as it will equip me with the appropriate know-how to perform the necessary data analyses, to advance knowledge discovery in the field of cancer omics.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

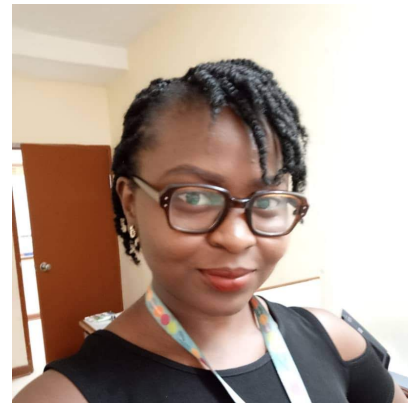
**Christina Balle    University of Cape Town, SOUTH AFRICA**

My research focuses on examining the impact of different hormonal contraceptive methods on the vaginal microbiota in a randomized, crossover cohort of adolescent females from Cape Town. We have assessed whether hormonal contraceptive-induced changes to the vaginal microbiota affect the numbers, activation status and co-receptor expression of genital immune cell populations (analyzed by flow cytometry) and the inflammation status of the genital tract mucosa (cervical cytokine levels analyzed by Luminex) to establish a biological basis of how hormonal contraception and the vaginal microbiota potentially alters HIV susceptibility. Whole genome, virome and RNA sequencing and analyses are yet to be conducted.



**Chisom Ezekannagha    Bioscience Center, NIGERIA**

My current research focuses on developing RNA-Seq data analysis pipeline in order to understand the molecular mechanism and identify set of genes that can demonstrate that they are differentially expressed under different conditions of the Cassava Mosaic disease (CMD). NGS techniques are clearly suited to explaining the downstream analyses of these transcriptomic data from which I seek to identify important biological information. Being involved in the course, I expect to acquire deep and comprehensive training required to assess and analyze the large volumes of RNA-Seq data and to gain practical experience of analytical techniques in NGS.



**Abiodun Fatoba    University of KwaZulu-Natal, SOUTH AFRICA**

Coccidiosis is a renowned intestinal disease of livestock which is caused by the apicomplexa parasite of the genus *Eimeria*. I am currently working on these parasite species among broiler chickens in KwaZulu-Natal province of South Africa. My research involves the use of different genomic fragments genes to identify and determine the genetic diversity among these species. In addition, we are also exploring the genomic diversity of *Eimeria tenella* which is highly pathogenic and widely distributed species of this genus with a genetic diversity and population structure that is geographically unique. This big data necessitates strong bioinformatics skills.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

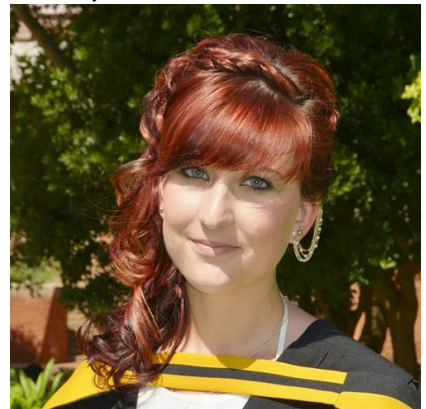
**Amira Khalaf    Medical research institute, EGYPT**

My PhD Thesis work entitled (Molecular characterization of Oculoauriculovertebral spectrum in Egyptian patients) relies mainly on array CGH and NGS technologies to reveal the etiology of this heterogeneous group of disorders. The work is performed in France and data is now being ready for analysis. I was hosted for one month training there but I wish I could gain more skills in the field of Bioinformatics to be qualified to begin my data analysis .Attending this course is a great opportunity and will help me a lot in completing my thesis which is the corner stone in building my career.



**Duodane Kindler    National Zoological gardens Pretori, SOUTH AFRICA**

The Molecular Disease Epidemiology unit at the National Zoological Garden of South Africa performs epidemiological research on various infectious diseases (viruses, bacteria and parasites) that occur in wildlife (for which there is very little data/information). This includes conventional characterisation and classification of these pathogens via traditional parasitology, virology, bacteriology and mycology techniques. Next-Generation Sequencing (NGS) and bioinformatics is being used to further investigate and characterise selected veterinary and zoonotic pathogens, which will facilitate development of a national sequence database with reference genomes of South African isolates/strains of important veterinary pathogens and parasites (e.g. Toxoplasma gondii and Gastrointestinal parasites in wildlife in South Africa).



**Standford Kwenda    National Institute for Communicable Diseases,  
SOUTH AFRICA**

My goal is to become an independent researcher in the field of Systems Immunology, focusing on implementation of bioinformatics methods, in deciphering key cell types of the immune response and their gene expression profiles contributing to health and disease. My current interests include HIV-1 functional cure research, using RNA-seq for mRNA, long noncoding RNA and microRNA profiling, including implementation of co-expression network analysis to identify potential regulatory ncRNA biomarkers of importance in HIV-1 control. This course will help enhance my bioinformatics skills by equipping me with current, advanced techniques and necessary expertise in transcriptome profiling and network construction.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Imen Larbi    Pasteur Institut of Tunis, TUNISIA**

Imen Larbi graduated from the Ecole veterinaire de Sidi Thabet, Tunisia. She holds a MS and a PhD degree in Biology from the Faculty of Sciences Tunis. Currently she is conducting researches in the Laboratory of Epidemiology and Veterinary Microbiology, Institut Pasteur of Tunis. The sphere of her research includes virology poultry infectious disease and it mainly covers epidemiological studies on avian influenza disease. She earned from the University of Paris Diderot, France a degree (diplome d'université). She had a fellowship from the U.S. Civilian Research & Development Foundation (CRDF Global) to conduct her project on avian influenza virus. She is taking part in the Tunisian national surveillance of avian influenza research programs carried by the Institute Pasteur de Tunis.



**Jessica Levesley    University of Witwatersrand, SOUTH AFRICA (Zuid Afrika)**

Currently, I am involved in investigating the allele sequence diversity of the loci associated with Huntington disease in African Ancestry individuals. To ultimately assess the role of sequence diversity in modifying disease. In order to achieve this, a specialised high throughput next-generation sequencing assay and data analysis pipeline has been used. Attending the course would allow for additional knowledge and skills to aid in the analysis of the generated NGS data. As well as provide exposure to methodologies that could be applied to my current dataset as well as future projects.



**Zané Lombard    University of Witwatersrand, SOUTH AFRICA**

My research is focused on understanding human genetic variation and how it impacts on disease. I study African populations, and therefore I am also keen to better understand African-specific genetic variation. I am currently at a place in my career where I am using my genetics and computational skills to build my own research group focused on neurodevelopment and - disorders. I am committed to applying new technologies to improve the diagnosis of patients with rare developmental diseases and believe understanding the genetic causes of these disorders, and their underlying biological mechanisms, will give us fundamental insights into human development.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Faridath Abèni Tatiane Massou    National Reference Laboratory of  
Mycobacteria, BENIN**

I am currently working on a project named DIAMA (DIAGnosis of Multidrug resistant tuberculosis in Africa). The goal is to identify a test that will allow avoiding the use of culture for both diagnosis and follow-up of multidrug resistant tuberculosis patients by using molecular tools such as the sequencing of several genes. We have already been trained on the technical part but not on the analysis of the results. This training will be helpful because no fund was planned for that. I will get essential bioinformatics skills and knowledge required to analyze our data during the project and beyond.



**Christina Meiring    Stellenbosch University, SOUTH AFRICA**

My research involves the investigation of genomic variation within African wild dogs in the Kruger National Park (KNP). The aim of this study is to sequence the genomes of wild dog individuals from KNP to understand their susceptibility/resistance to diseases and other threats; and to develop tools to identify genetic factors conferring adaptive advantages. This study requires working with large volumes of next-generation sequence data and it is therefore relevant to this course as I will gain the necessary expertise to be able to handle large amounts of data and to subject it to numerous analyses.



**Kelebogile Moremi    Stellenbosch University, SOUTH AFRICA**

My current research interest is to embrace the application of NGS technologies and bioinformatics tools to advance precision medicine in Africa. This involves analysing WES data of breast cancer patients to filter for deleterious mutations in known disease-associated genes (ExAc gene-list). Variant classification is performed using international criteria, extended to family screening where appropriate. Ongoing research is focused on comparative analyses of data generated using WES and Nanopore sequencing platforms in genetically uncharacterised breast cancer patients.

The course outline set for the WGCAC NGS Bioinformatics Course will provide an ideal training environment to better equip myself.



**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Abel Abera Negash    Armauer Hansen Research Institute, ETHIOPIA**

I am currently working on a PhD project that aims to serotype and genotype *Streptococcus pneumoniae* isolates from Ethiopian children with pneumococcal diseases. We are now planning to perform whole genome sequencing (WGS) on selected isolates. This course will therefore prepare me for the ensuing data analysis. There is also a considerable lack of expertise in the area of performing next generation sequencing and analyzing the generated data in Ethiopia. The training will therefore give me a platform in which I can contribute my share in filling this void. Through and beyond my PhD project, I aspire to apply WGS to study the molecular epidemiology, transmission dynamics and antimicrobial resistance of bacterial pathogens in Ethiopia. The training will therefore undoubtedly be a stepping stone for me to achieve these goals.



**Patracia Nevondwe    University of the Witwatersrand, SOUTH AFRICA**

My PhD project will be looking at the contribution of de novo mutations in the development of neurodevelopmental disorders. I will perform a targeted NGS mutation screening on genes known to cause or interact with causative genes of neurodevelopmental disorders. My objectives for the course are to learn how to analyse NGS data generated from targeted and exome sequencing, with more emphasis on how to call and interpret de novo mutations.



**Jean Pierre Rutanga    University of Rwanda, RWANDA**

I am currently working on my PhD project entitled “Development of new molecular tools for diagnosis and surveillance of invasive Salmonellosis”. This project has two main goals:  
1) development of 16S rRNA based diagnostic tools for bacterial bloodstream infections (using Illumina Miseq) and 2) development of point-of-care *Salmonella* Typhi genome sequencing workflows (using Illumina Miniseq and MinIon™). Therefore, I am confident that following this course, I will get sufficient knowledge in next generation sequencing bioinformatics tools that I will be using during the analysis of data resulting from my project.





**Wellcome Genome Campus Advanced Course  
Next Generation Sequencing Bioinformatics  
University of Witwatersrand, Johannesburg, South Africa  
27 January - 1 February 2019  
Participants**

**Natalie Smyth Sydney Brenner Institute for Molecular Bioscience,  
SOUTH AFRICA**

I am currently a research assistant at the SBIMB biobank. My work involves extracting, quantifying and storing DNA from human tissue samples, and performing PCR and genotyping. In 2019, I will be registering for my PhD through the University of Witwatersrand, with the title 'Understanding the genetic basis of high and low LDL-cholesterol in African populations.' This course will be beneficial for the completion of this project as the protocol includes using a targeted sequencing approach on approximately 150 samples in order to elucidate pathogenic variants that may contribute to high and low LDL- cholesterol in Africa.



**Ashraf Yahia Osman Mohamed University of Khartoum, SUDAN**

I am trying to explore the genes and molecular pathways involved in the pathogenesis of hereditary spinocerebellar degeneration (SCD) in a cohort of Sudanese families. To accomplish this; my toolbox contains: next-generation sequencing, homozygosity mapping, in addition to the basic molecular genetic techniques. I am planning also to use the Zebra fish as an animal model to study the molecular consequences of the identified novel mutations.



## PROGRAMME

The hands-on programme will cover several aspects of next generation sequencing data analysis, including lectures, discussions and practical computational sessions\* covering the following:

- Introduction to NGS technologies
- Introduction to the unix command line
- Advanced unix
- NGS data formats and tools
- Sequence alignment+QC
- SNP/indel theory and practical
- Structural variation theory and practical
- RNA-seq analysis
- ChiP-seq analysis
- Sequencing data visualisation with the Integrated Genomics Viewer
- Accessing public sequencing repositories
- Participant projects and presentations (final day)

## Learning Outcomes

On completion of the course, participants should be able to:

- Use the unix command-line as a tool for data analysis
- Describe the different NGS data file formats available
- Perform QC assessment of high throughput sequencing data
- Explain the algorithmic concepts behind short read alignment, variant calling and structural variant detection
- Perform read alignment, variant calling and structural variation detection using standard tools
- Analyse RNA-Seq and CHiP-seq data
- Perform a genome assembly using NGS data
- Describe the different data types available in public sequence repositories and how they are organised

The practical sessions will be taught exclusively through Unix/Linux. Introductory tutorials to the UNIX/Linux operating system and command line, can be found on the following links:

<http://www.ee.surrey.ac.uk/Teaching/Unix>  
<http://swcarpentry.github.io/shell-novice/>

**Next Generation Sequencing Bioinformatics, University of Witwatersrand, Johannesburg, South Africa, 27 January - 1 February 2019**

	Sunday 27/1/2019	Monday 28/1/2019	Tuesday 29/1/2019	Wednesday 30/1/2019	Thursday 31/1/2019	Friday 1/2/2019	
08:00						Bus to Institute	08:00
08:30		Bus to Institute	Bus to Institute	Bus to Institute Variant Calling Petr	Bus to Institute ChIP-Seq Victoria	Assembly Thomas	08:30
09:00		Introduction to Sequencing Technologies Fatma	File Formats QC & Data Processing Petr				09:00
09:30							09:30
10:00		Tea/Coffee	Tea/Coffee	Tea/Coffee	Tea/Coffee	Tea/Coffee	10:00
10:30		Intro the the VM Jacqui					10:30
11:00							11:00
11:30							11:30
12:00		Seminar	Seminar				12:00
12:30					Lunch		12:30
13:00	Registration buffet lunch Place: Hotel	Lunch	Lunch	Lunch		Lunch	13:00
13:30					Seminar		13:30
14:00	Intro and Advanced Learning and Training (ALT) Session 1 Place: Hotel Martin	Introduction to Unix Jacqui	Read alignment Thomas	Public Repositories Jacqui		Group Presentations	14:00
14:30				Structural Variants Thomas	RNA-Seq Victoria	All	14:30
15:00							15:00
15:30	Tea/Coffee	Tea/Coffee	Tea/Coffee	Tea/Coffee	Tea/Coffee	Tea/Coffee	15:30
16:00	Introduction WTAC	Advanced Unix Jacqui					16:00
16:30	Instructors and Participant Talks Place: Hotel						16:30
17:00							17:00
17:30						Bus to Hotel	17:30
18:00					Group Task Prep All		18:00
18:30	Welcome Drinks and speed networking Place: Hotel	Bus to Hotel	Bus to Hotel	Bus to Hotel			18:30
19:00					ALT session 2 Martin		19:00
19:30	Dinner Place: Hotel				Bus to Hotel		19:30
20:00							



# Sequencing Technologies

Michael Quail  
mq1@sanger.ac.uk



## Sequencers



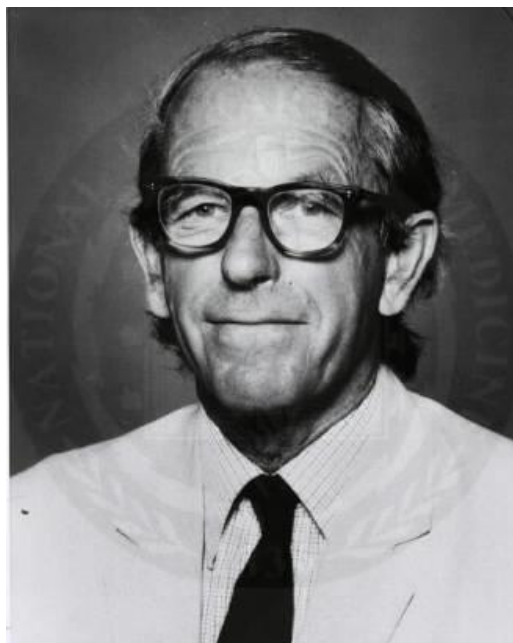


# Talk outline

- » History of sequencing
- » Description of different technologies
- » Outline of advantages and disadvantages
- » Look to the future



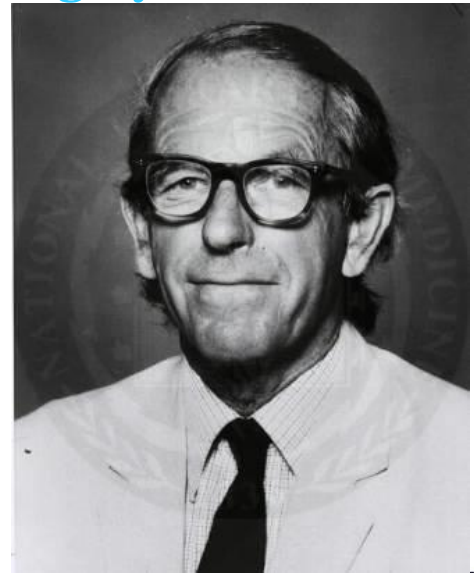
# Sanger Sequencing



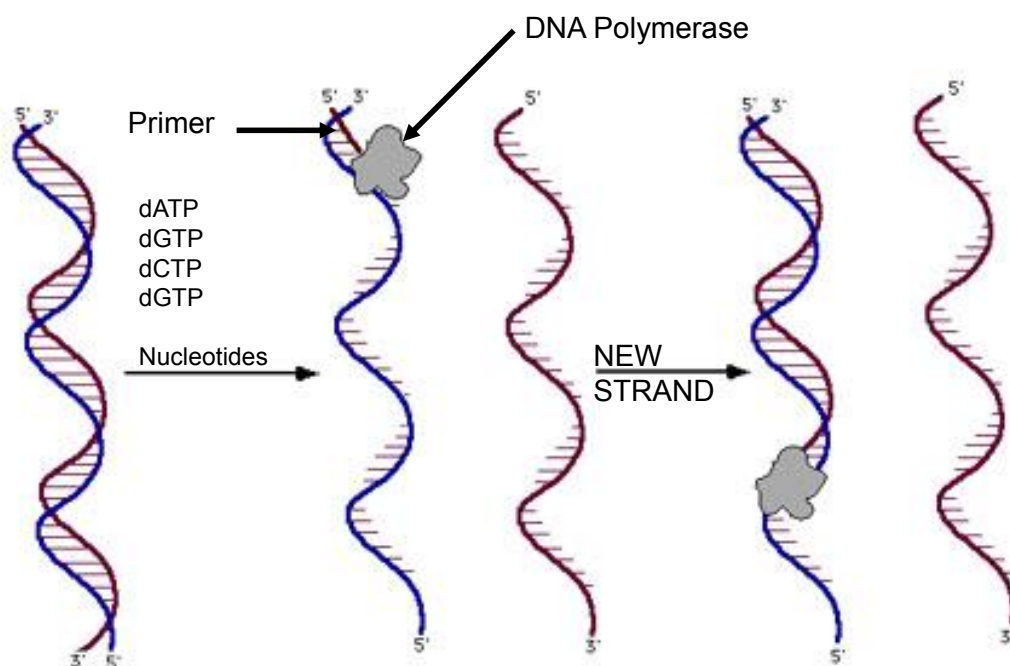


# Frederick Sanger

- » Discovered DNA sequencing by chain termination method
- » Nobel Prize 1 (1958)
  - » Complete amino acid sequence of insulin
- » Nobel Prize 2 (1980)
  - » For DNA sequencing



## Primer Extension

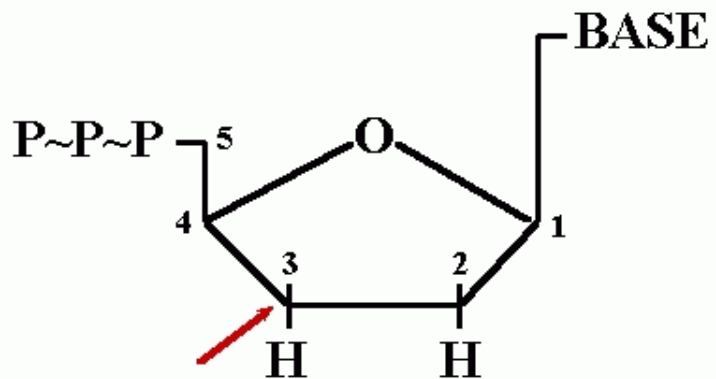


Template DNA



# Dideoxy Nucleotides

- Lack an -OH group at the 3-carbon position
- Cannot add another nucleoside at that position
- Prevent further DNA synthesis



## All Possible Terminations

DNA Polymerase reads the template strand and synthesizes a new second strand to match:



IF 5% of the T nucleotides are actually dideoxy T, then each strand will terminate when it gets a ddT on its growing end:

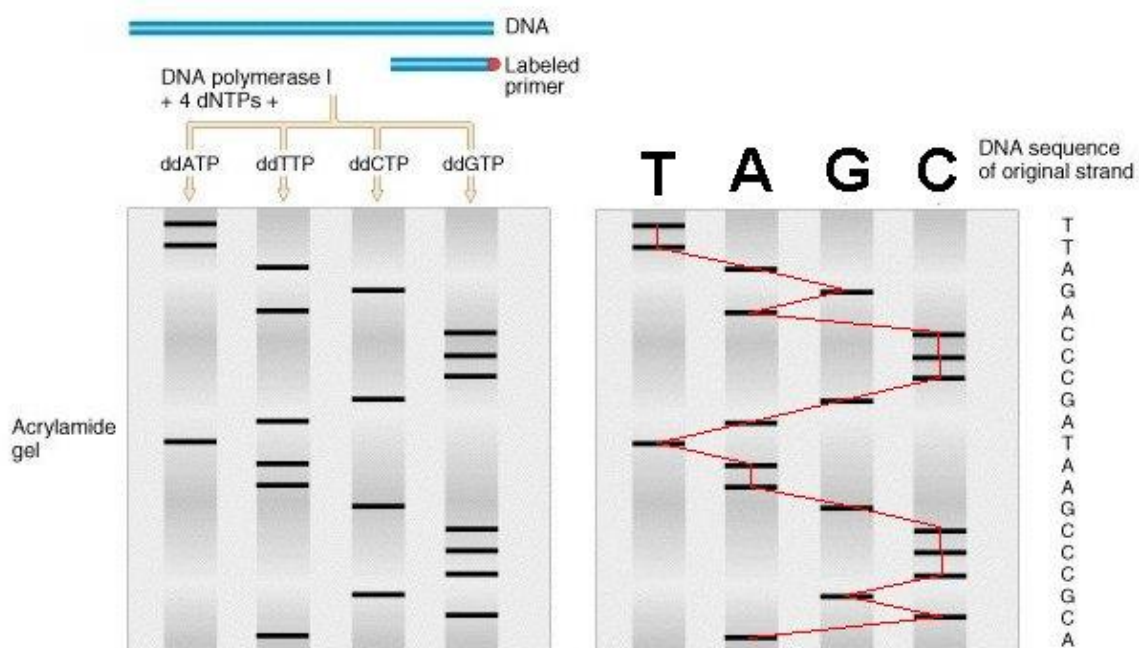
- 5' - TACGCGGTACGGTATGTTTCGACCGTTTAGCTACCGAT•
- 5' - TACGCGGTACGGTATGTTTCGACCGTTTAGCT•
- 5' - TACGCGGTACGGTATGTTTCGACCGTTT•
- 5' - TACGCGGTACGGTATGTTTCGACCGTT•
- 5' - TACGCGGTACGGTATGTTTCGACCGT•
- 5' - TACGCGGTACGGTATGTT•
- 5' - TACGCGGTACGGTATGT•
- 5' - TACGCGGTACGGTAT•
- 5' - TACGCGGTACGGT•
- 5' - TACGCGGT•

# Original Sanger Sequencing

- » 4 sequencing reactions performed for each template, each with different terminator
- » Radioactive primer or nucleotide used
- » Sequencing reactions run on <1mm polyacrylamide gel cast between two glass plates to separate fragments according to size
- » After run gel exposed to film and developed to reveal image



## Sequencing gel autorad

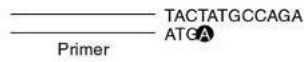




# Fluorescent Terminators

Primer extension reactions:

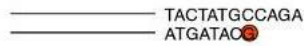
ddA reaction:



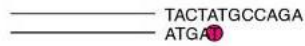
ddC reaction:



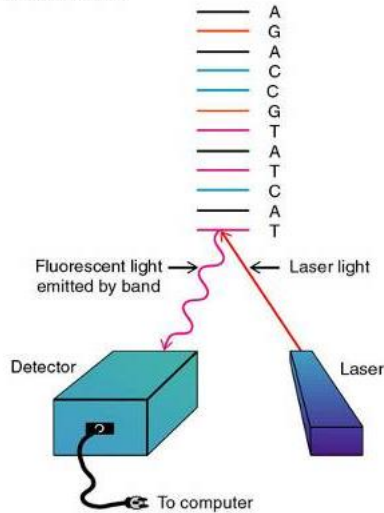
ddG reaction:



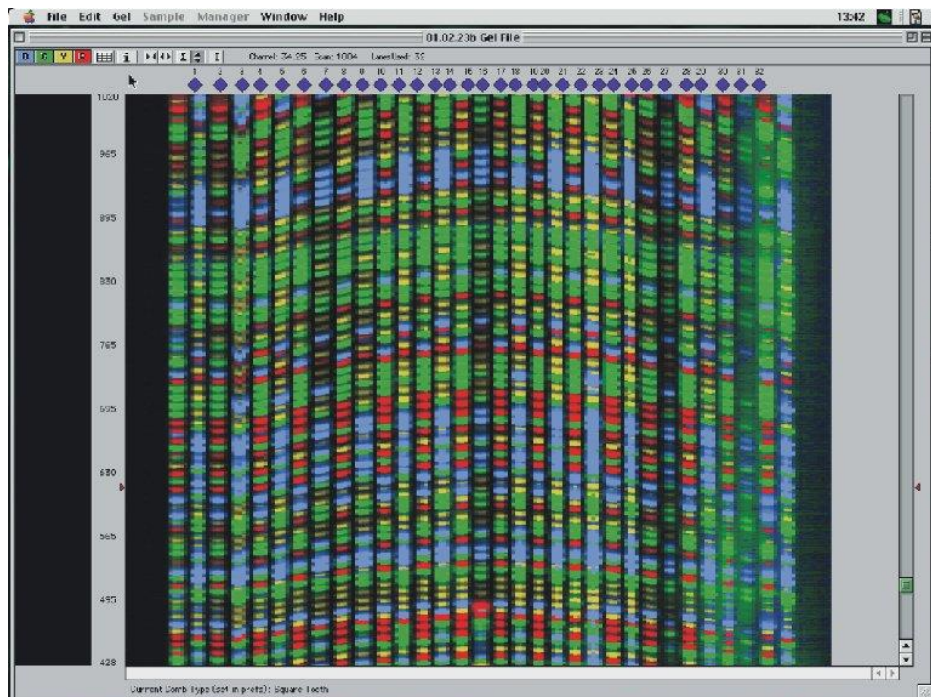
ddT reaction:

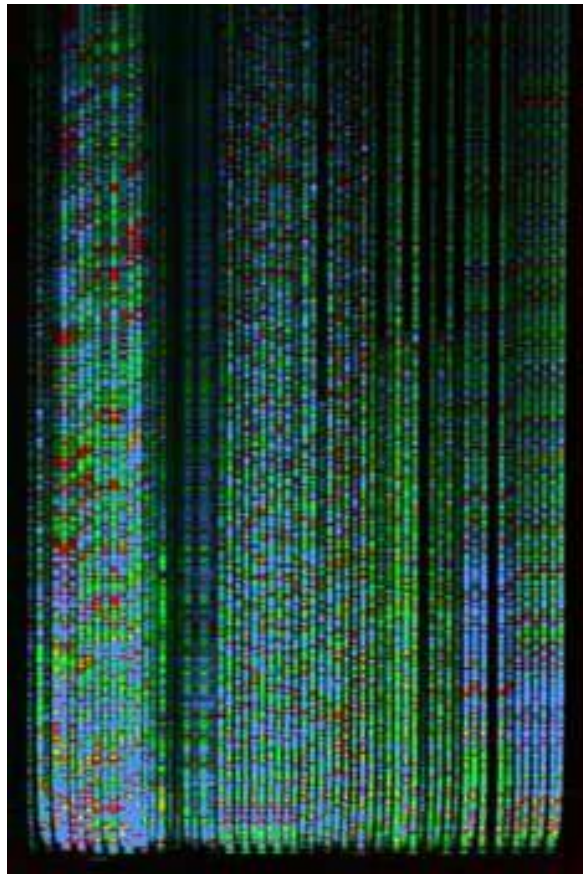


Electrophoresis:



# Fluorescent Gel Sequencing



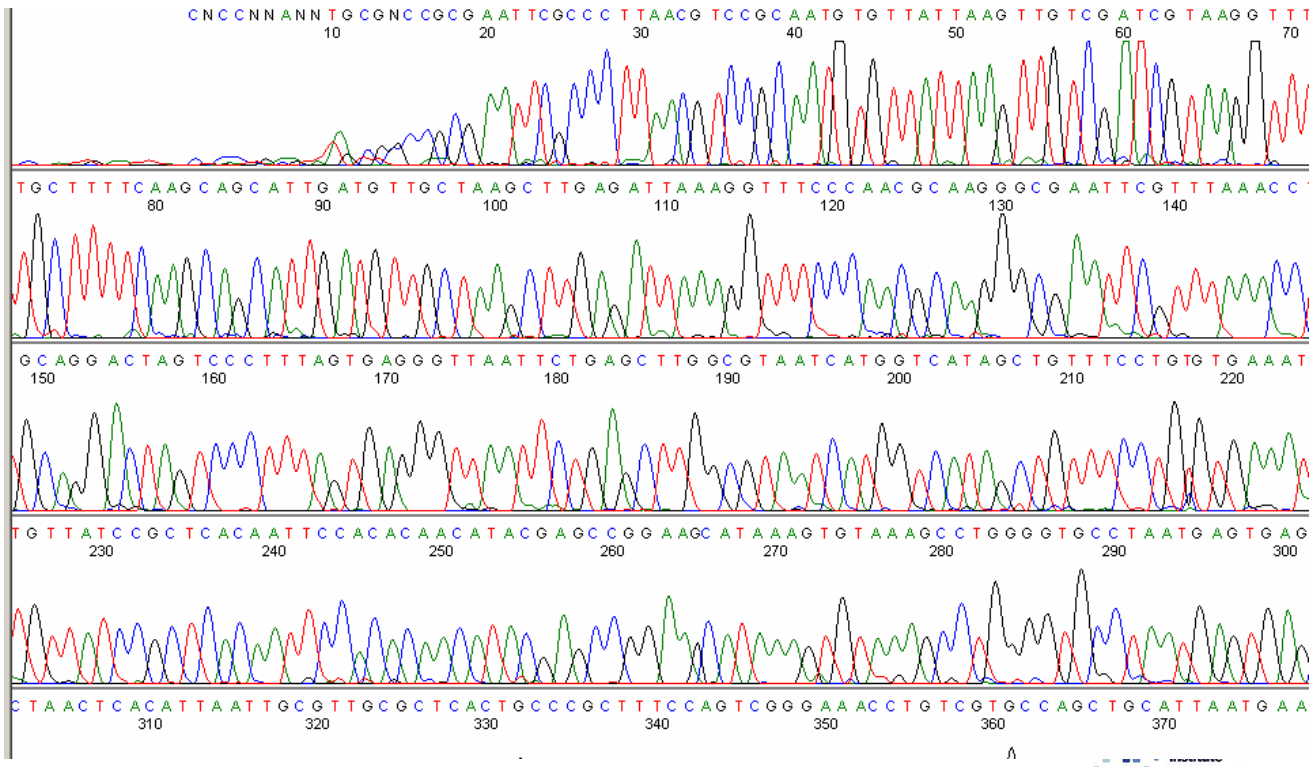


# ABI 3730





# DNA Sequence Files



# ABI Capillary 3730

- » Individual reactions -> 96 capillary array
- » Accurate, Q30
- » PCR errors
- » Cloning bias
- » 1000-base reads
- » 1-2 hour run time





# Error

- » Sequence quality Q is reported on a log scale
  - » Q10 is 1 error in 10
  - » Q20 is 1 error in 100
  - » Q30 is 1 error in 1000
  - » Q40 is 1 error in 10000
  - » Q50 is 1 error in 100000



# Capillary Sample Prep

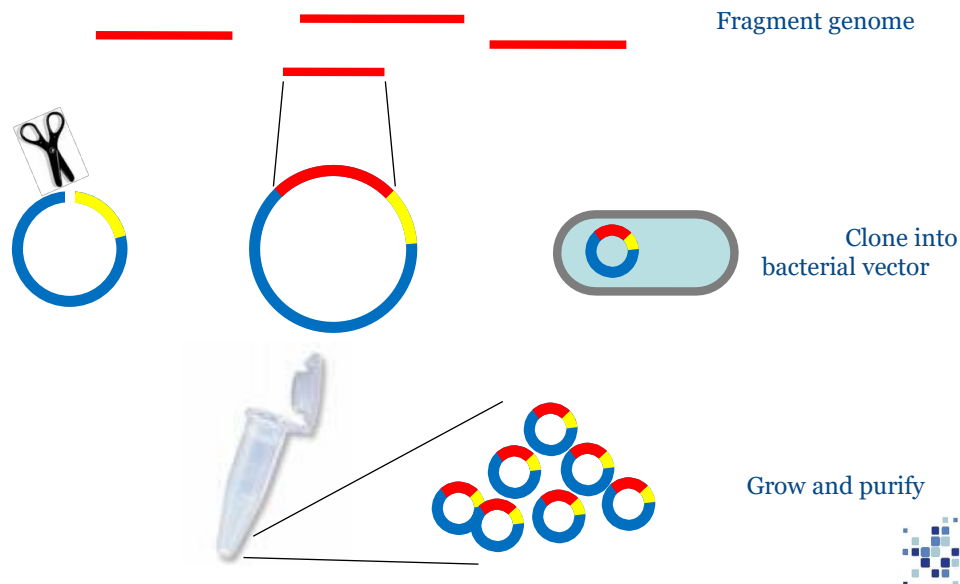
» PCR

Or

» DNA Miniprep



# Capillary Sample Prep



# ABI Capillary 3730

- » Fast + easy for individual samples
- » Robust technology
- » .0001 Gb / run
- » £150k instrument
- » \$1,000,000 / Gb





# NGS

2005 - present



## Next Generation sequencing

Is massively parallel  
Not limited to few reactions per run





## Capillary sequencing



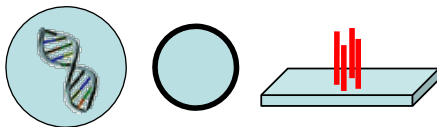
1 tube  
1 template



1 capillary  
1000 bases



## Next-Generation Sequencing



1 feature  
1 template



1 chip, thousands or  
millions of features  
Output Mb-Tb





454



454

» Started NGS (2005)

» First massively parallel sequencer

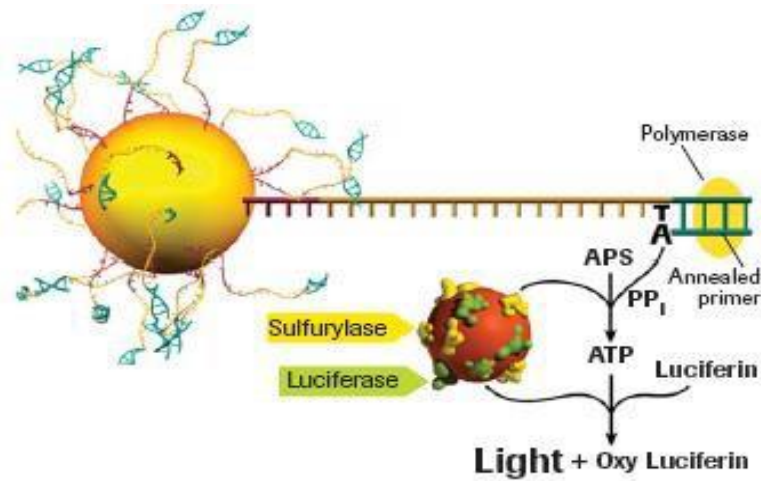
» Bought by Roche in 2007

» Based on pyrosequencing of bead-bound DNA in microwells





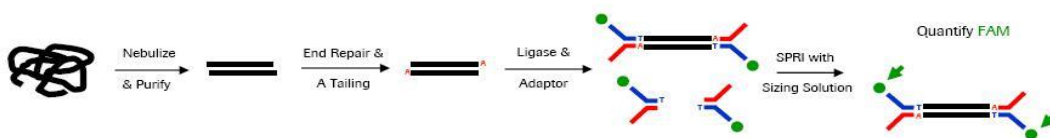
# Pyrosequencing



- » The incorporation of new bases releases inorganic pyrophosphate
- » A chemical cascade converts luciferin to oxy-luciferin + light



# 454 Shotgun Library Prep



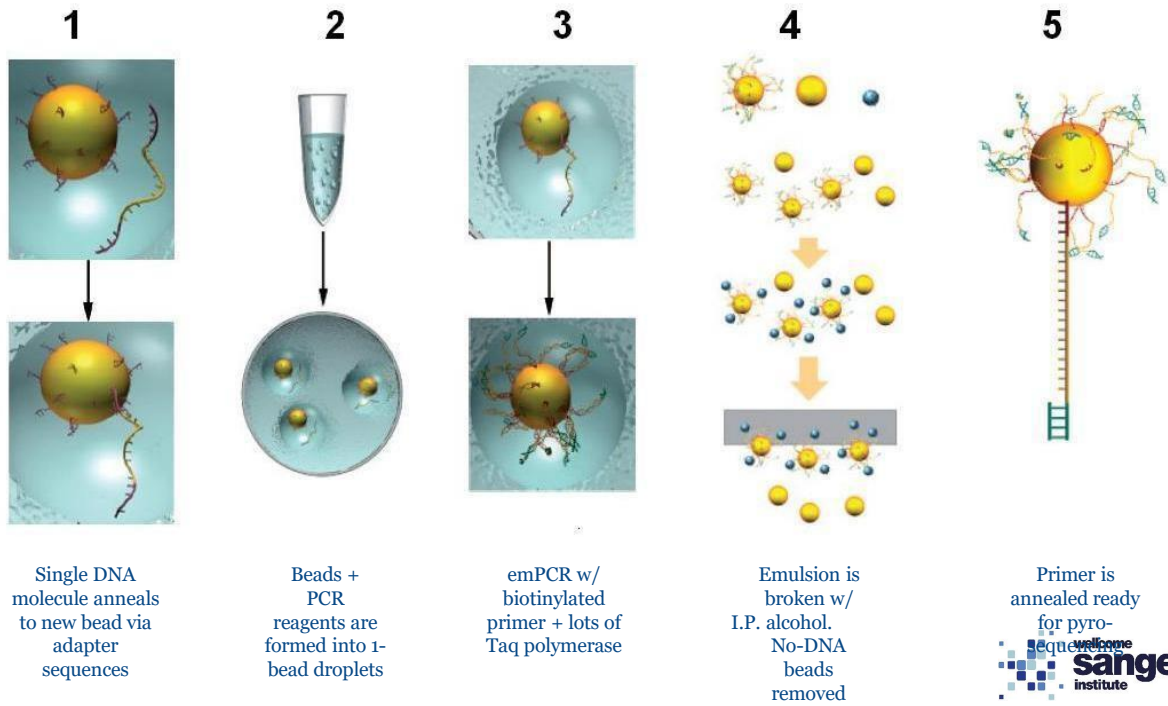
- Standard Library Preparation in 17 Steps**
1. Calibrate SPRI Beads
  2. Nebulize 3,000 ng
  3. Column Purification
  4. SPRI Cleanup
  5. DNA 7500 Chip
  6. End Repair
  7. Column Purification
  8. Ligation
  9. Column Purification
  10. SPRI Cleanup
  11. Bead Immobilization
  12. Fill-In Reaction
  13. Melt
  14. Column Purification
  15. RiboGreen Assay
  16. RNA 6000 Chip
  17. Calculate Molecules/ul based on Size and Mass

- Rapid Library Preparation in 6 Steps**
1. Nebulize 500 ng
  2. Column Purification
  3. End Repair and A Tailing
  4. Adaptor Ligation
  5. SPRI with Sizing Solution
  6. Quantify with FAM Standard

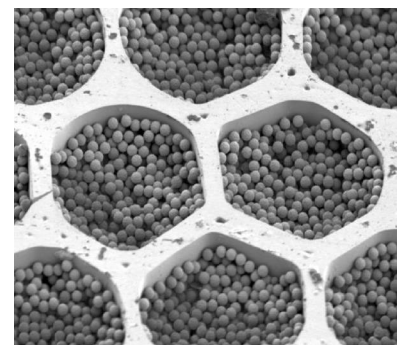
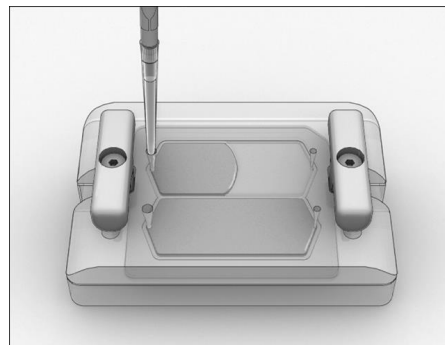
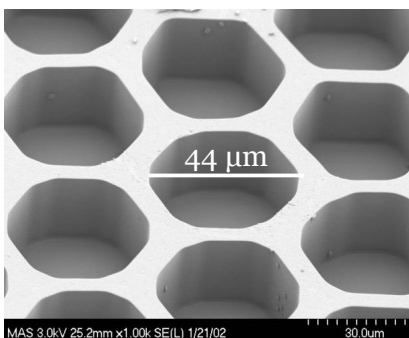
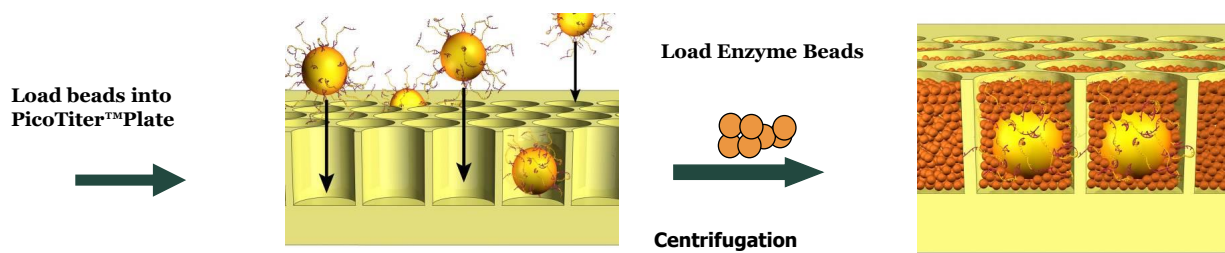




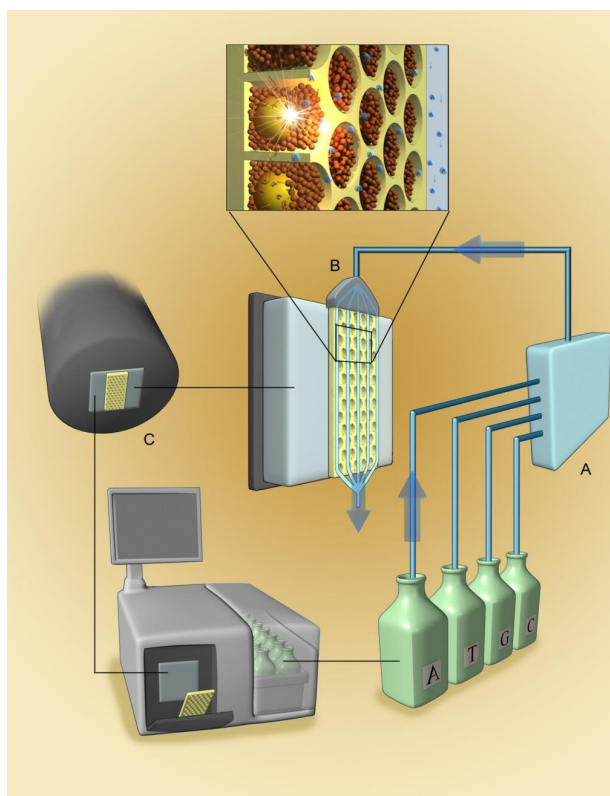
# 454 Emulsion PCR



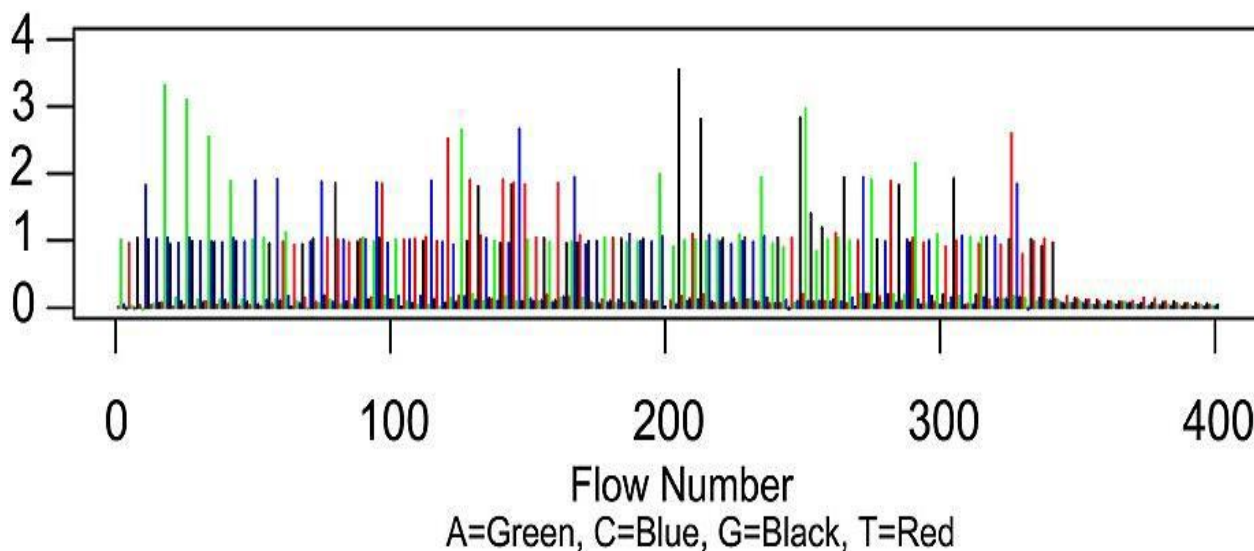
# Depositing DNA Beads into the PicoTiter™ Plate



# 454 Technology - Sequencing Instrument



# 454 Data Example





## 454/Roche Summary

- » Long read lengths – good for amplicons and *de-novo* sequencing
- » High error rate near homopolymers
- » Fast turnaround
- » Emulsion PCR
- » Single end only



## 454/Roche Summary

- » .7 Gb / run
- » 700 base reads
- » <24 hour run time
- » \$7,000 / Gb



Roche announced will discontinue in 2016



# Illumina Sequencing Technology

Michael Quail  
mq1@sanger.ac.uk

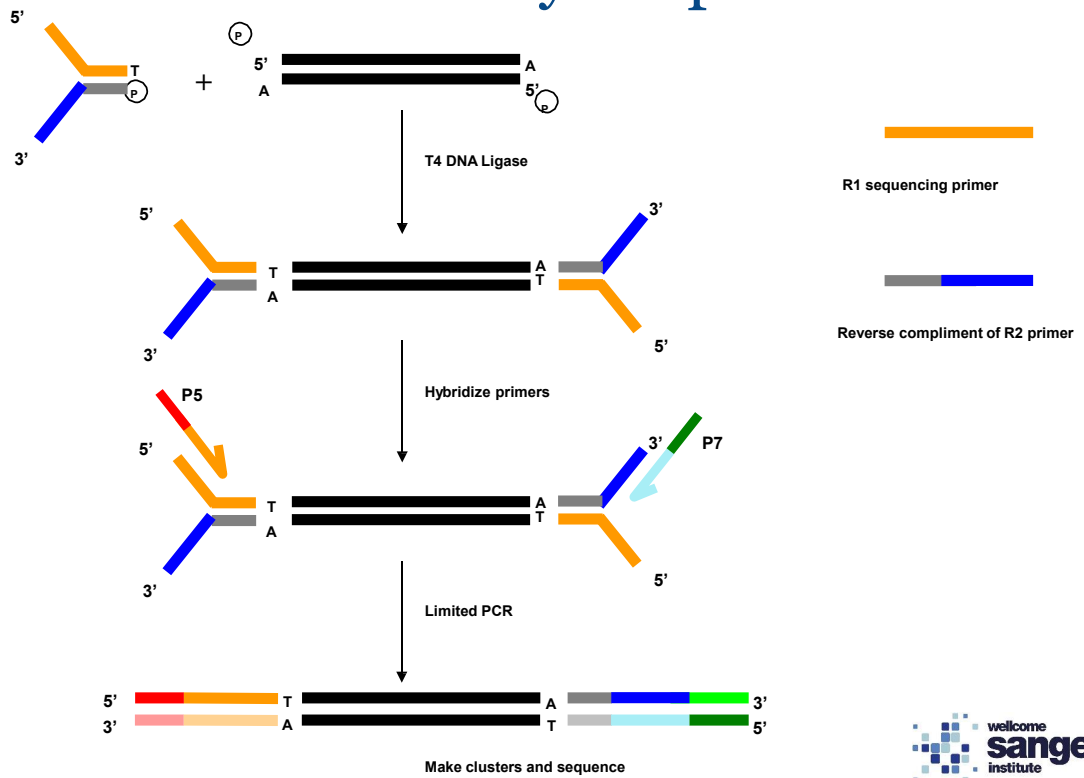


## Solexa

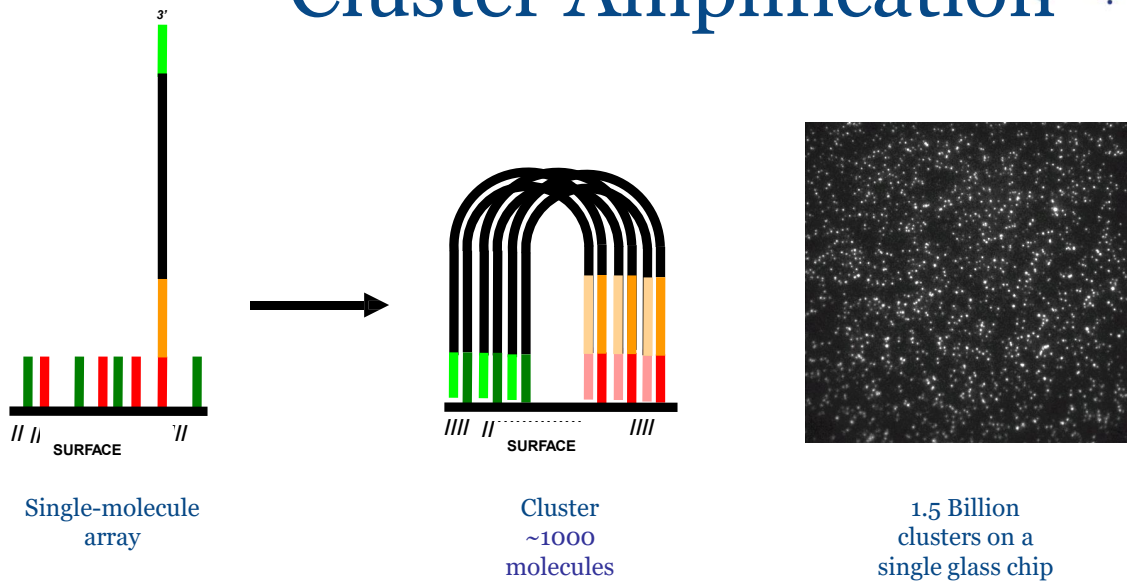
- » Launched their Genome Analyzer in 2006
- » Spinout from Cambridge University, set up at Gt. Chesterford in 2000
- » Genome Analyzer; 1Gb/run
- » Acquired by Illumina in 2007



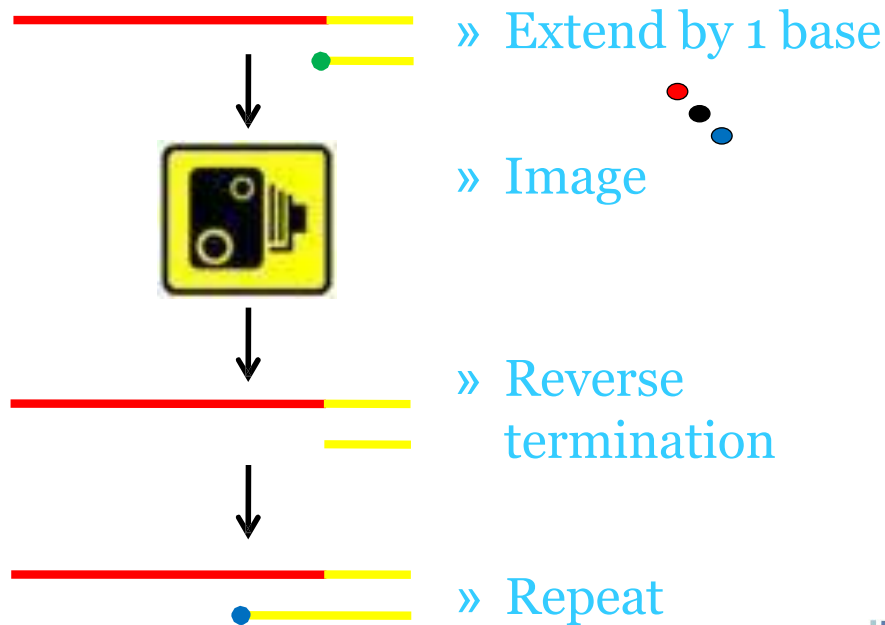
# Illumina Paired End Library Prep



# Cluster Amplification

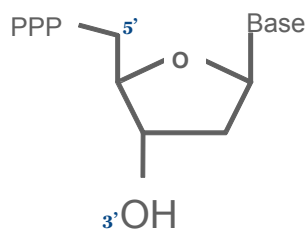


# Sequencing by Synthesis

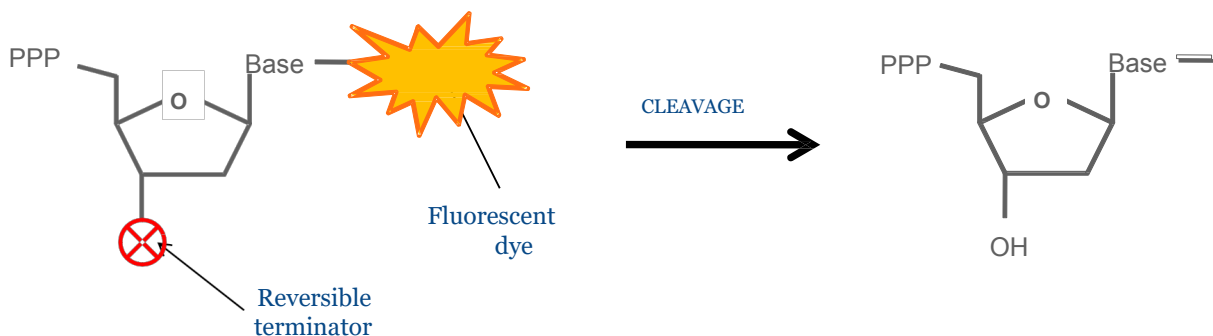


# Illumina Modified Nucleotides

Natural dNTP:



Illumina modified NTP:



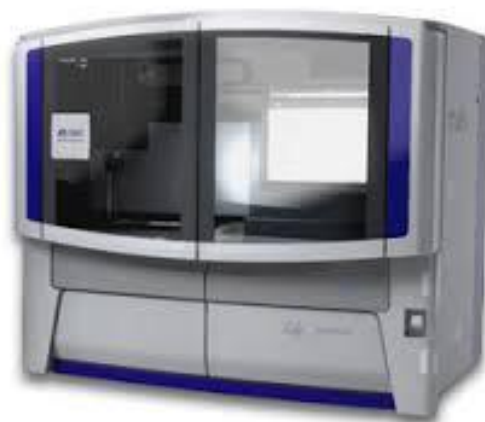


# Illumina

- » Cheap \$6-\$30/Gb
- » Highly accurate data mostly Q30
- » Massively parallel. Millions of reads
- » Short read

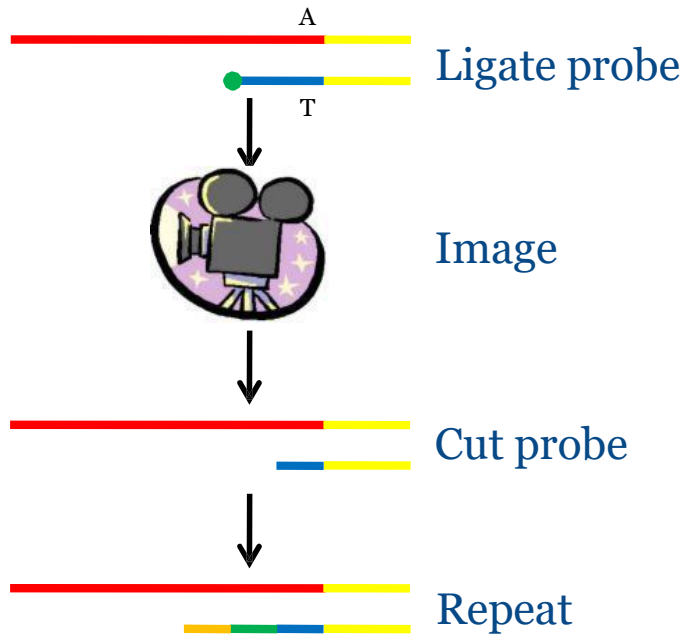


# Life Technologies SOLiD

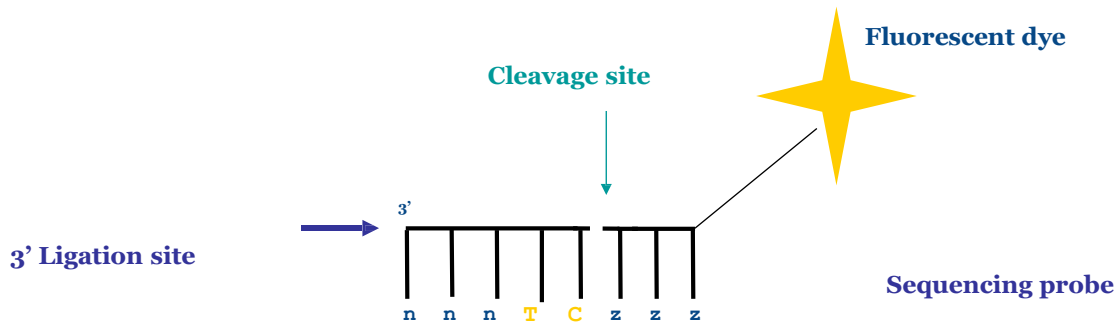




# Sequencing by Ligation



## ABI SOLiD



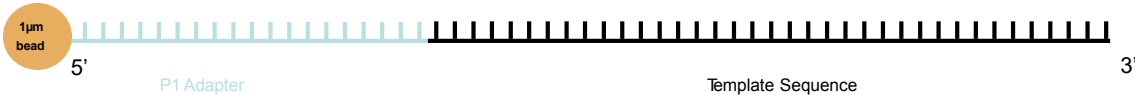
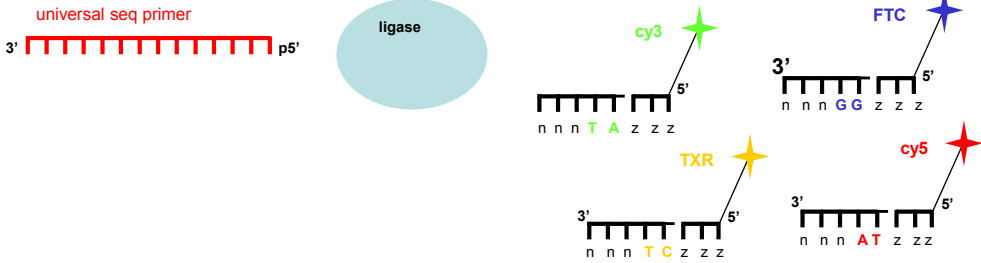
4 Dyes, 256 probes per dye (1,024 total)

n= degenerate bases

z= universal bases

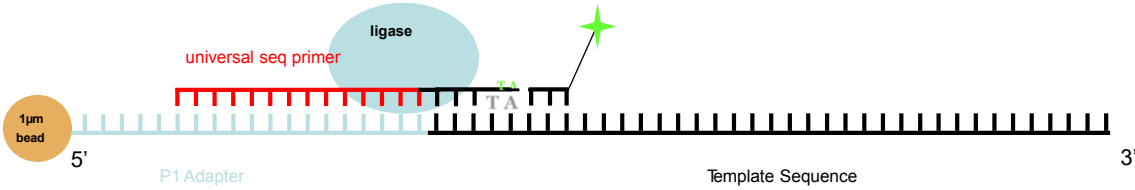
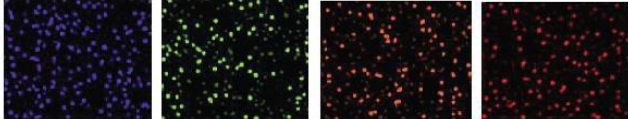


# Ligation-Based Chemistry

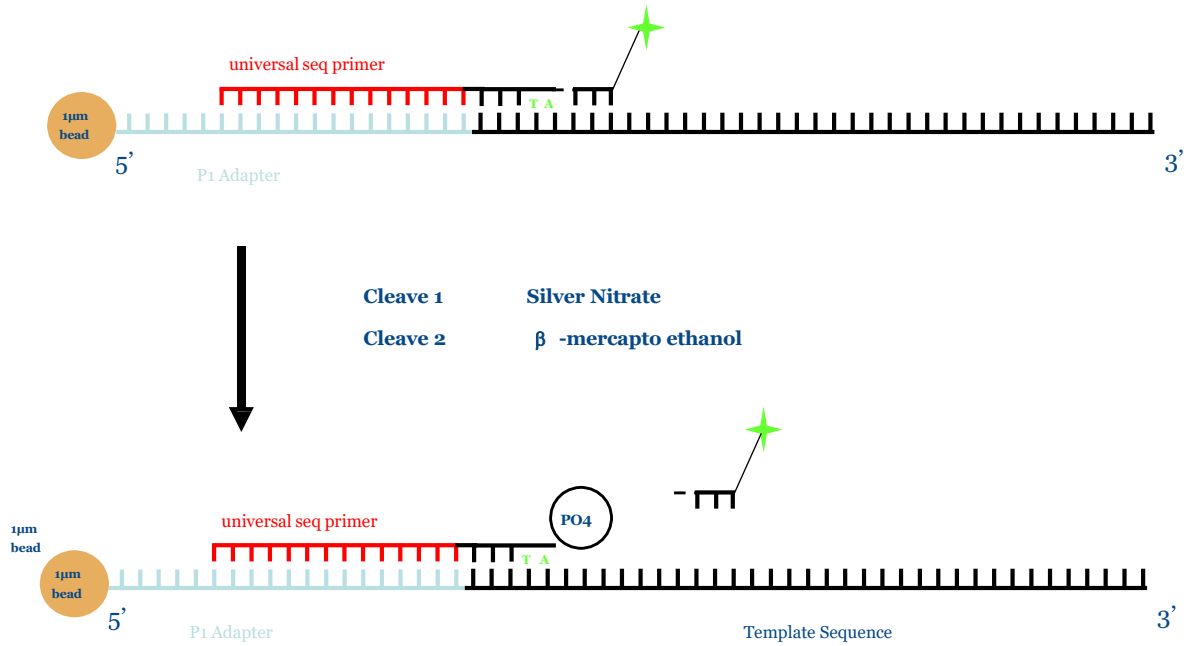


# Ligation

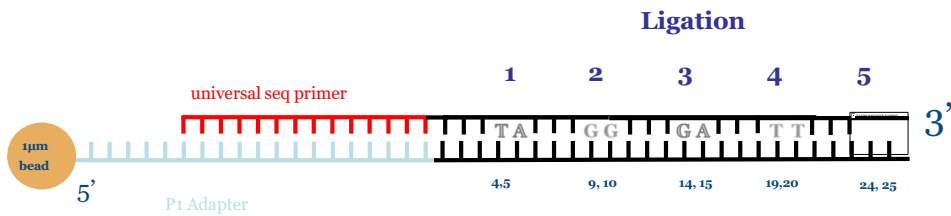
## Imaging



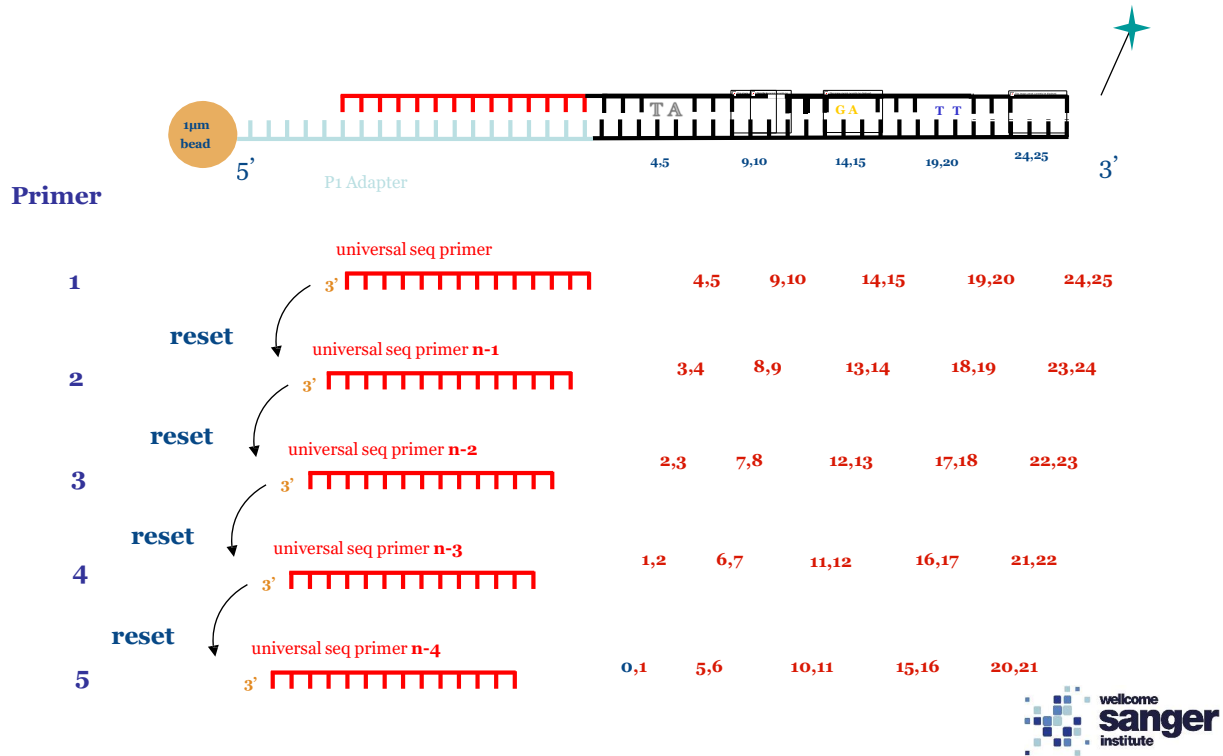
# Cleavage



## 4 More Ligations Completes First Primer sequence

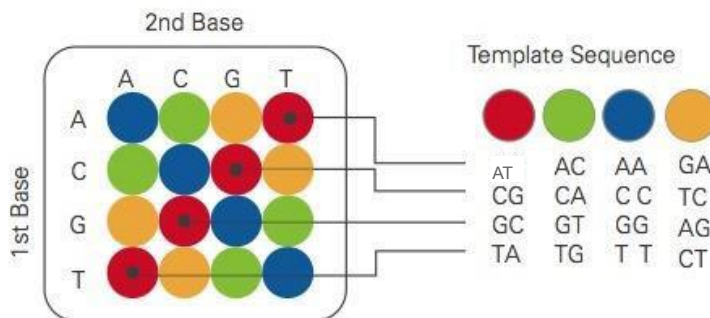


# 4 More Probe sets completes sequence



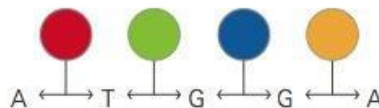
# Colour Space 2 base encoding

Possible Dinucleotides Encoded By Each Color



Double Interrogation

With 2 base encoding each base is defined twice





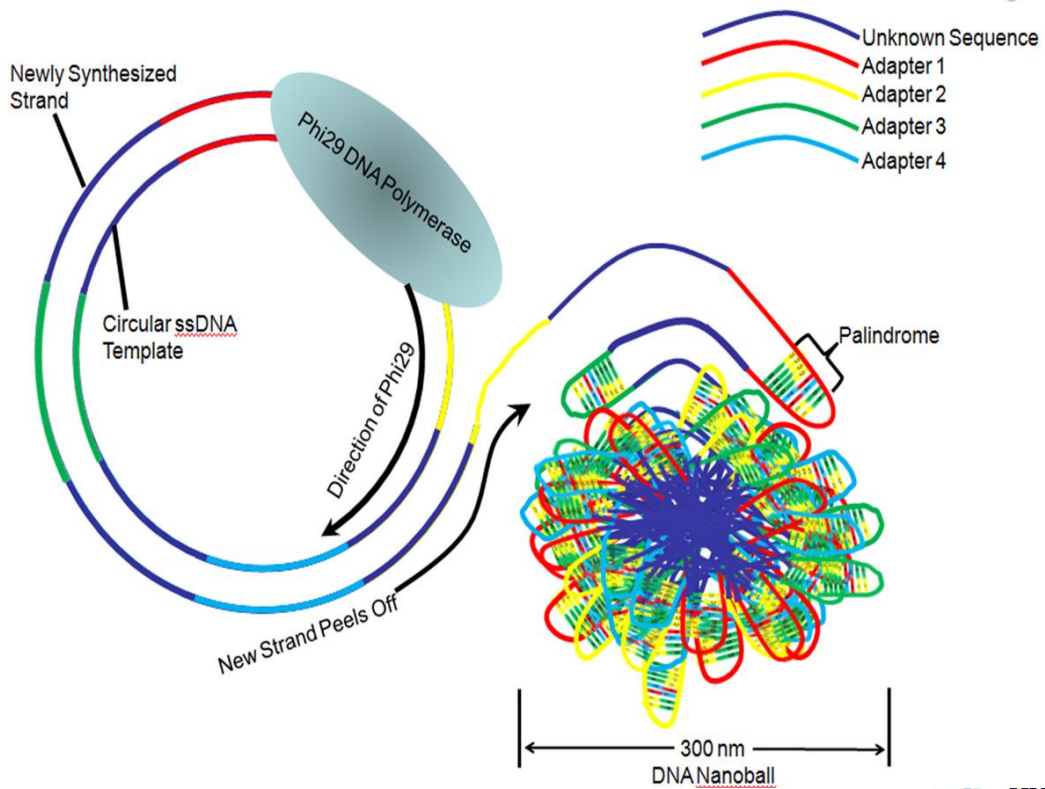
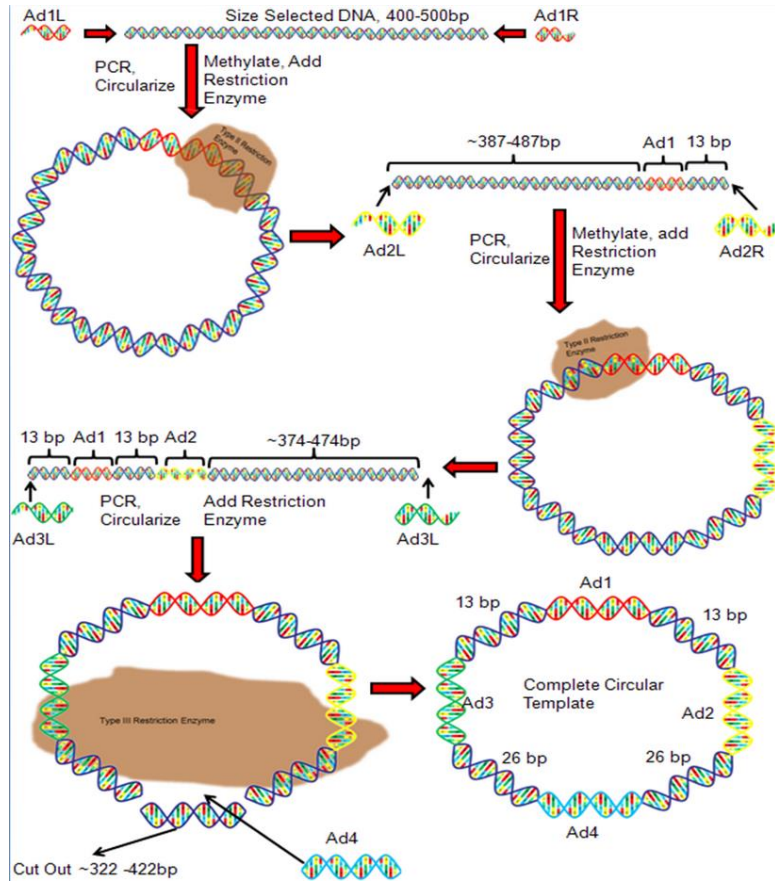


# Complete Genomics



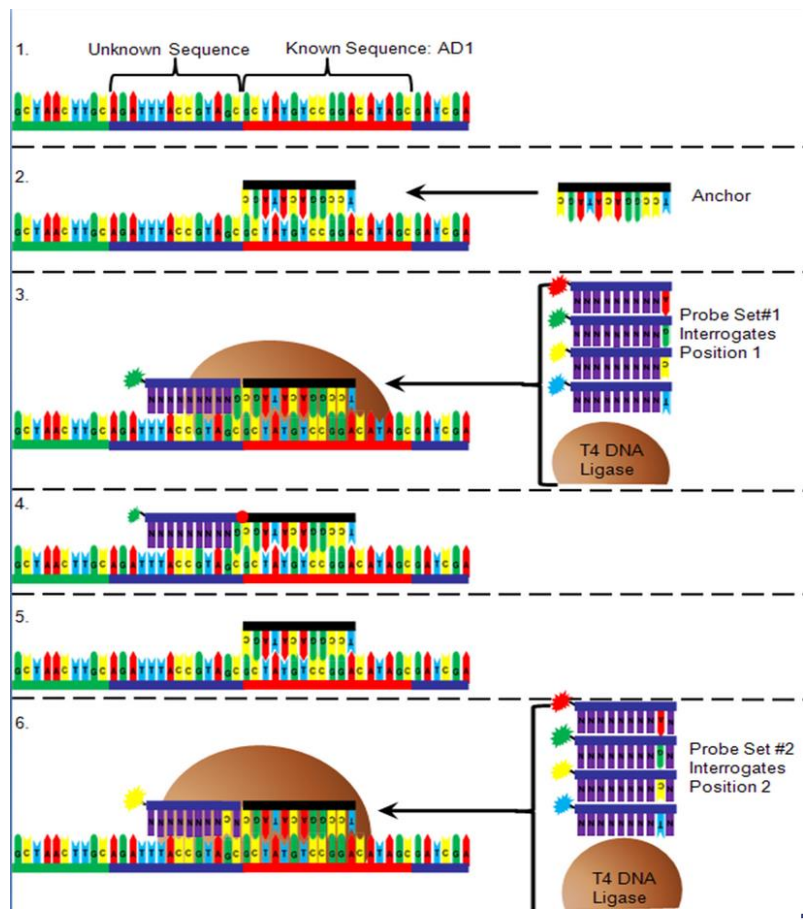
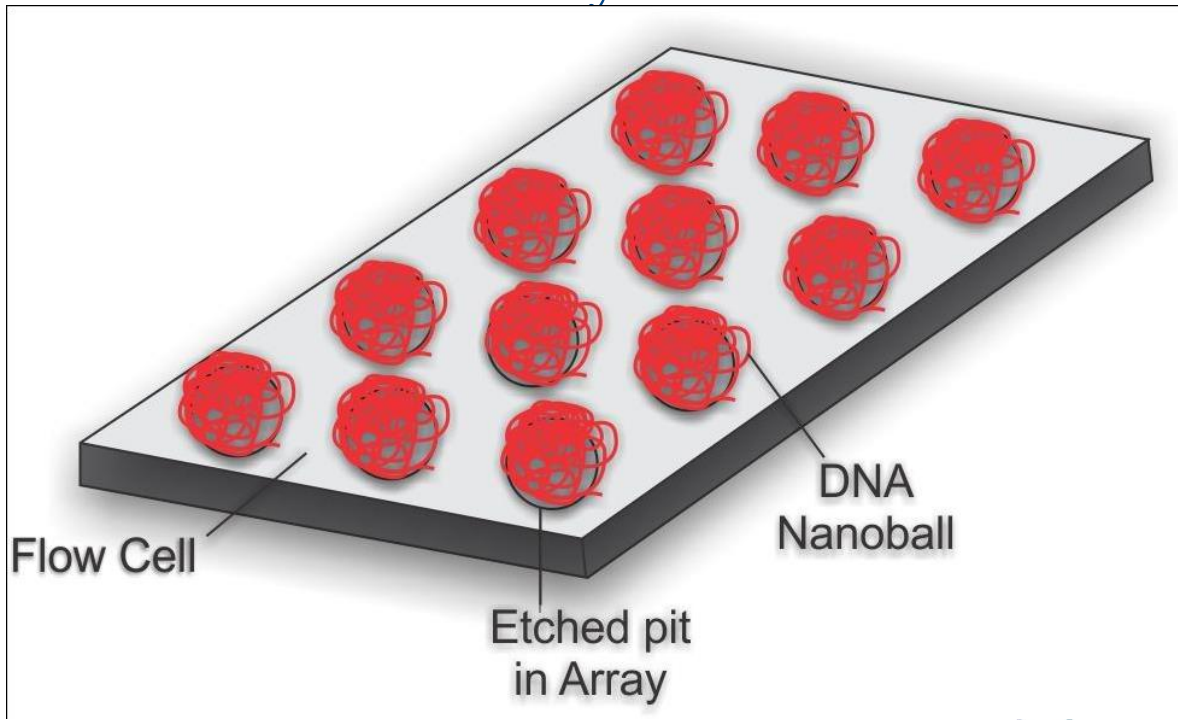
# Complete Genomics





# Complete Genomics

## Nanoball arrays on flowcell

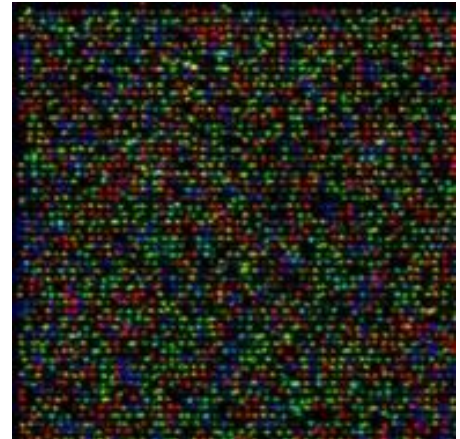




# Complete Genomics



- » DNA nano-balls, ordered array
- » Ligation-based sequencing
- » 210 Gb / slide (18 slides)
- » 35–base paired reads
- » 1 week run time
- » \$50 / Gb (finished)



# Complete Genomics



- » Raw error rate 0.5%
- » Consensus accuracy  $10^{-5}$
- » Short read lengths
- » Service business model
- » 500 Genomes/month cap.
- » Only 40x human genomes
  - » ~\$5,000 per genome





# Complete Genomics Revolocity



- » 10,000 genomes/year
- » \$12M
- » 10-120 samples at a time
- » Sample to answer
- » 8 day turnaround
- » 96% genome coverage
- » 1 error in a million (raw 0.5%)
- » 300 bp insert. 2 x 28bp reads



# Complete Genomics BGISEQ-500



- » 8-200 Gb output
- » 2 chips. CG nanoball sequencing
- » 100 base reads
- » 24 hour turnaround
- » Currently for Chinese market
- » High accuracy
- » Instrument 33% less than cost of a HiSeq
- » Competes with HiSeq on cost per Gb





# Pacific Biosciences



RSII : 1800 lbs. and ~11 feet long !



# Pacific Biosciences

- » \$750k
- » SMRT technology: Single Molecule Real Time
- » Yield 200-500Mb
- » Some reads 20Kb +
- » \$400 / Gb



SMRT cell

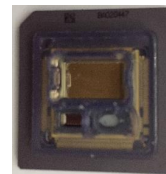




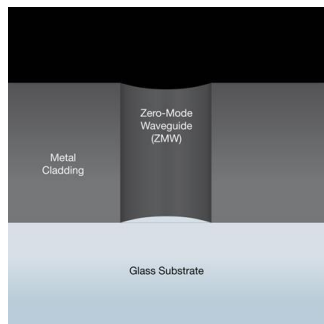
# Pacific Biosciences Sequel



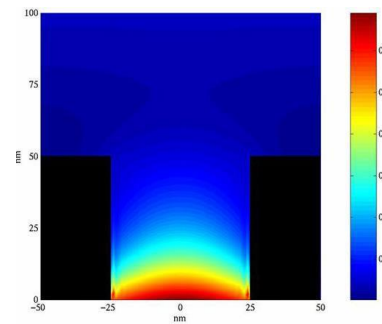
- » 2016 release
- » 1 million ZMWs/SMRT
- » 7Gb/SMRT cell
- » ~\$100/Gb
- » \$350K instrument



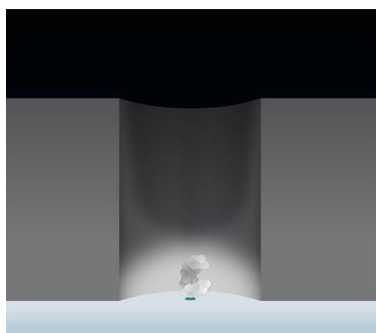
## Pac Bio Technology



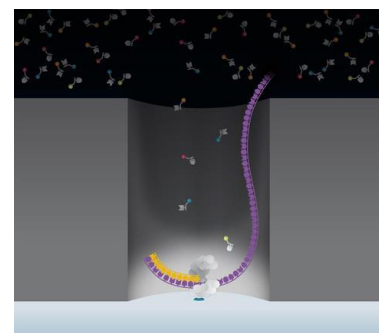
Individual ZMW



Laser light illuminates the ZMW



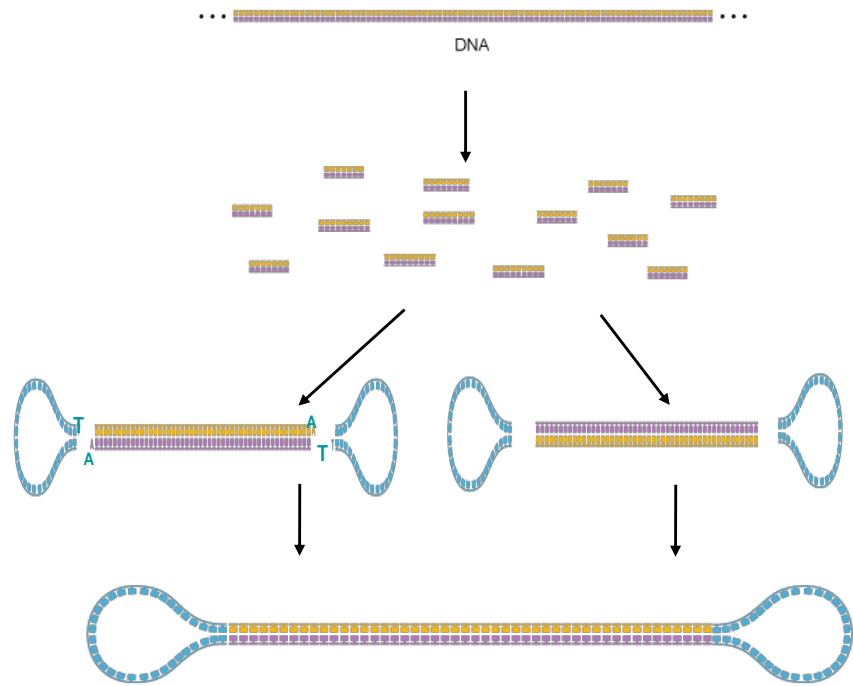
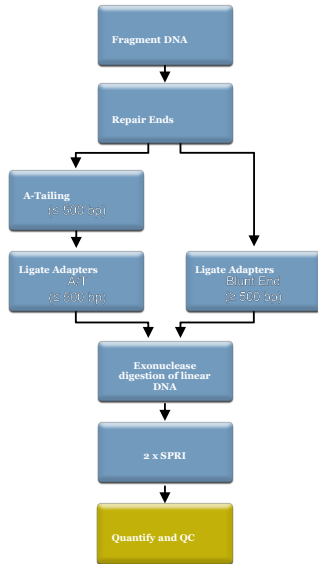
ZMW with DNA polymerase



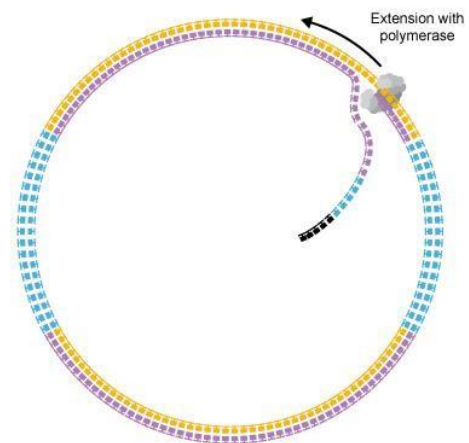
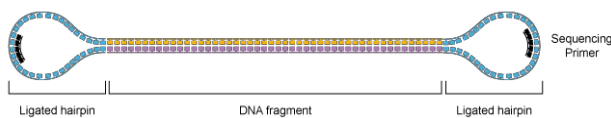
ZMW with polymerase + nucs.



# PacBio Library Prep



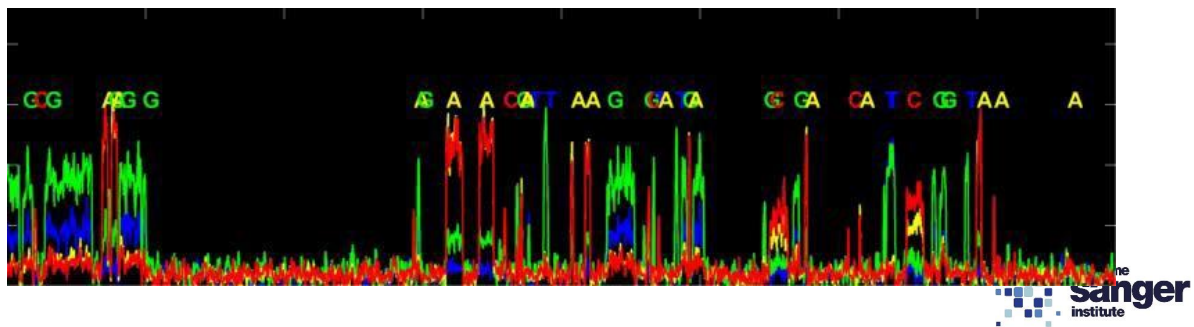
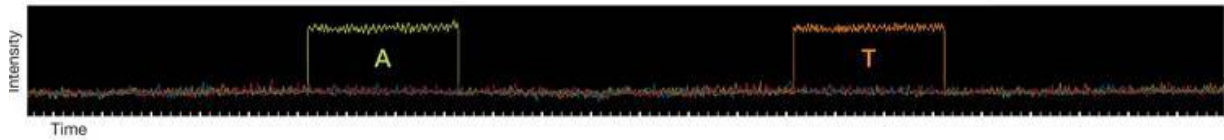
# PacBio Template Preparation



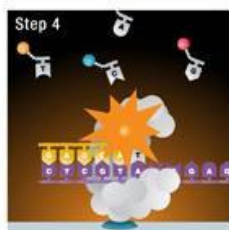
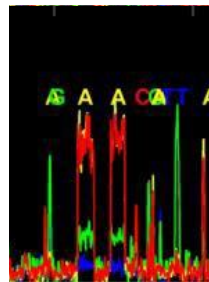
- 1** Anneal primer
- 2** Bind polymerase
- 3** Sequence



## PacBio Sequencing

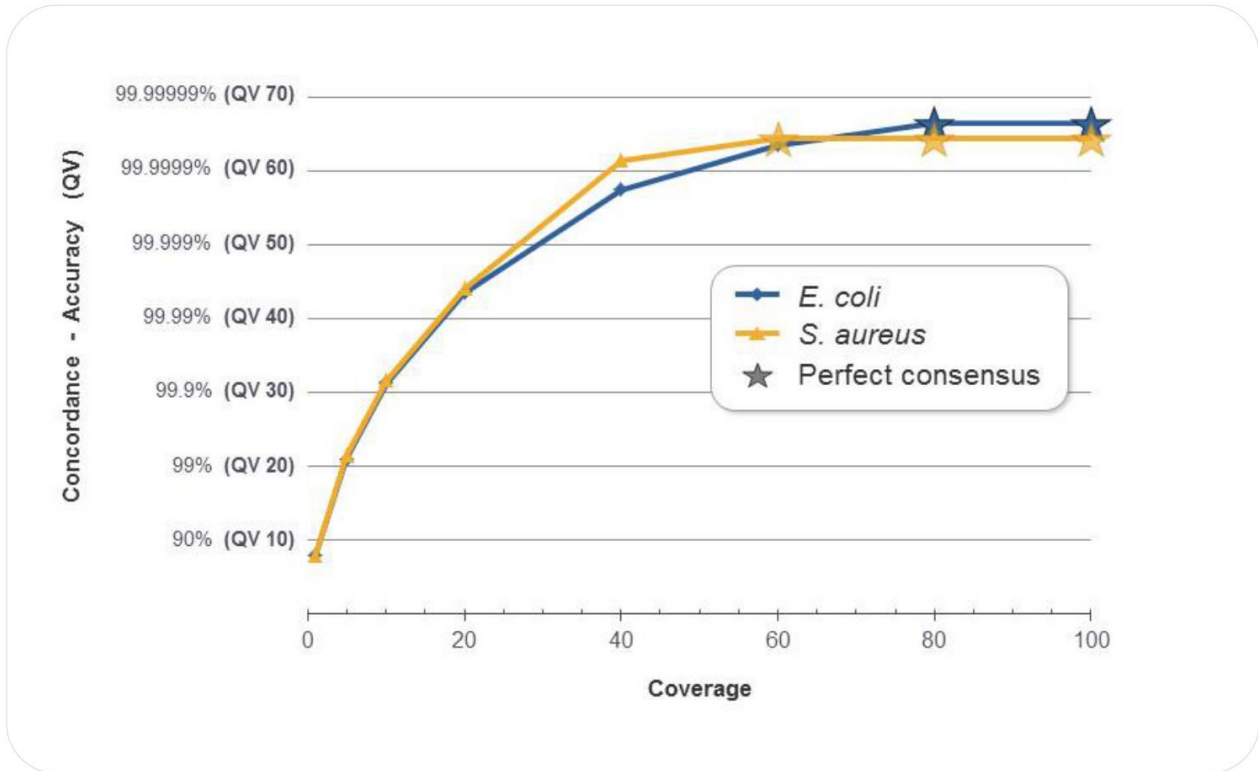


## PacBio Sequencing

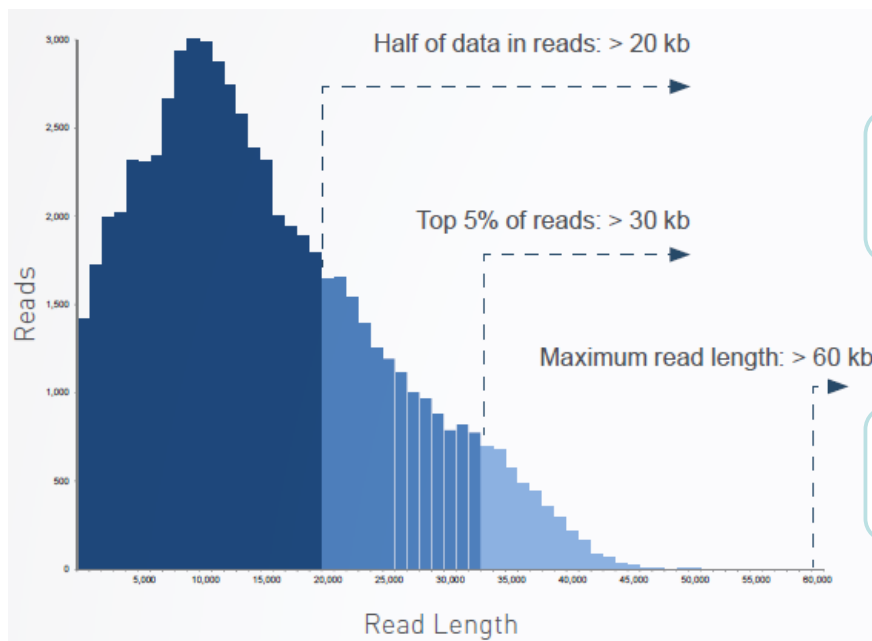


- » Some bases added very quickly and missed
- » Some wrong bases flirt with active site and go away
- » SMRT cell has 150,000 ZMWs
- » \$100 each

# Accuracy



## P6-C4 CHEMISTRY READ LENGTH PERFORMANCE (RS II)

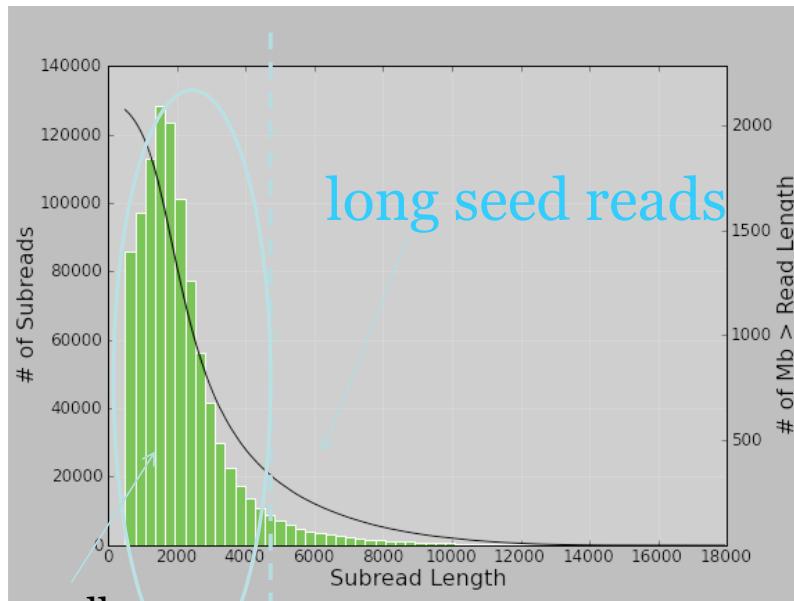


**Yield / SMRT Cell:** 0.5 – 1 Gb  
 20 kb size-selected library  
 4 hour movie  
 P6-C4 chemistry

**Users have reported:**  
 Max Read Lengths: >80 kb  
 Yield / SMRT Cell: >1.5 Gb



# HGAP - Pre-assembly



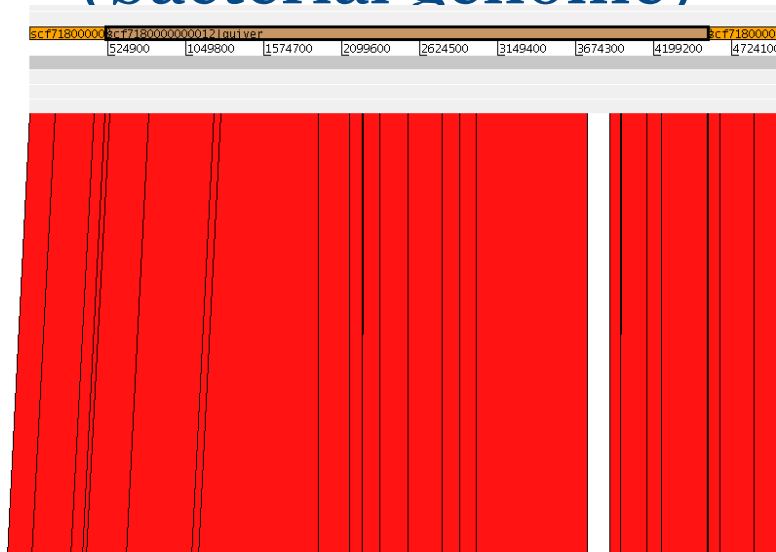
correct with smaller reads

pre-assembled to 20K accurate reads  
mean, mode, median, all around 6 kb



# Example assembly (bacterial genome)

pacbio

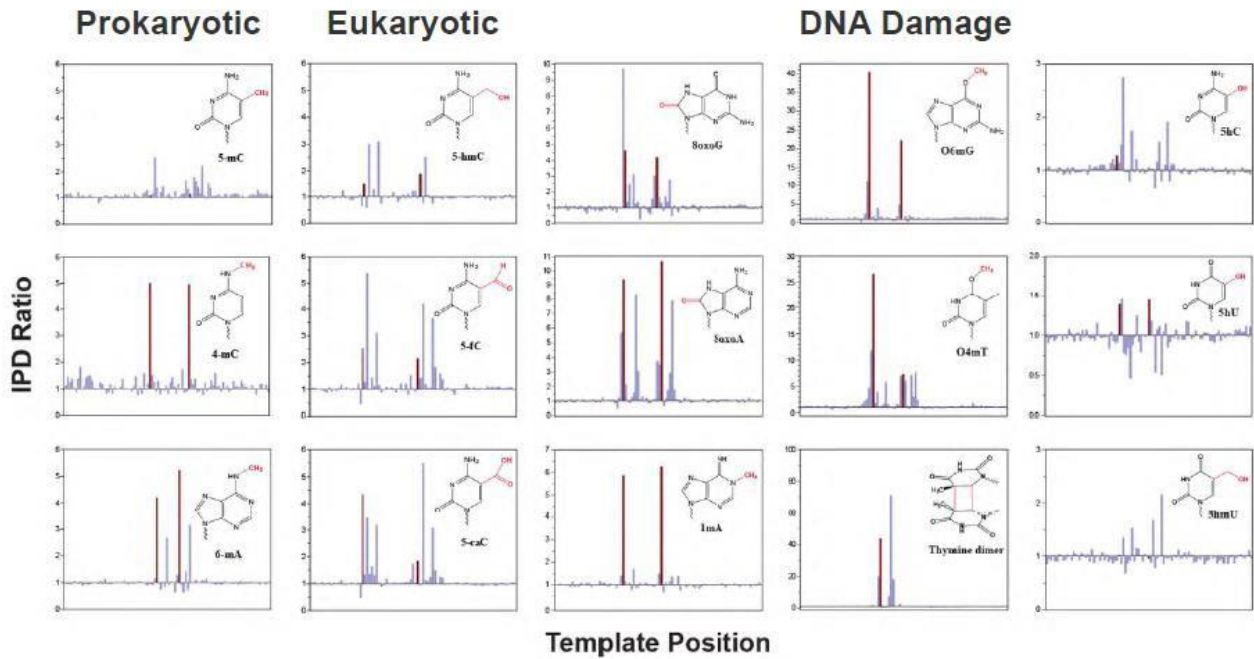


illumina



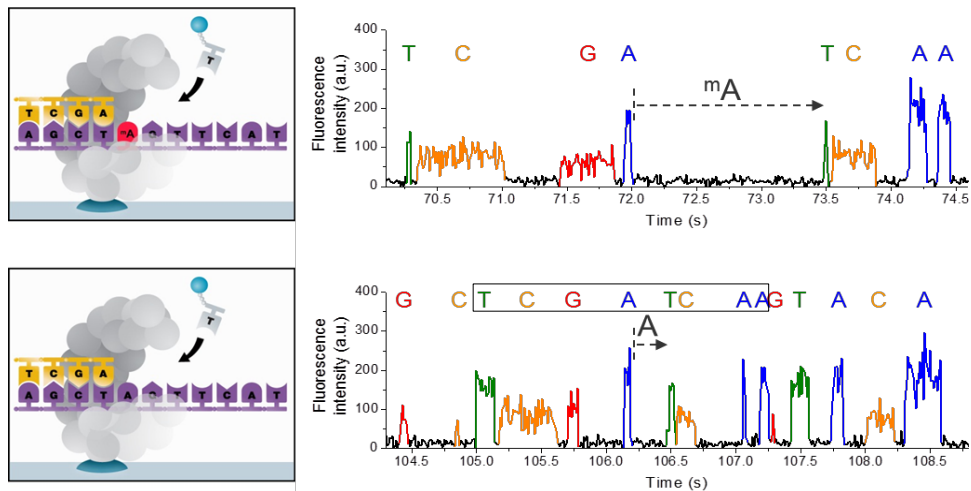


# Base Modifications



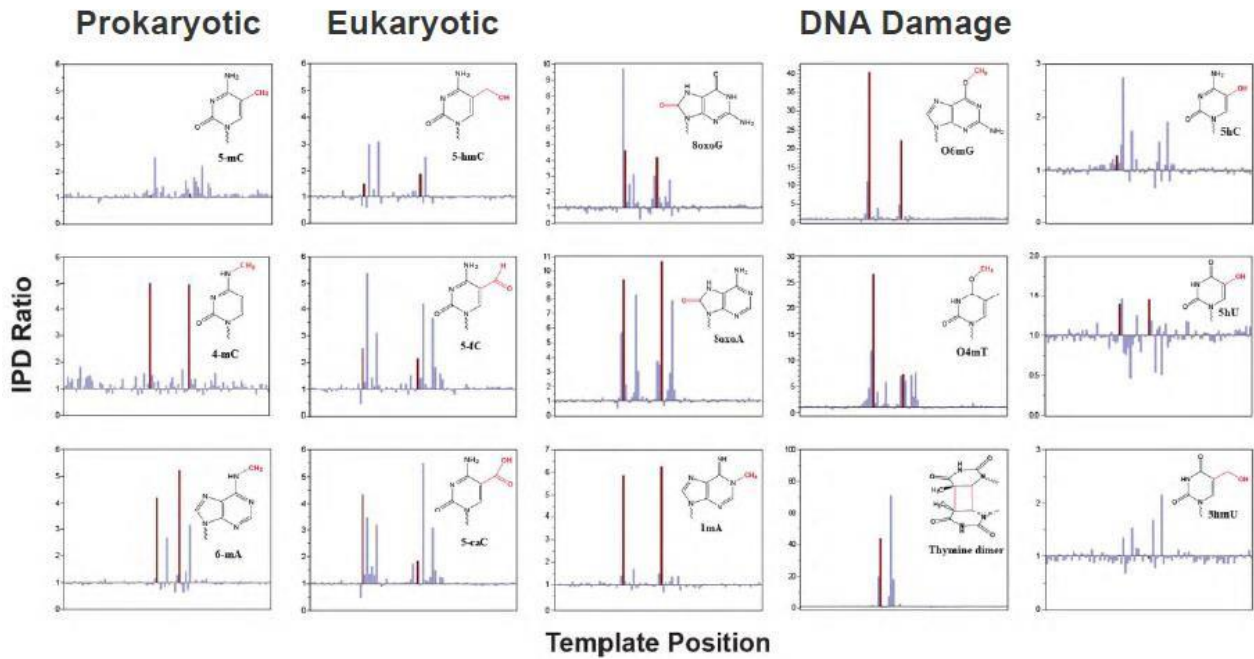
## DETECTION OF DNA BASE MODIFICATIONS USING KINETICS

### Example: N<sup>6</sup>-methyladenine

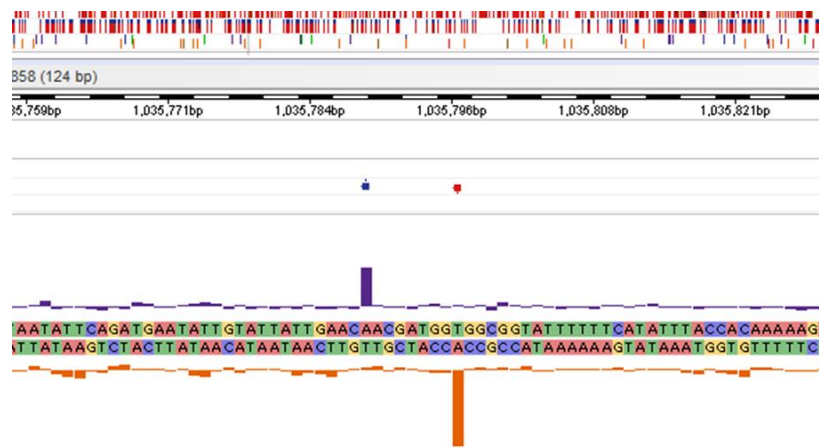


- SMRT Sequencing uses kinetic information from each nucleotide addition to call bases
- This same information can be used to distinguish modified and native bases by comparing results of SMRT Sequencing to an *in silico* kinetic reference for incorporation dynamics without modifications.

# Base Modifications



An example of signal strength with m6A



# Pacific BioSciences Applications



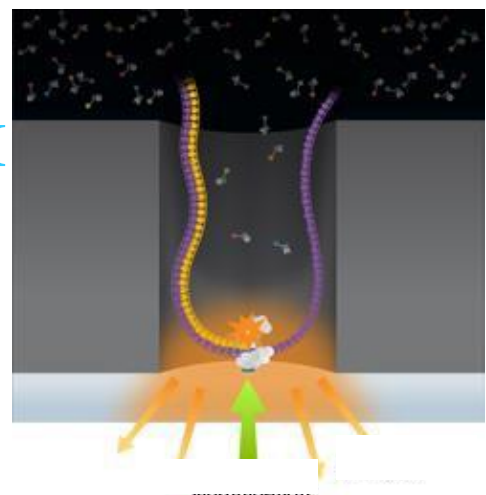
- » Long read applications
  - » De Novo sequencing
  - » Full length cDNA sequencing
  - » Haplotyping
- » DNA modification studies



# Pacific Biosciences



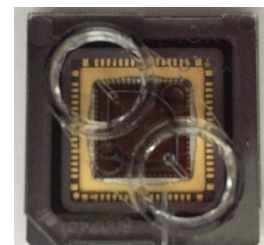
- » Single polymerase mol. in a 20nm hole
- » Watch incorporation in real time
- » ~2 bases per second
- » Yield 200-500Mb on RSII
- » Yield upto 7Gb Sequel
- » Some reads 20Kb +



# Semiconductor Sequencing



# Ion Torrent's PGM

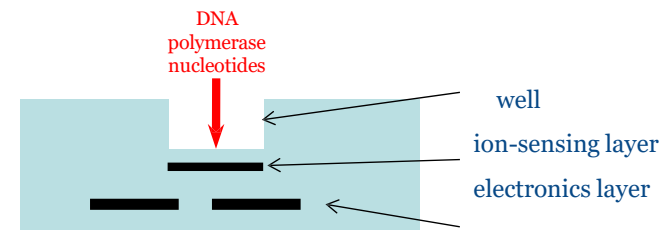


Capital cost \$50,000

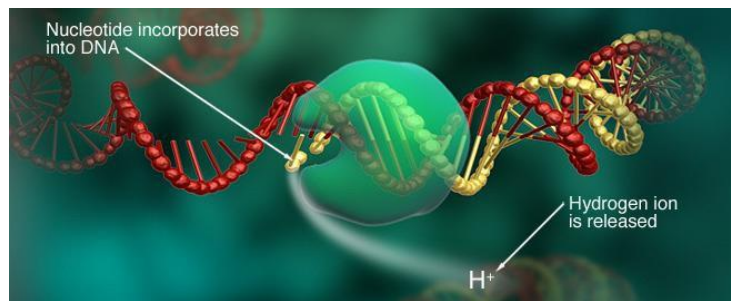




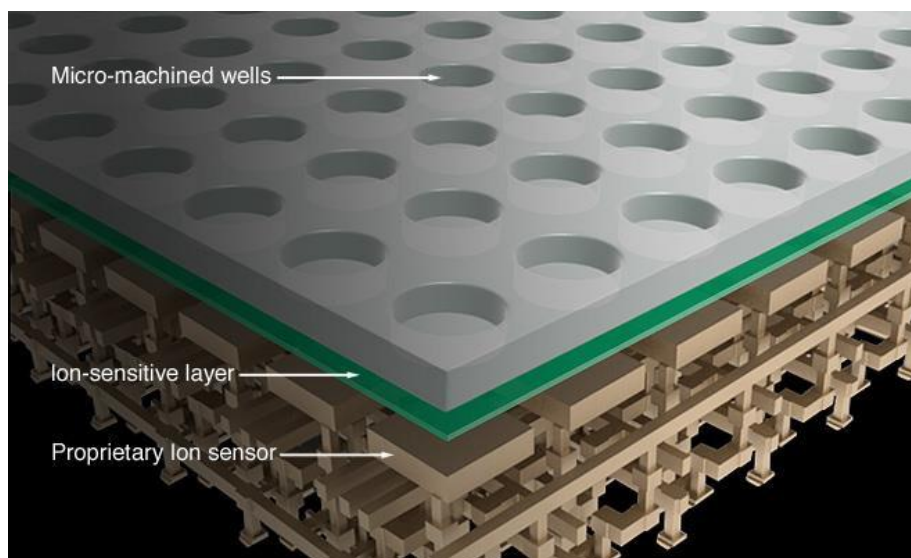
# Ion Torrent's Technology



Flow through A then T then ... like 454



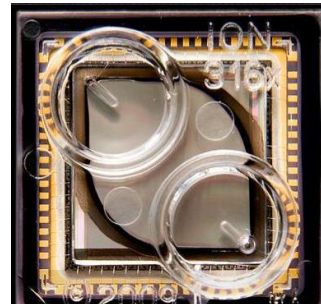
# Ion Torrent Chip Drawing





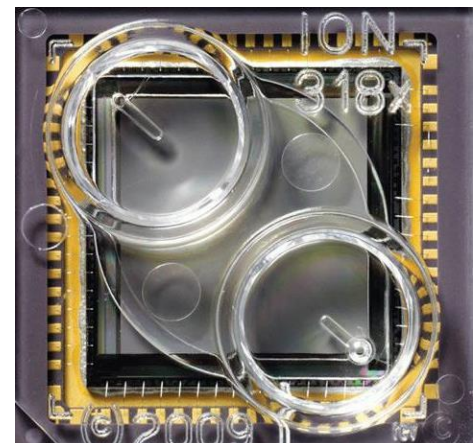
# Ion Torrent

- » Library prep like original 454
- » Amplification on beads by emPCR
- » CMOS chip detection
- » Cyclic addition sequencing - pH changes
- » Not single-molecule
- » Emulsion-PCR being replaced



# Ion Torrent 318 (PGM)

- » 11 million wells
- » 1 Gb / run
- » Read length ~400
- » Error rate 1-2%
- » 2-4 hr. run time
- » \$1000 / Gb (\$75K)





# Ion Proton

- » Proton I: 165 million wells
- » 10 Gb / run
- » Read length ~100
- » Error rate 1-2%
- » 2 hr. run time
- » \$100 / Gb (\$250K)



## Ion AmpliSeq™ technology: As Simple As PCR Your Targets, Your Genome, Your Panel

---

The most comprehensive gene coverage  
with the lowest amount of DNA or RNA Input

### Simple

- 10 ng of DNA per pool
- FFPE-compatible
- PCR-based target selection



### Scalable

- Up to 24,000 primers per pool
- 1–1000s of genes
- 96 barcodes for multiplexing



### Fast

- 1 day from DNA to results
- 2 hours to design custom panels
- 3.5 hours for target selection and library preparation



# Ion GeneStudio S5 Series | Flexible Portfolio Configurable to Your Needs

Ion GeneStudio™ S5



**Fast.**

Ion GeneStudio™ S5 Plus



**Flexible.**

Ion GeneStudio™ S5 Prime



**Powerful.**



**Ion 510™ Chip**  
2–3 M reads  
Up to 400 bp



**Ion 520™ Chip**  
3–6 M reads  
Up to 600 bp



**Ion 530™ Chip**  
15–20 M reads  
Up to 600 bp



**Ion 540™ Chip**  
60–80 M reads  
Up to 200 bp



**Ion 550™ Chip**  
100–130 M reads  
Up to 200 bp

For Research Use Only. Not for use in diagnostic procedures. \* Throughputs based on 200bp sequencing

# Output and Turn-Around Time to Meet Your Lab's Peak Volume Needs

Ion GeneStudio™ S5



Ion GeneStudio™ S5 Plus



Ion GeneStudio™ S5 Prime

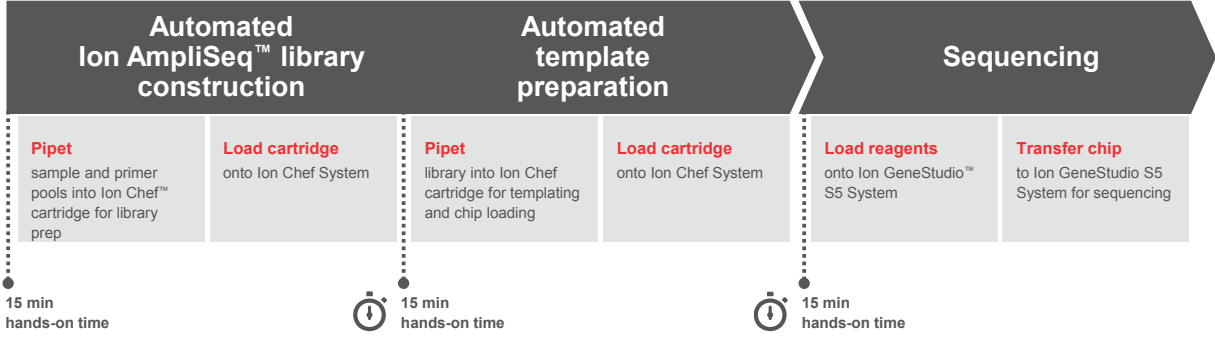


<b>Speed*</b>	19 hrs	10 hrs	6.5 hrs
<b>Output (max/day):</b>	15 Gb/80 M	30 Gb/160 M	50 Gb/260 M
<b>Chips (max/day):</b>	1 x 540	<u>2 x 540</u> or 1 x 550	2 x 550

\* Based off 540 chip – sequencing (2.5 hours) and analysis (varies) time

For Research Use Only. Not for use in diagnostic procedures.





For Research Use Only. Not for use in diagnostic procedures.

## Coming Soon: Ion AmpliSeq HD Technology

### Technology exclusively available for Ion Torrent™ customers

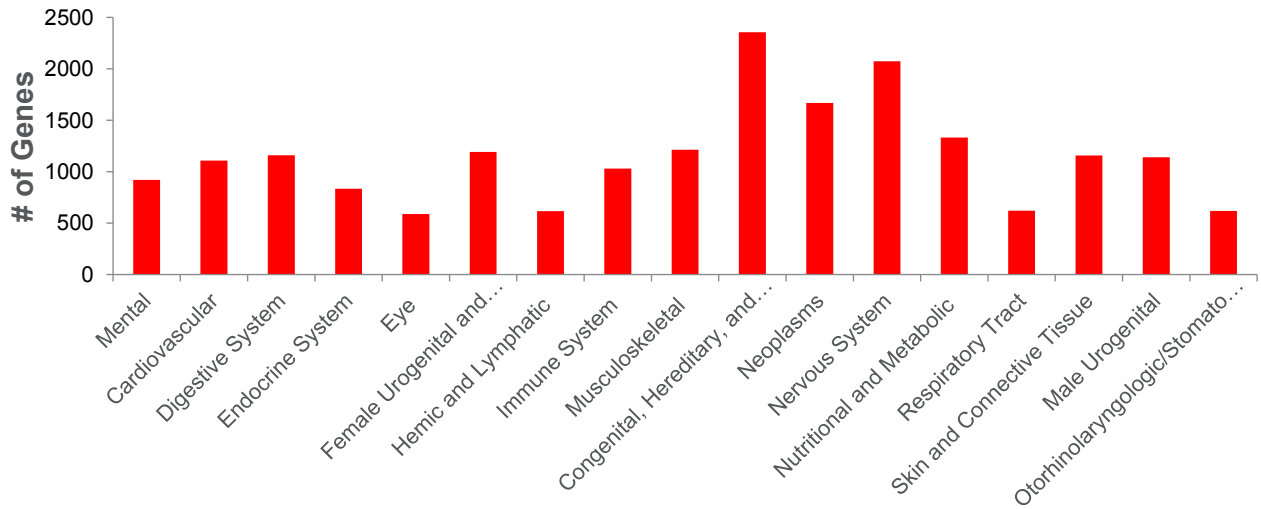
Novel core technology with the ability to process mixed or challenging sample types—extendable to multiple applications



The content provided herein may relate to products that have not been officially released and are subject to change without notice.

For Research Use Only. Not for use in diagnostic procedures.

### Expanded Gene Content Across Disease Research Areas



Average of 1154 genes per major UMLS disease research area category

For Research Use Only. Not for use in diagnostic procedures.

### List prices 2018

- Ion GeneStudio S5™ System A38194 Runs 510, 520, 530 and 540 chip. **50,528 GBP**
- Ion GeneStudio S5™ Plus System A38195 Runs 510, 520, 530, 540 and 550 chip. **104,942 GBP**
- Ion GeneStudio S5™ Prime System A38196 Runs 510, 520, 530, 540 and 550 chip. **132,150 GBP**
- Ion Chef(TM) System 4484177 **45,240 GBP**

All instruments include 12 months warranty

Promo: Trade-in any current NGS or CE instrument for up to 50% discount.

## World's first complete Sample to Insight NGS solution

Designed to deliver actionable insights



96

GeneReader NGS System PROM 10696 000 1000

ThermoFisher  
SCIENTIFIC

## Qiagen Gene Reader

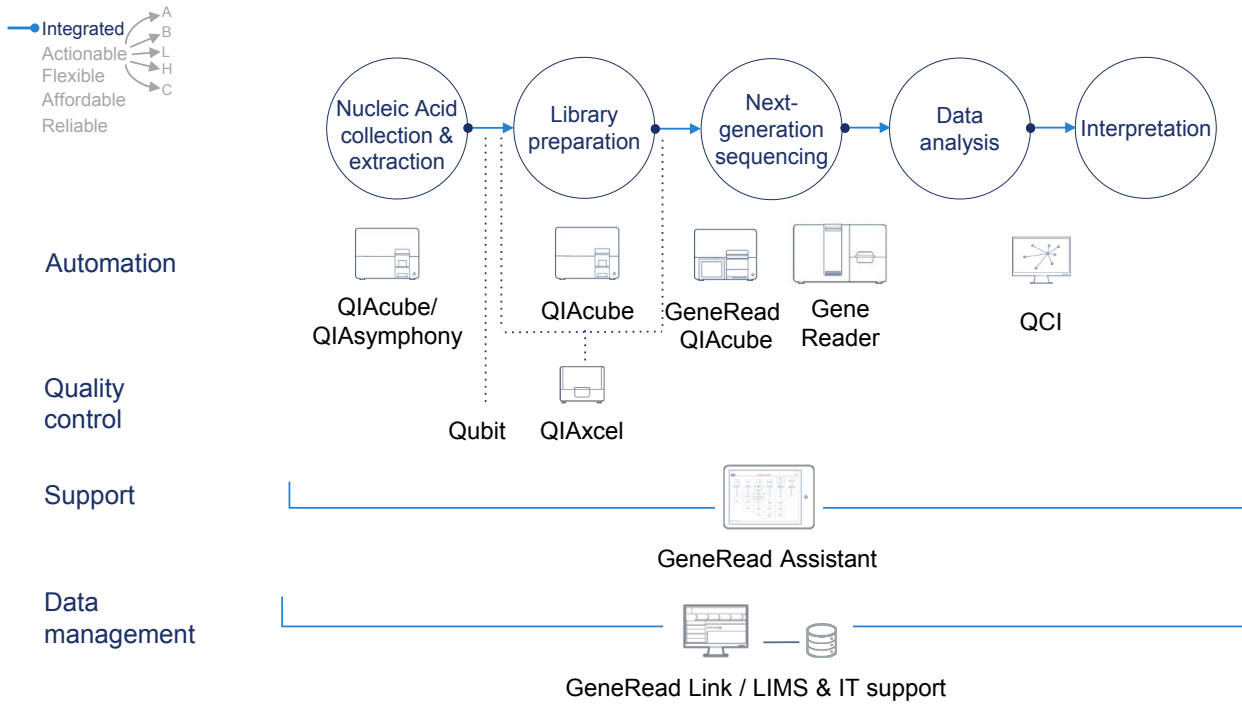
- SBS chemistry developed by intelligent biosystems
- Sequencing system that can prep DNA, do targeted library prep and run upto 20 x 1Gb yield flowcells
- Sequencing by synthesis but one base at a time



97

ThermoFisher  
SCIENTIFIC

# First truly complete NGS solution



Offering everything needed from one vendor for NGS solutions

## Best amplicon & insight coverage, plus best batching flexibility



- Integrated
- Actionable
- Flexible
- Affordable
- Reliable

	GeneRead Actionable Insights Tumor Panel	ILMN Tumor 15 Panel	AmpliSeq™ HotSpot Tumor Panel V2
<b>Panel size</b>	12 genes / 16.7 kb	15 genes / 44 kb	50 genes / Not provided
<b>Insight size</b>	773 unique variant positions	Not available	2855 variants
<b>Amplicons</b>	330	250	207
<b>Average amplicon size</b>	134 bp	150–175 bp	154 bp
<b>DNA input</b>	10 ng x 4	10 ng x 2	10 ng x 1
<b>Throughput</b>	1–48 samples per run	8 samples per run	2/8/16 samples per chip
<b>Variant frequency</b>	5%	5%	5%
<b>Amplicon coverage</b>	>500x: 96.4% >200x: 98.5% Average: ~5000x	>500x: 93.5% >200x: not available Average: n/a	>500x: not available >200x: not available Average: 1000-4000x
<b>Variant insight coverage</b>	>500x: 99.8% >200x: 99.9%	Not available	Not available

Note: Coverage is average of 12 NA12878 datasets

# The ultimate objective of NGS in molecular pathology research

Sample Metadata		Laboratory Information		Specimen	
Sample ID	Sample 1	Lab	FacilityName	Specimen Type	Biopsy
Date of birth	Aug 5, 1967	Technician	Technician ID	Specimen ID	Specimen ID
Ethnicity	African	Scientist	Scientist ID	Collection Date	Jan 1, 2015
Sex	Male			Accession Date	Jan 3, 2015
Accession	1somatic_max_api_test			Primary Tumor Site	Blood Vessel
				Diagnosis	stage 42 glioblastoma

Summary of Clinically Significant Variants				
Variants Reported	FDA Approved Therapies for Indication	FDA Approved Therapies for Other Indications	Therapies Associated with Resistance	Potential Clinical Trials
<b>EGFR</b> p.E746_A750del				2 potential trials

Gene	Exon #	Nucleotide Change	Amino Acid Change	Effect on Protein
<b>EGFR</b>	19	NM_005228.3: c.2235_2249delGG	p.E746_A750	Gain of Function

EGFR is an oncogenic receptor tyrosine kinase involved in cell survival by regulating the RAS and PI3K pathways. Gain of function mutations and amplification cause EGFR activation. Somatic mutations have been reported in 36% of NSCLCs, 8.6% of gliomas and 2% of breast cancers [PMID: 15938733]. EGFRvIII allele is associated with poor prognosis in patients with glioblastoma. EGFR amplification in patients with EGFR mutation-positive NSCLC is associated with increased resistance to osimertinib, erlotinib and gefitinib therapies. In over 40% of patients with NSCLC, EGFR amplification coexists with EGFR mutations [PMID: 17666241]. TKI-sensitive EGFR mutations are more common in female never-smokers and in patients with adenocarcinomas [PMID: 14555582].

Page 1 of 3  
QIAGEN | support.qiagen.com | QIAGEN.com

The QIAGEN GeneReader® is intended for research use only (RUO). This product is not intended for the diagnosis, prevention or treatment of a disease.

100

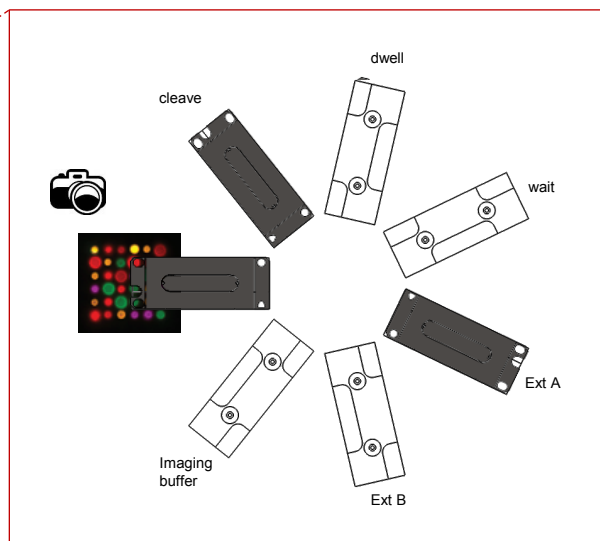
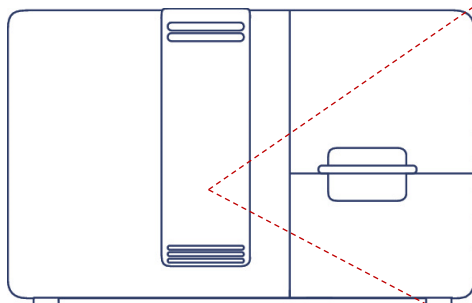
For Research Use Only not for use in diagnostic procedures



## Sequencer behaving like 3 sequencers in 1

- Integrated
- Actionable
- Flexible
- Affordable
- Reliable

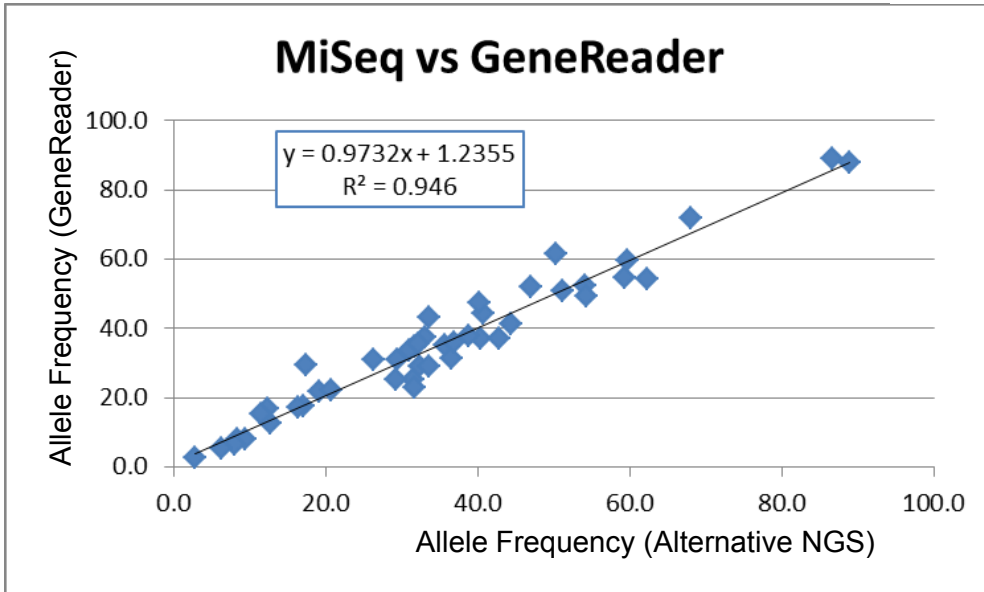
### Turntable design



Scaling up and down depending on number of samples

101

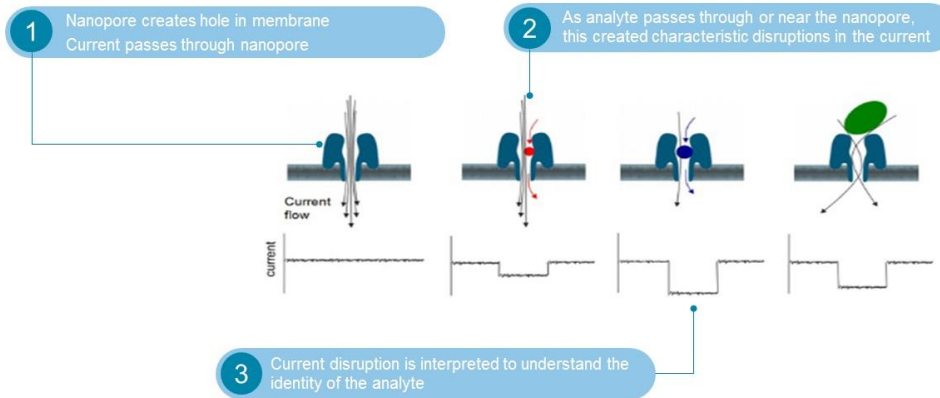




# Nanopores



## NANOPORE SENSING

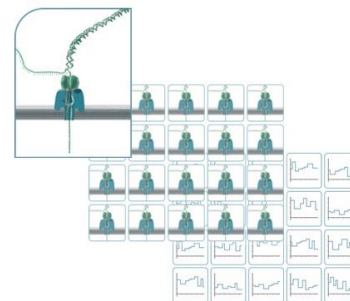
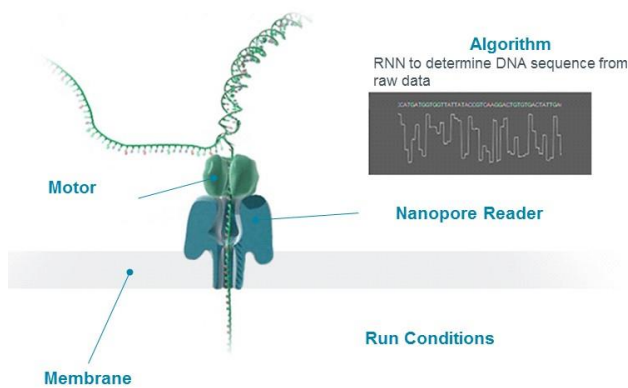


An explanation from Oxford **NANOPORE** Technologies™



## NANOPORE DNA SEQUENCING

How does it work?



Multiple nanopore sensors arrayed in one device

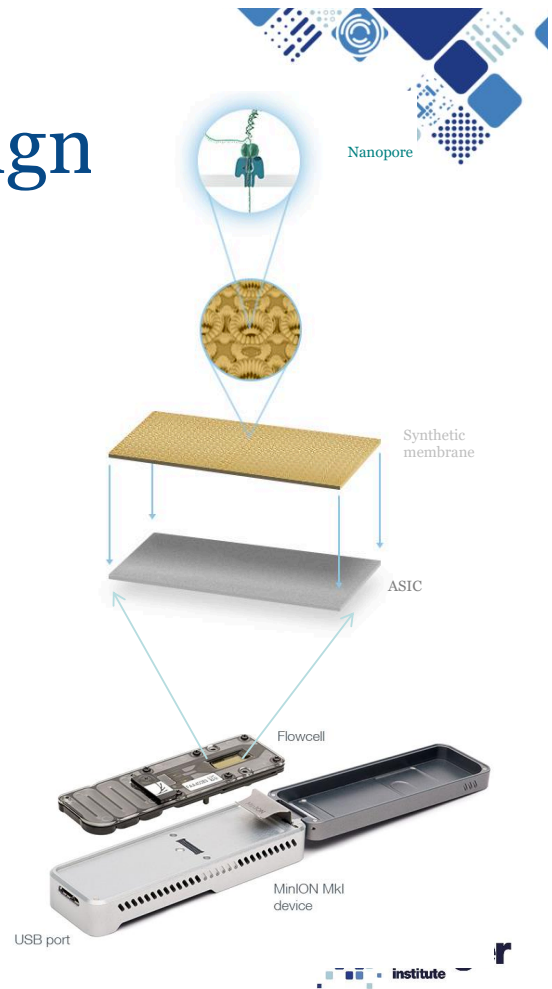
Operate independently but at the same time

An explanation from Oxford **NANOPORE** Technologies™

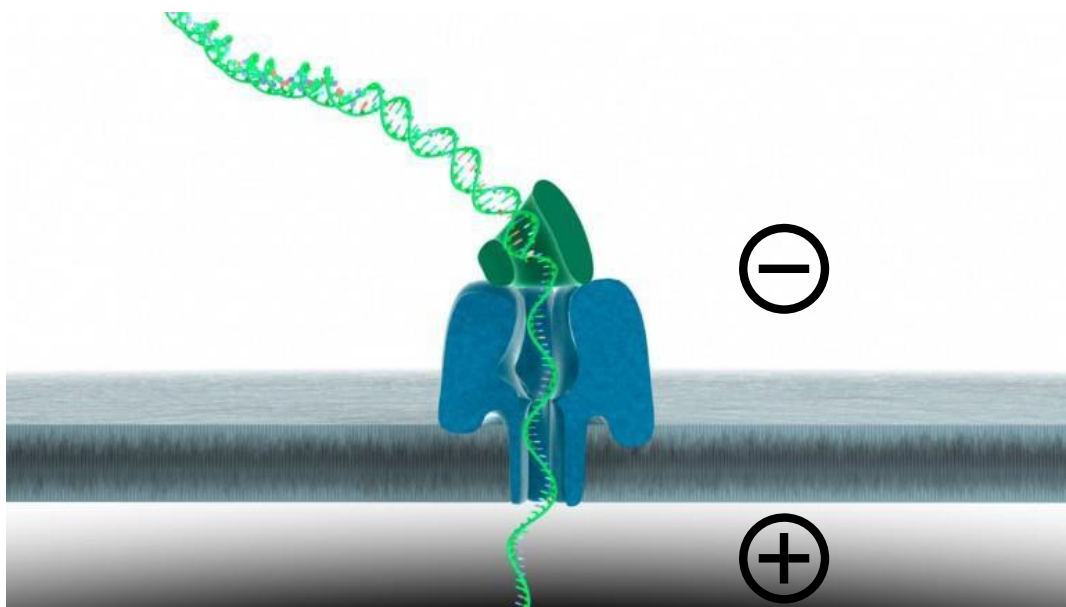


# Flow cell design

- » Application-Specific Integrated Circuits (ASICs) contains 512 channels
- » Each channel is surrounded by 4 pores & records only 1 at the tie
- » 512 pores max recorded at the time
- » Scan for “fresh” active pores automatically every 24h or when manual restarted



# Oxford Nanopore Technologies

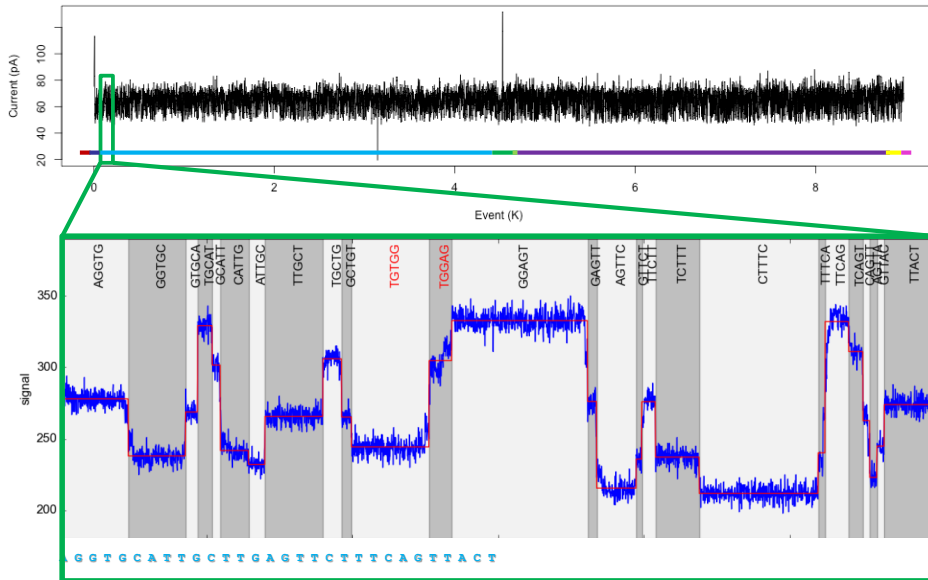


5-base words = 1024 current levels





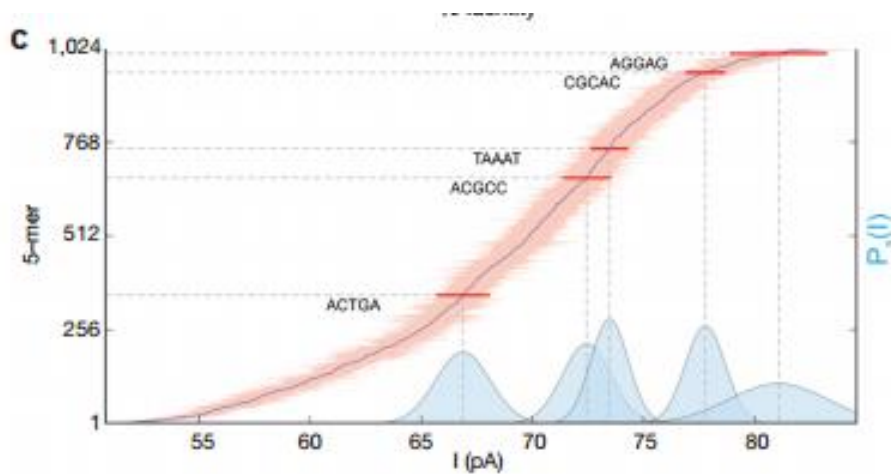
# ONT: The squiggles



Slide courtesy of David Buck. WTCGH



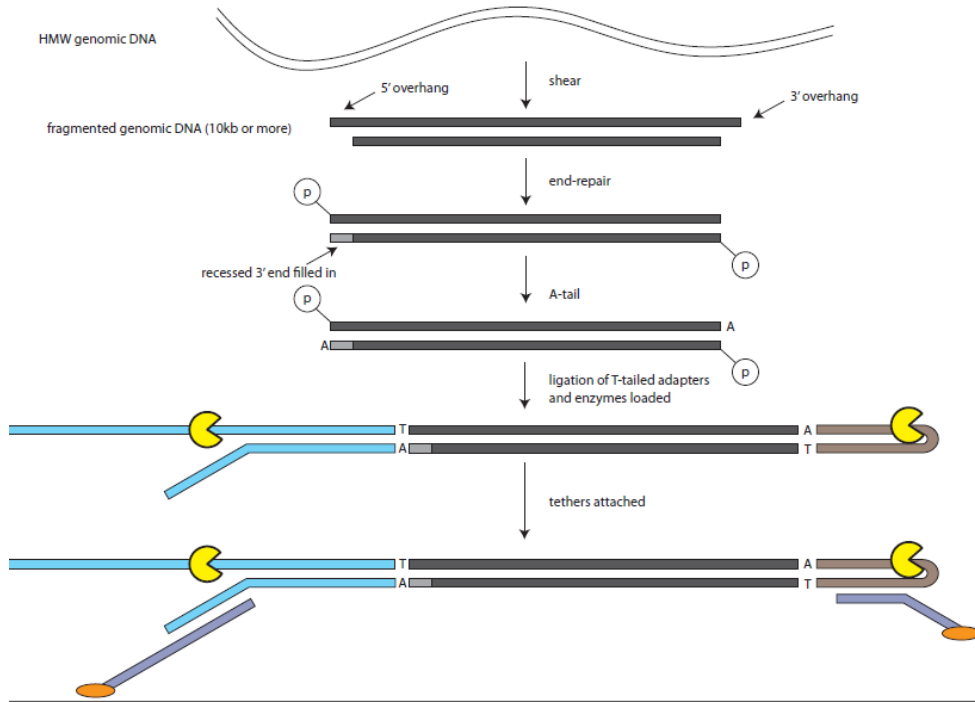
## 5mer current signals



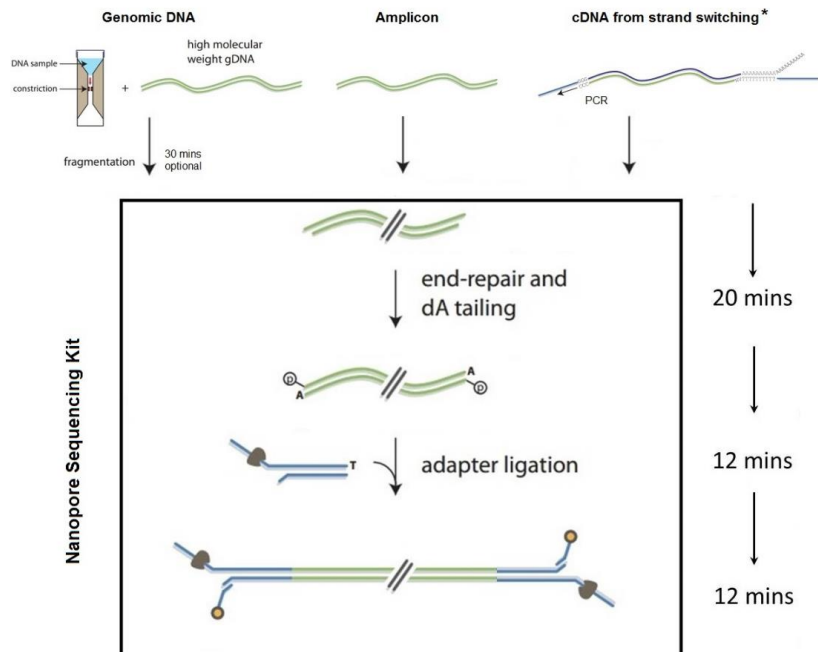
From Szalay & Golovchenko, Nat. Biotech., 2015.



# ONT 2D Library prep



# ONT 1D Library prep



\* or 2D cDNA synthesis

# 1D<sup>2</sup>

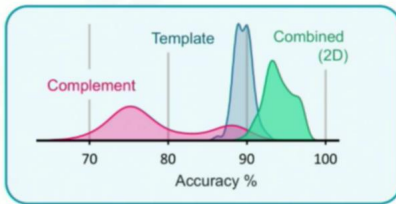


## 1D<sup>2</sup>

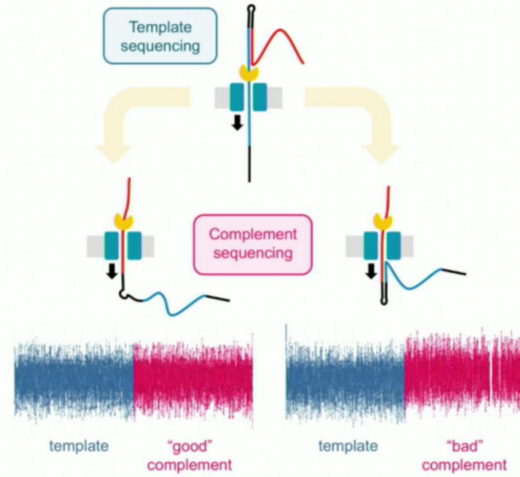
Problems with 2D

### 2D HAS BEEN CHALLENGING FOR MANY YEARS

- 50% of sample is lost in library preparation
- Signal problems arise from 2<sup>o</sup> structure under the pore
  - Hairpins formed change the DNA signal ("uplift")
  - Secondary structure is variable, broad distribution of accuracies
  - "Uplifted" signal results in lower accuracy
  - Missing sections also present from enzyme slips
- Problems much worse with 450 bps chemistry



© Copyright 2017 Oxford Nanopore Technologies | 45



# 1D<sup>2</sup>



Nanopore Live

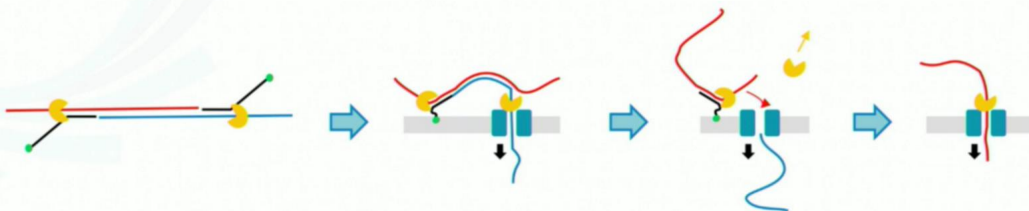
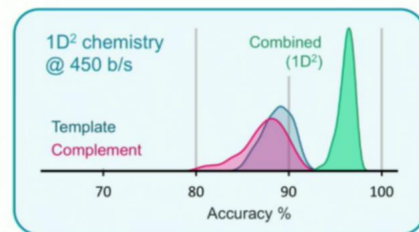
LIVE

## 1D<sup>2</sup>

Improved template – complement data

### SEQUENCING SCHEME WHERE STRANDS ARE NOT JOINED

- Complement follows template as separate independent strand
  - Each molecule has its own motor-adaptor
  - Each individual strand has high 1D accuracy
  - No secondary structure problems
- Simple library preparation, compatible addition to 1D methods
  - Compatible with E8 and 450 bps



Oxford



# 1D<sup>2</sup>



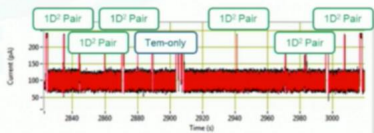
## 1D<sup>2</sup> Efficiency

### BUILDING ON 1D CHEMISTRY

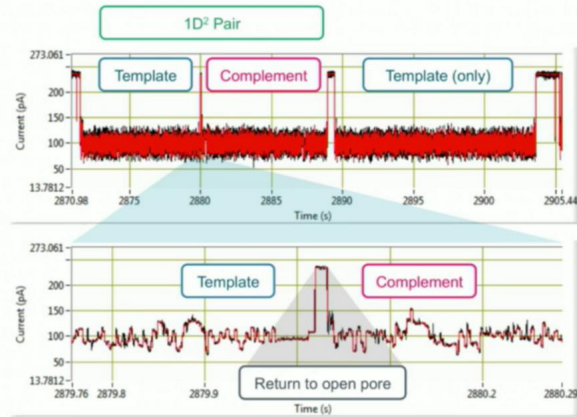
- Targeting same throughput as 1D @ 450 bps
  - 1D<sup>2</sup> is free information
- Typically the second strand immediately follows the template, with a short return to open-pore

### EFFICIENCY

- 1D<sup>2</sup> occurs naturally at < 1% efficiency
- Changes to the R9.4 pore have improved efficiency
- For R9.5 > 60% of strands are 1D<sup>2</sup> pairs



© Copyright 2017 Oxford Nanopore Technologies | 47

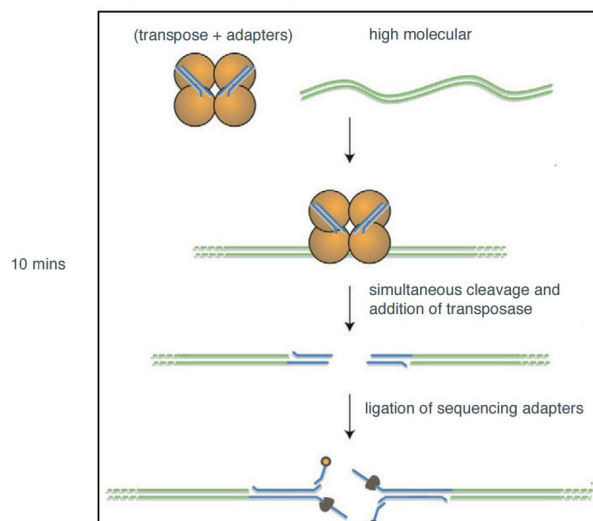


OXFORD  
NANOPORE  
TECHNOLOGIES



## RAPID SEQUENCING KIT

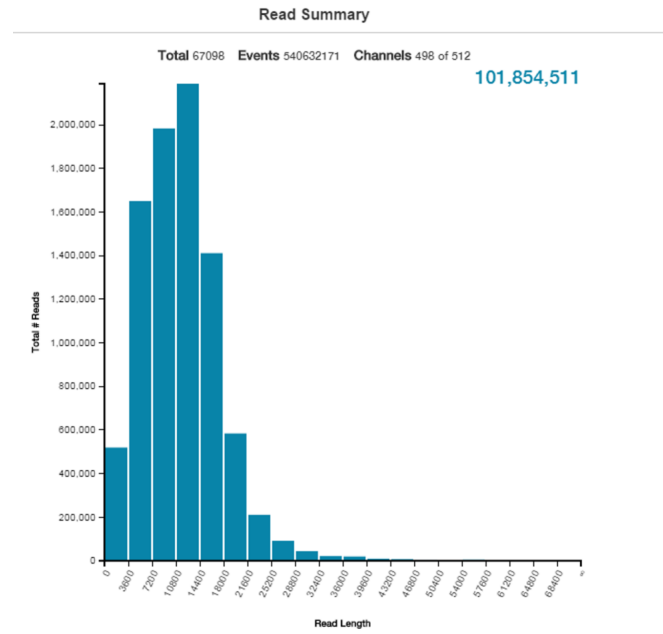
A two-step, 10 minute protocol



Starting material will be fragmented; recommended starting size >30 kb for genomic DNA



# MinKNOW Interface read distribution graph



## Data Quality

2D error rate 2-3%, 1D error rate 8-9%

Read lengths as long as template. Record >1Mb but shorter fragments give higher yield

Some AT bias? Low coverage at high AT and GC. Systematic errors in and around homopolymers

Yield per minION flowcell dependent on DNA 1-17Gb.



## MinION: PORTABLE DNA/RNA SEQUENCING

Sample to scientific insight easily, quickly

Channels\*  
512



An explanation from **NANOPORE** Technologies



# GridION. For service sector



Nanopore Live LIVE

## GridION X5

Bench top sequencing device


**SEQUENCING**

- 5 individually addressable flow cells
- Based on current MinION flow cell design
- Road map to on board Run Until... and Read Until...

**COMPUTE**

- Embedded high performance compute
- Full Real time basecalling and data analysis in the box
- Simple user interface and single ethernet for data transfer

Component	Specification
Size and weight	H200 x W 360x D 360 mm , 10 kg
Power	600 W
Compute spec:	8 TB SSD Storage, 64 GB RAM Latest Gen CPU for OS and orchestration FPGA processor
Pre-loaded software	Linux OS, MinKNOW
Connections	5x USB 3. 1x USB-C 1 x HDMI. 1 x Ethernet



© Oxford





## PromethION: ON-DEMAND, HIGH-THROUGHPUT

Channels\*  
48 x  
3,000  
=144,000



Sample added to flow cell here. Currently this is prepared DNA/RNA, from any original sample



Each flow cell comprises up to 3,000 active channels and has four sample inputs



### Instrument and flow cells

- Modular: 48 flow cells can be used individually/together for on-demand sequencing

### Integrated compute module

- Real time compute provided
- Web based/remote, real time administration & monitoring

An explanation from **NANOPORE** Technologies



Nanopore Live

## NANOPORE SYSTEMS Summary



	MinION	GridION X5	PromethION
Sequencer type	Mobile	Benchtop	Benchtop
System Price	Starter pack of \$1000	\$0 when ordering 300 flow cells	Starter pack of \$135,000
Data produced by starter pack <i>(based on internal best Mar 17)</i>	Up to 40GB	Up to 6TB	Coming soon
Fee For Service available	No	Yes	Yes
<b>Specifications based on internal best Mar 2017</b>			
Run Time	1 min – 48 hours	1 min – 48 hours	1 min – 48 hours
Yield per flow cell	20GB	20GB	50GB*
Yield per Instrument run	20GB	100GB	2.4TB*

\* PromethION yield still in development through the PromethION Early Access Programme

Oxford





# ONT applications

- » Super-long reads (>1Mb)
- » “Run until” done. W.I.M.P
- » Selective reads
- » Cas9 mediated enrichment
- » Mobile sequencing
- » Direct blood analysis?
- » Direct methylation detection?
- » Direct RNA sequencing



## ONT Predicted Performance

- » MinION: 100 Mb/hr. 1-20Gb
- » PromethION: >1Tb/day
- » Read length potentially as long as molecule
- » Run until done
- » \$60-600/Gb minION, no capital cost
- » \$15-25/Gb GridION







## Current Sanger Platforms



- » ABI 3730 Capillary (outsource)
- » Illumina Xten + 5 HS2500 + 2 HS4000
- » 5 Novaseq
- » MiSeq (6)
- » PacBio RSII and Sequel(2)
- » Oxford Nanopore minION (4)
- » Oxford Nanopore gridION (1)
- » Oxford Nanopore promethION (1)





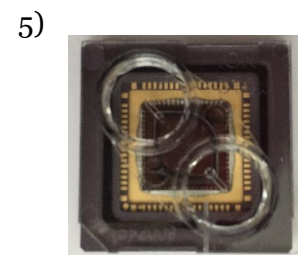
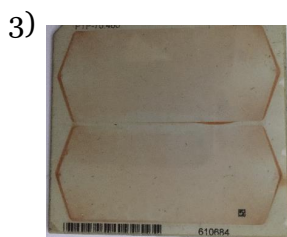
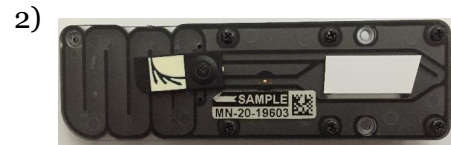
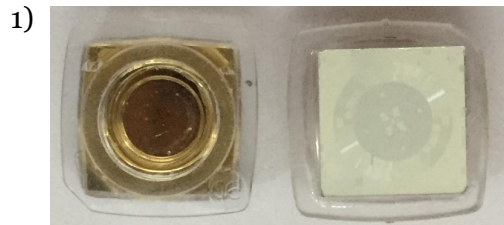
**KEEP  
CALM  
ITS  
COMPETITION  
TIME!**



who wants  
to win...



# Identify the Sequencer

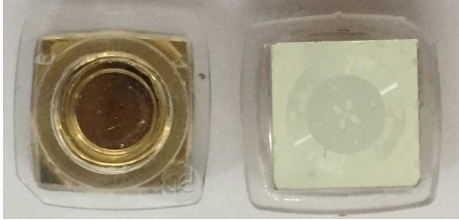


6. How accurate is Q20?
7. What technology features nanoballs
8. What does SBS stand for
9. What does SMRT stand for
10. Name 2 technologies that use sequencing by ligation



# Identify the Sequencer

Pacific Biosciences RSII



minION

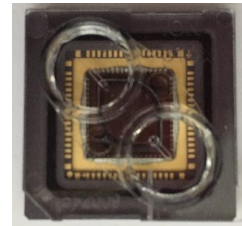


Illumina NovaSeq

454



Ion Torrent PGM



6. How accurate is Q20?

1 error in 100; 99%

7. What technology features nanoballs

Complete Genomics (BGI)

8. What does SBS stand for

Sequencing by Synthesis

9. What does SMRT stand for

Single Molecule Real Time

10. Name 2 technologies that use sequencing by ligation

Solid and Complete Genomics

# The future



STRATOS  
genomics inc.



## Nanostring Hyb n Seq targeted sequencer

Details on James Hadfield's blog:

<http://enseqlopedia.com/2017/02/nanostring-hybseq/>



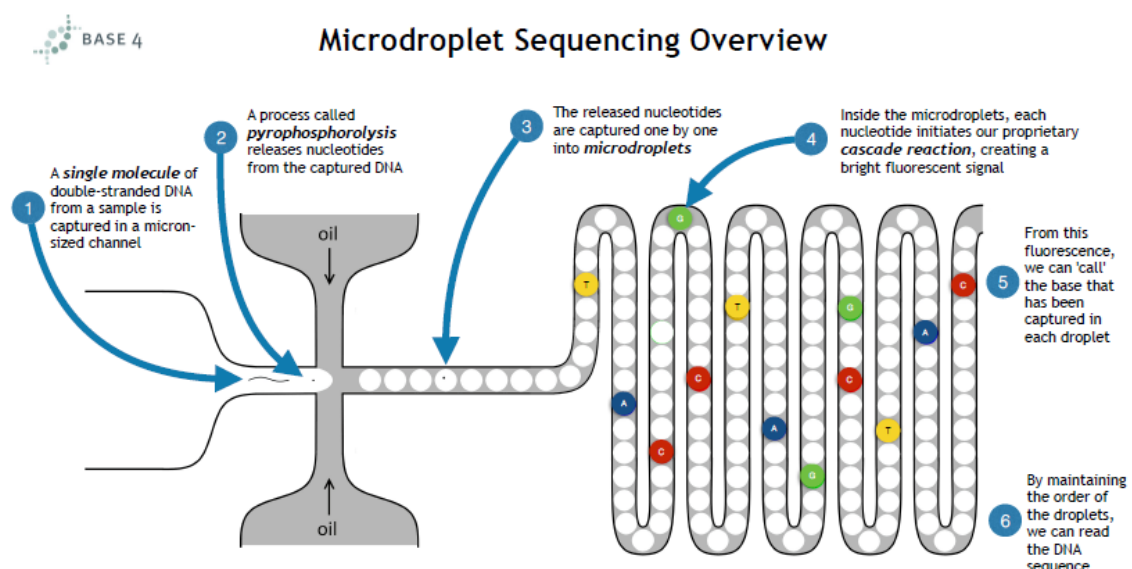


# Base4

- » Based in Cambridge, UK.
- » Developing microfluidics based sequencer
- » Individual bases are cleaved from DNA one at a time, captured into droplets in oil
- » Presence of particular base in droplet causes characteristic fluorescence. Order of droplets = order of bases



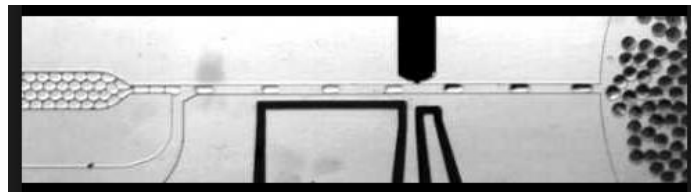
# Base4



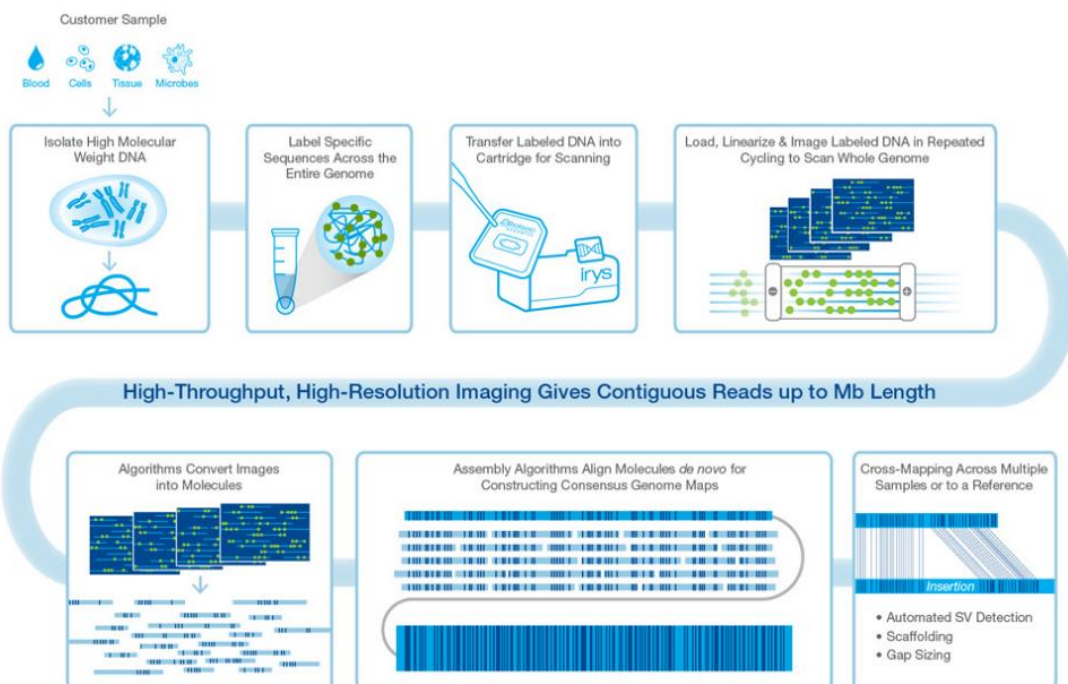


# GnuBio

- » Acquired by BioRad
- » Sequencing by hybridisation in microfluidic oil droplets
- » Extremely accurate
- » \$200 for ~100 amplicons of 1kb

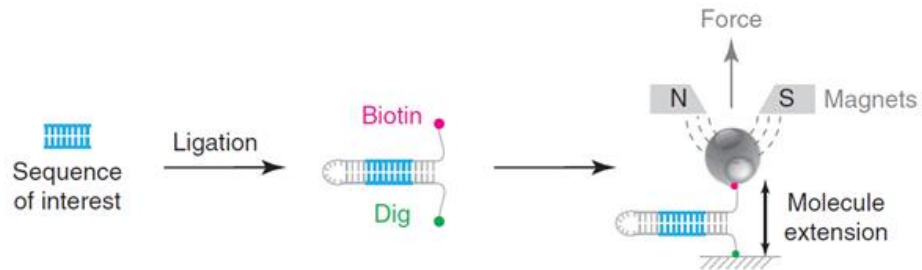


# BioNanogenomics



# PicoSeq

# Depixus

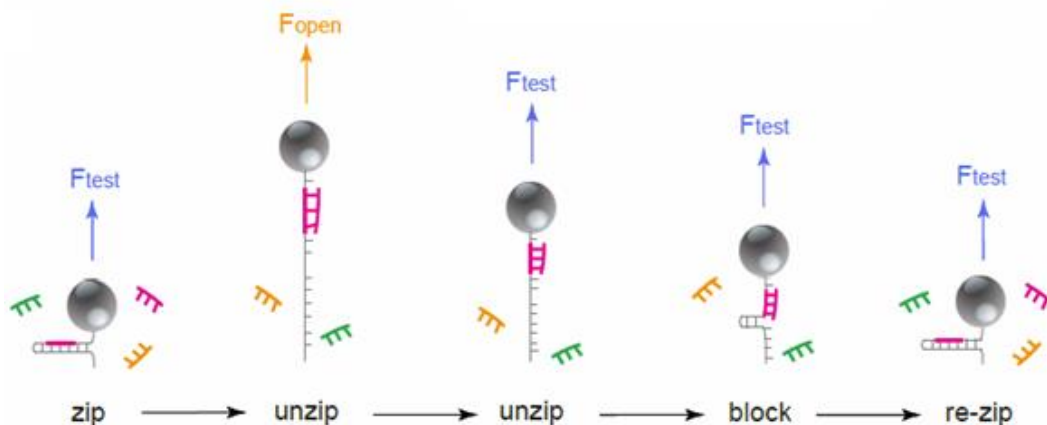


“Single-molecule mechanical identification and sequencing” Ding F., Manosas M., Spiering M.M., Benkovic S.J., Bensimon D., Allemand J-F., & Croquette V. Nature Methods, published online: 11 March 2012 | doi:10.1038/nmeth.1925



# Depixus

Parisian company using magnetic tweezers and SPR to detect binding to surface bound DNA





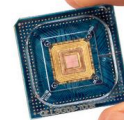
# QuantMDx



- » UK company based in Newcastle. Currently sell microfluidics genotyping device (mPOC). Disease specific cartridges. Uses probes on nanowires. Hybridisation to target gives signal in < 5mins.
- » SBS. Base incorporation changes charge of DNA fragments. This can be detected with nanowires



# Genapsys



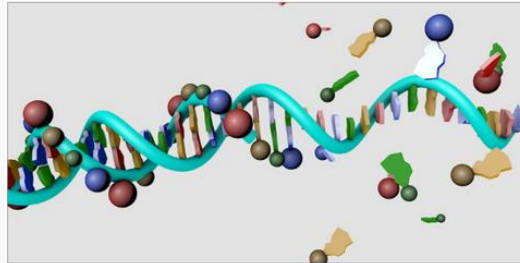
- » Small benchtop sequencer. Works in similar way to Ion Torrent
- » Eventually may be 10x cheaper than Illumina
- » 1kb reads



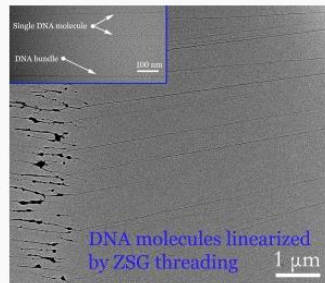


# ZS Genetics

The single-stranded DNA is made into double-stranded DNA using heavy-atoms/modified dNTPs.



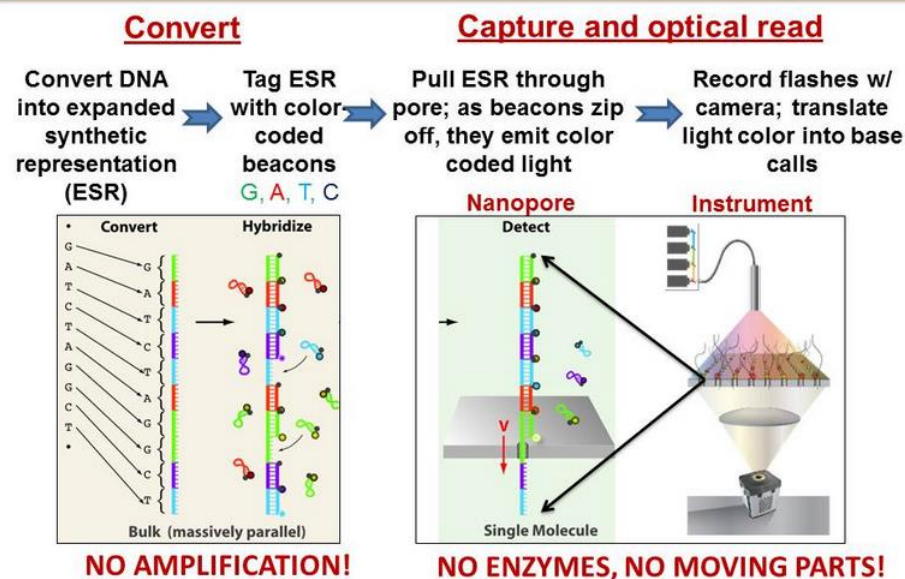
Each modified dNTP-type contains a unique label that is separately identifiable in the electron microscope-generated image, for example by "large" black dots, smaller black dots and large grey dots.



er



# Noblegen



Automated, scalable workflow w/ small instrument

**Noblegen**  
BIOSCIENCES

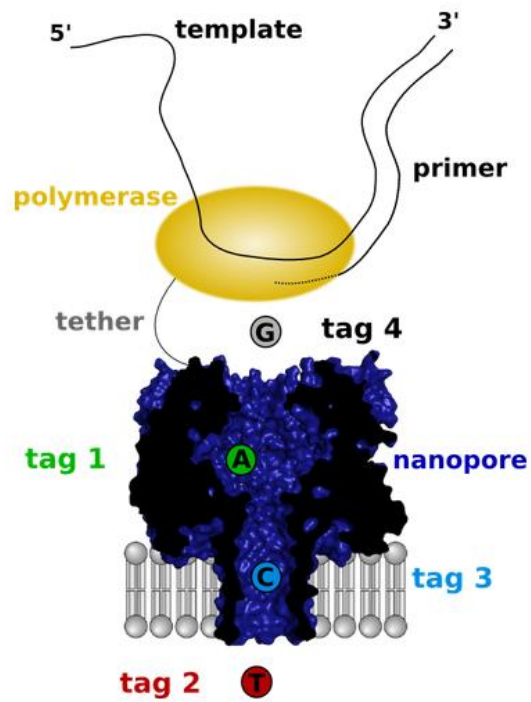
Conversion: 5h

Read 96 genomes: 17h  
1 genome < 30min

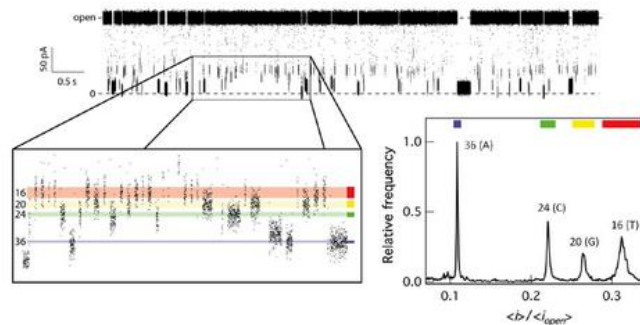




# Genia



# Genia

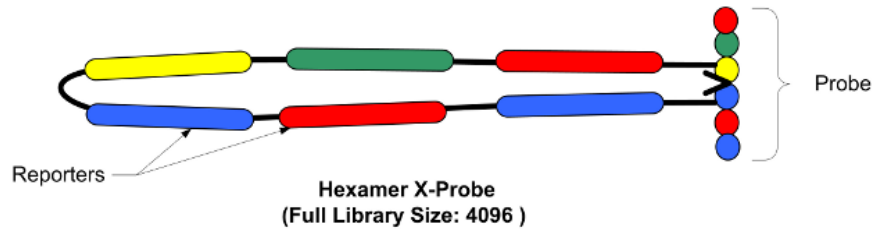


Potential for \$100 genome

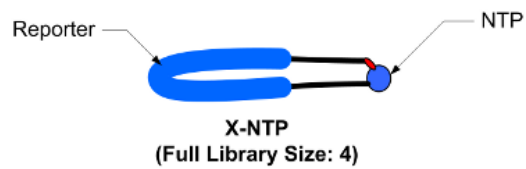




# Stratos



# Stratos







## Joakim Lundeberg -Spatial Transcriptomics (ST) At SciLifeLab

- » Histochemistry/RNA seq on single cell layer
- » Have spatially segregated barcoded oligo dT primers on glass slides



## Joakim Lundeberg -Spatial Transcriptomics (ST) At SciLifeLab

- » Lyse, capture and do cDNA synthesis on surface. Cut off cDNA and make library. Sequence and sort according to barcode.
- » High density array. 135k unique Barcodes .These are 3' tag libraries. Sequence 50 K reads per feature
- » Each 13um square approximates a single cell. All cells on array treated equally





## Joakim Lundeberg - Spatial Transcriptomics (ST) At SciLifeLab

- » Have software to display gene list for each cell
- » Costly and challenging to do deep Sequencing. 40k cells at 50k per run
- » Can couple barcode to gene specific probe to target capture. Also thinking about how would sequence on slide.



Thanks!



**Any Questions ?**

**[mq1@sanger.ac.uk](mailto:mq1@sanger.ac.uk)**





## NGS Bioinformatics: Virtual Machine

Jacqui Keane

jm15@sanger.ac.uk



---

---

---

---

---

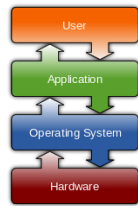
---

---

---

## Operating System (OS)

- ▶ Software that supports the computer's basic functions
  - ▶ Manages computer hardware (screen, mouse, keyboard)
  - ▶ Provides tools for managing files, running software
  - ▶ Provides a way via software applications to interact with the computer



WT NGS Bioinformatics



---

---

---

---

---

---

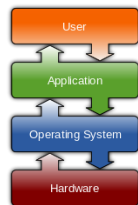
---

---

## Operating System (OS)

- ▶ Software that supports the computer's basic functions
  - ▶ Manages computer hardware (screen, mouse, keyboard)
  - ▶ Provides tools for managing files, running software
  - ▶ Provides a way via software applications to interact with the computer

- ▶ Examples:
  - ▶ Windows
  - ▶ OS X
  - ▶ Unix
  - ▶ Linux



WT NGS Bioinformatics



---

---

---

---

---

---

---

---

## Operating System (OS)

- ▶ Software that supports the computer's basic functions
  - ▶ Manages computer hardware (screen, mouse, keyboard)
  - ▶ Provides tools for managing files, running software
  - ▶ Provides a way via software applications to interact with the computer

- ▶ Examples:
  - ▶ **Windows**
  - ▶ OS X
  - ▶ Unix
  - ▶ **Linux**



WT NGS Bioinformatics



---

---

---

---

---

---

---

---

## Virtual Machine (VM)

- ▶ VM is a computer environment that can be run on any computer
  - ▶ OS, data, software applications
- ▶ Allows you to run one OS (Linux) on another OS (Windows)
- ▶ Created a VM for this course
  - ▶ Linux OS
  - ▶ Data for practicals
  - ▶ Bioinformatics software (bwa, samtools, vcftools, velvet etc.)
- ▶ Take it home and use on your own machine
  - ▶ Take the course again!
  - ▶ Run bioinformatics analysis

WT NGS Bioinformatics



---

---

---

---

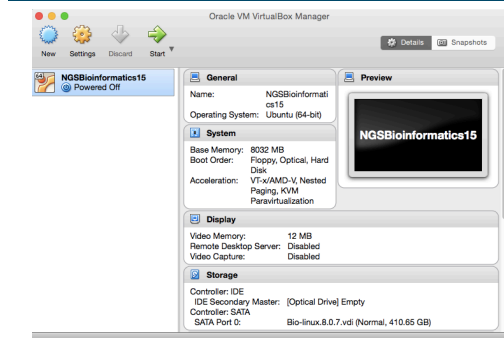
---

---

---

---

## Getting Started



WT NGS Bioinformatics



---

---

---

---

---

---

---

---

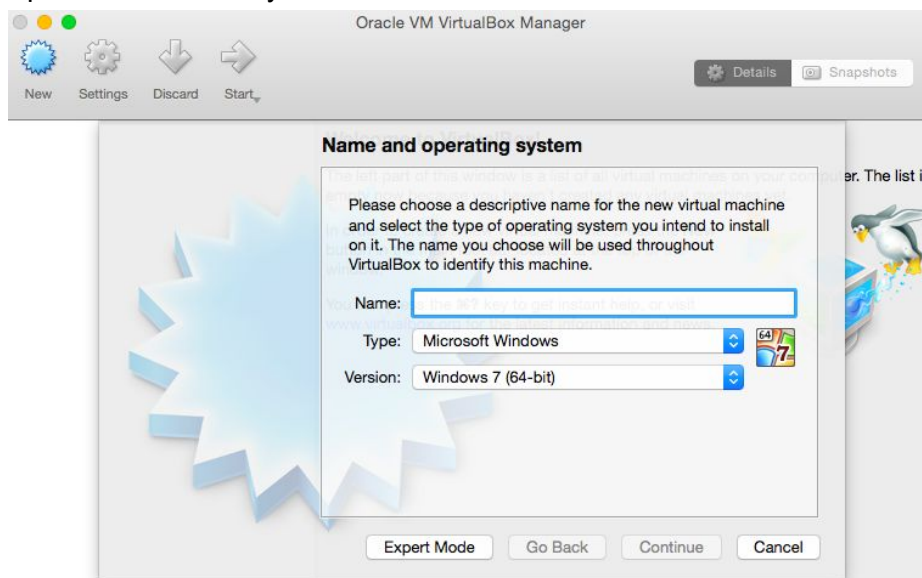
# Virtual machine at home

## What is a virtual machine?

A virtual machine (VM) is a software computer that, like a physical computer, runs an operating system and a set of software applications. It allows you to run one operating system (e.g. Linux) within another operating system (e.g. Windows).

## Adding the virtual machine to VirtualBox

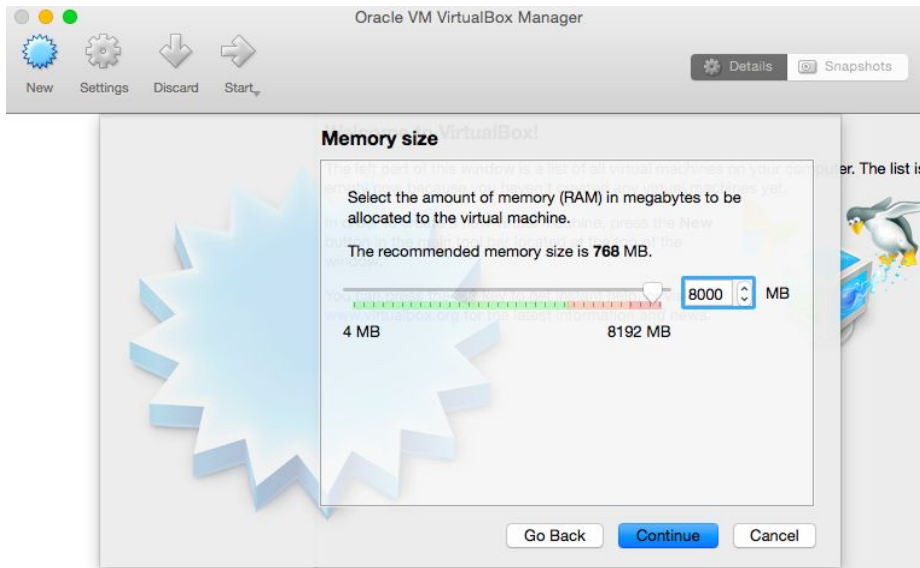
Open VirtualBox on your machine and select 'New'.



Enter a name for your VM, 'NGSBioinformatics16'. Select 'Linux as the' type of operating system that will run on your VM. Select the version as 'Ubuntu (64-bit)' and click Continue.



Select the amount of memory (RAM) to allocate to this VM, the values available to you will depend on the amount of memory (RAM) available on the host machine. For this course, please select 12GB and click 'Continue'.



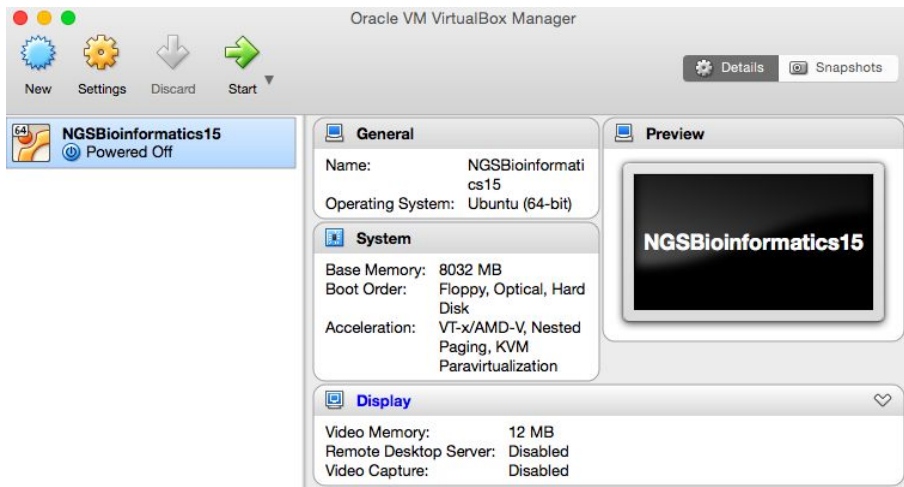
As we have already created the virtual hard disk for you, select 'Use an existing virtual hard disk file' and click on the folder with a green plus on the right hand side.



Navigate to the USB drive and select the file 'Bio-linux.8.0.7.vdi' and click 'Create'.

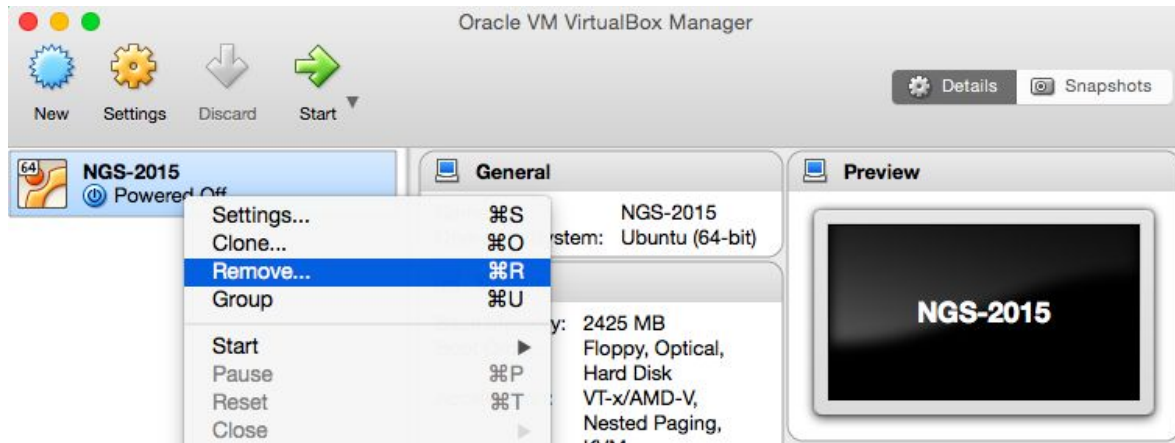


You should then see the screen below containing your VM 'NGSBioinformatics16' switched off.

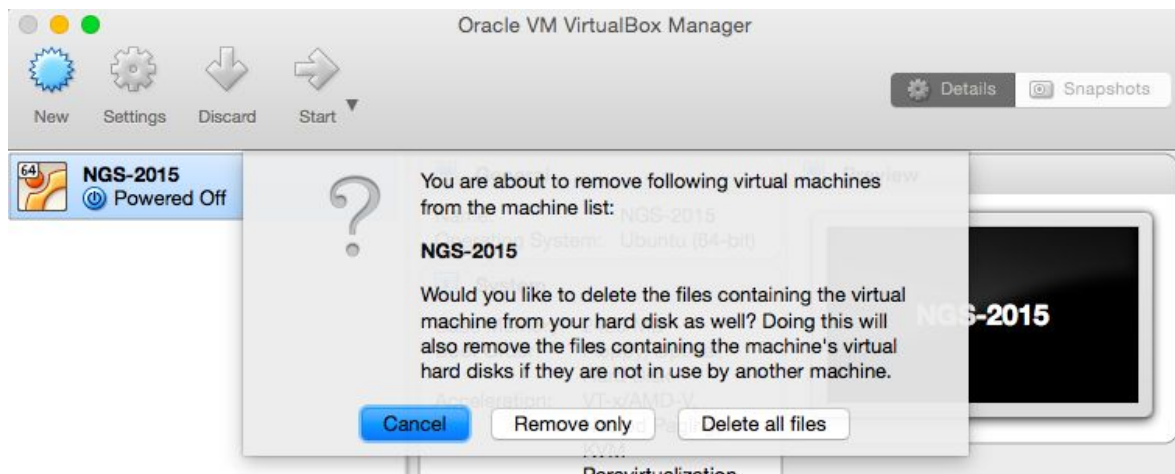


# Removing the virtual machine from VirtualBox

In the main VirtualBox window, select your virtual machine (on the left). Then right click on the name of the machine and select 'Remove'.



Then select 'Remove only', **do not** select the 'Delete all files' option as this will delete the the vdi file that contains the virtual machine.



## Username and password

The username is "manager" and the password is also "manager".

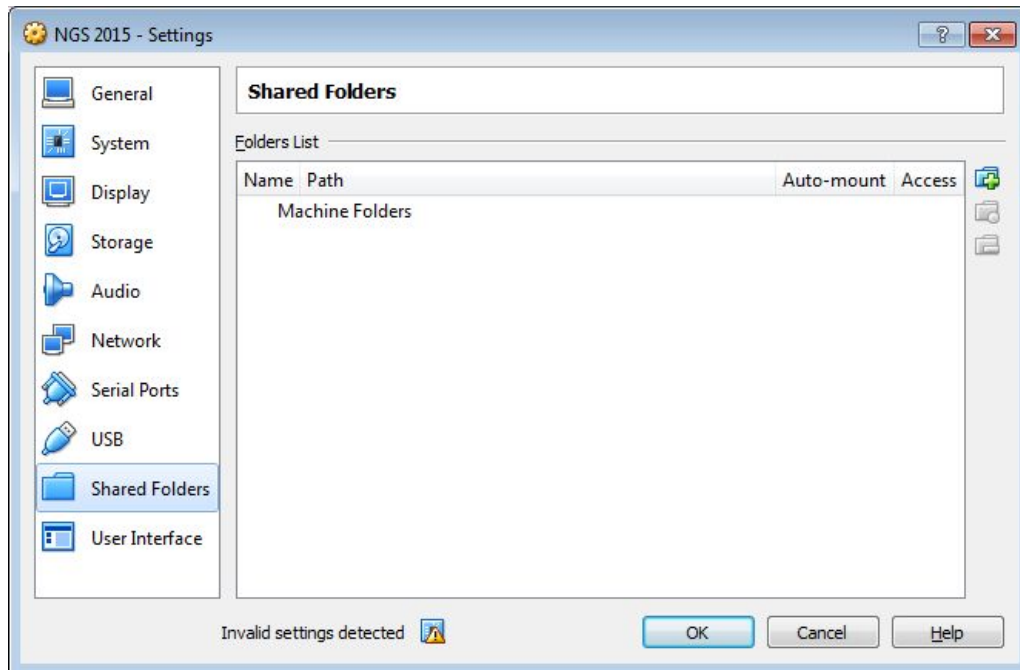
## Shared folders

By default, none of the files on the host machine are visible from inside the virtual machine. The instructions below show how to share data between the host and the virtual machine. This is done by sharing a folder on the host, so that the folder can be used inside the virtual machine.

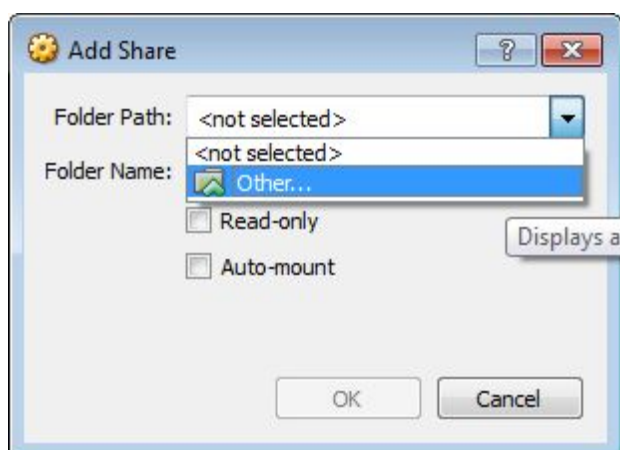
## Settings in VirtualBox

First, make sure that the virtual machine is not running.

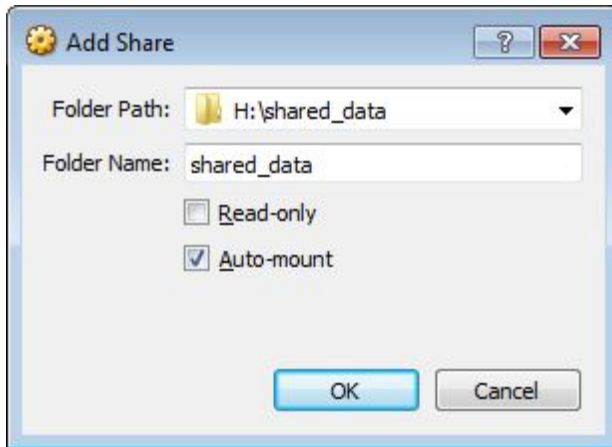
In the main VirtualBox window, select your virtual machine (on the left). Then go to Settings -> Shared Folders.



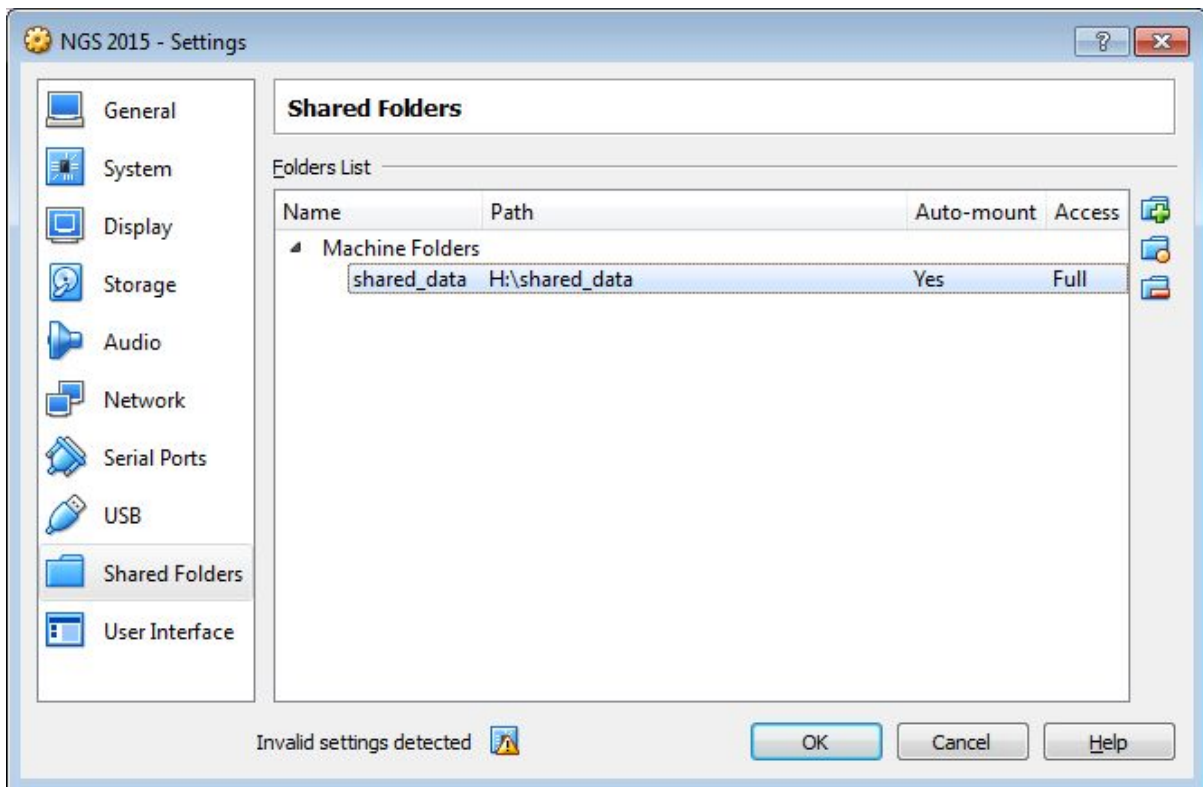
Click on the blue folder with a green plus on the right hand side. In the pop up box, select "Other" from the Folder Path option.



Find the folder that you would like to share with the Virtual Machine. It should now be visible in the Folder Path box. Tick the Auto-mount box.



Now click on OK. The new shared folder should be in the Folders List:



Select OK and you will return to the main VirtualBox window. Now start the virtual machine.

## Finding the shared folder in the virtual machine

Inside the virtual machine, the shared folder will be inside the directory `/media`. It will have the same name as on the folder on the host machine, but with `"sf_"` added. For example, if the folder is called `"shared_data"` on the host, then it will be called `"/media/sf_shared_data/"` inside the virtual machine.



# Introduction to Unix

Jacqui Keane

Pathogen Informatics  
Wellcome Sanger Institute  
jm15@sanger.ac.uk



---

---

---

---

---

---

---

---

## Unix

- ▶ What is Unix?
  - ▶ Standard operating system (alternative to MS Windows, Mac OS)
  - ▶ Provides a way for you to interact with the computer
  - ▶ Many 'flavours' of Unix, using Linux
  - ▶ Originally created to provide a free UNIX-like OS for PCs
- ▶ Why use Unix?
  - ▶ Output of lots of biological research exists in large text files
  - ▶ Very suitable for working with such files
  - ▶ Powerful and flexible commands for processing large text files
  - ▶ Save you time
  - ▶ Widely used in scientific community
  - ▶ Powerful, robust and stable operating system



---

---

---

---

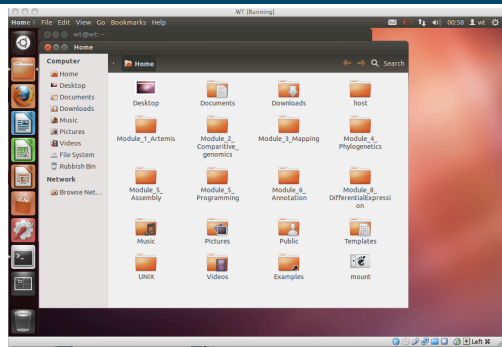
---

---

---

---

## Using Unix



---

---

---

---

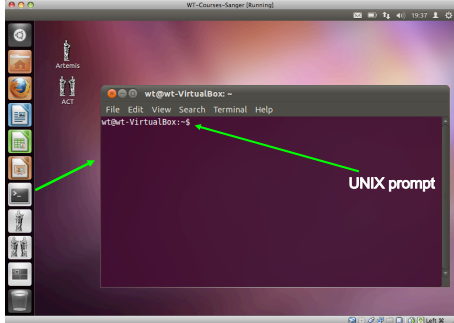
---

---

---

---

## Terminals and Commandline



WT NGS Bioinformatics



---

---

---

---

---

---

---

---

## Unix Commands

Command	What it does
<b>ls</b>	List the contents of the current directory
<b>cd</b>	Changes a directory
<b>mv</b>	Moves a file
<b>cp</b>	Copies a file
<b>rm</b>	Remove a file
<b>less</b>	Displays the contents of a file
<b>head</b>	Displays the first ten lines of a file
<b>tail</b>	Displays the last ten lines of a file
<b>cat</b>	Concatenate files together
<b>pwd</b>	Print working directory
<b>mkdir</b>	Make a new directory

WT NGS Bioinformatics



---

---

---

---

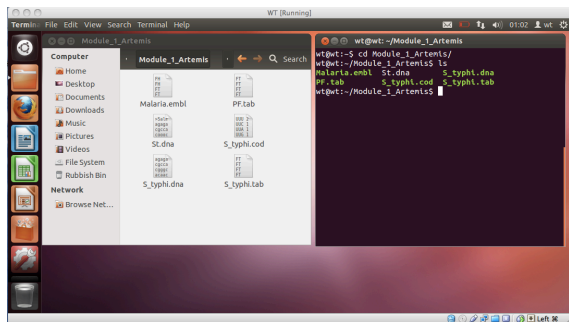
---

---

---

---

## ls command



WT NGS Bioinformatics



---

---

---

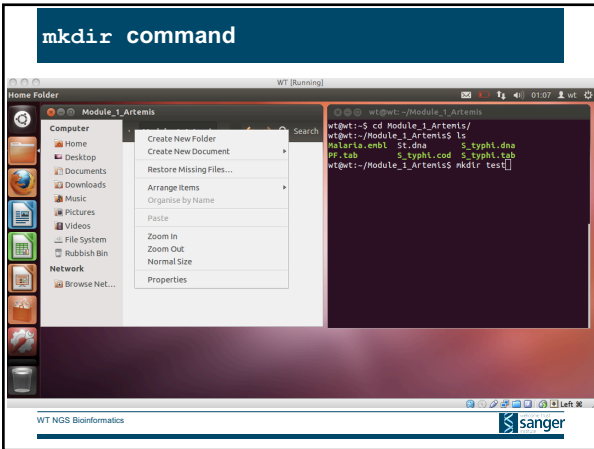
---

---

---

---

---




---

---

---

---

---

---

---

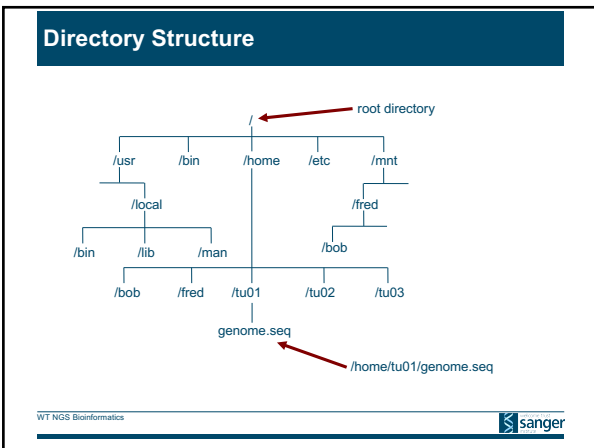
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

---

---

### Unix Tips & Tricks

- ▶ Unix is case sensitive
  - ▶ Typing LS is NOT the same as typing ls
- ▶ You need to put spaces between
  - ▶ a command
  - ▶ the values passed to the command
    - ▶ **mkdir new\_dir will create a new directory**
    - ▶ **mkdirmew\_dir will just give an error!**
- ▶ Unix is not psychic! If you misspell the name of command or a file it will not understand you

---

---

---

---

---

---

---

---

---

---

---

---

## 1 Unix module

### 1.1 Introducing Unix

Unix is the standard operating system on most large computer systems in scientific research, in the same way that Microsoft Windows is the dominant operating system on desktop PCs.

Unix and MS Windows both perform the important job of managing the computer's hardware (screen, keyboard, mouse, hard disks, network connections, etc...) on your behalf. They also provide you with tools to manage your files and to run application software. They both offer a graphical user interface (desktop). These desktop interfaces look different between the operating systems, use different names for things (e.g. directory versus folder) and have different images but they mostly offer the same functionality.

Unix is a powerful, secure, robust and stable operating system which allows dozens of people to run programs on the same computer at the same time. This is why it is the preferred operating system for large-scale scientific computing. It runs on all kinds of machines, from mobile phones (Android), desktop PCs... to supercomputers.

### 1.2 Why Unix?

Increasingly, the output of biological research exists as *in silico* data, usually in the form of large text files. Unix is particularly suitable for working with such files and has several powerful and flexible commands that can be used to process and analyse this data. One advantage of learning Unix is that many of the commands can be combined in an almost unlimited fashion. So if you can learn just six Unix commands, you will be able to do a lot more than just six things.

Unix contains hundreds of commands, but to conduct your analysis you will probably only need 10 or so to achieve most of what you want to do. In this course we will introduce you to some basic Unix commands followed by some more advanced commands and provide examples of how they can be used in bioinformatics analyses.

### 1.3 Sections of the Unix course

- Basic unix
- Files
- grep
- awk
- Bash scripts


We've also included a cheat sheet. It probably won't make a lot of sense now, but it might be a useful reminder of this module later in the course.

### 1.4 Following the course in a terminal

To follow this course, please type all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS

commands to manage files.

To get started, double-click the icon on the desktop called “Module 1 Unix”. This will open a terminal, ready for you to follow the module.

Throughout this manual, each command to be typed in the terminal is shown in white text on a black background, next to a picture of a keyboard: . The expected output is shown underneath with a yellow background.

For example, try typing this into your terminal, followed by the Return key.



```
echo hello world
```

```
hello world
```

Now follow the instructions in the Basic Unix section.

## 2 Basic Unix

### 2.1 The Commandline

The commandline or ‘terminal’ is an interface you can use to run programs and analyse your data. If this is your first time using one it will seem pretty daunting at first but, with just a few commands, you’ll start to see how it helps you to get things done much quicker. You’re probably more familiar with software which uses a graphical user interface, also known as a GUI; unfortunately most of the best bioinformatics software has not been programmed with this capability.

### 2.2 Getting started

Before we get started, let’s check that you’re in the right place. If you haven’t done so already, double-click the desktop icon called “Module 1 Unix”, which will open a new terminal window. Then type the following, and press Return. Do not worry about the meaning of this for now – it is explained later in the module.



```
cd basic
```

```
(no output)
```

There should be no output to your terminal. Now continue through the course, entering any commands that you encounter into your terminal window.

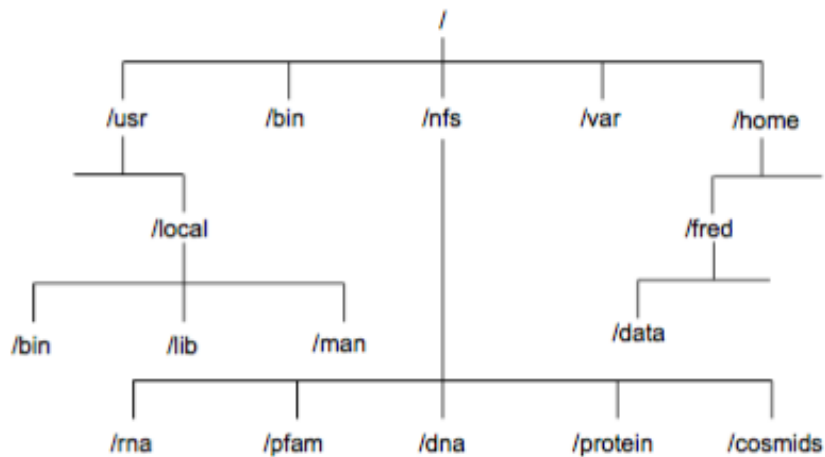
However, before getting started there are some general points to remember that will make your life easier:

- Unix is case sensitive - typing `ls` is not the same as typing `LS`.
- Often when you have problems with Unix, it is due to a spelling mistake. Check that you have not missed or added a space. Pay careful attention when typing commands across a couple of lines.

### 2.3 Files and directories

*Directories* are the Unix equivalent of folders on a PC or Mac. They are organised in a hierarchy, so directories can have sub-directories and so on. Directories are very useful for organising your work and keeping your account tidy - for example, if you have more than one project, you can organise the files for each project into different directories to keep them separate. You can think of directories as rooms in a house. You can only be in one room (directory) at a time. When you are in a room you can see everything in that room easily. To see things in other rooms, you have to go to the appropriate door and crane your head around. Unix works in a similar manner, moving from directory to directory to access files. The location or directory that you are in is referred to as the current working directory.

If there is a file called `genome.seq` in the `dna` directory its location or full pathname can be expressed as `/nfs/dna/genome.seq`.

**Directory structure example**

Hierarchy

**2.4 pwd - find where you are**

The command `pwd` stands for print working directory. A *command* (also known as a *program*) is something which tells the computer to do something. Commands are therefore often the first thing that you type into the terminal (although we'll show you some advanced exceptions to this rule later).

As described above, directories are arranged in a hierarchical structure. To determine where you are in the hierarchy you can use the `pwd` command to display the name of the current working directory. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.

To find out where you are, type this into your terminal.



```
pwd
```

```
/home/manager/course_data/Module1_Unix/Notebooks/Unix/basic
```

Remember that Unix is case sensitive, `PWD` is not the same as `pwd`.

`pwd` will list each of the folders you would need to navigate through to get from the `root` of the file system to your current directory. This is sometimes referred to as your 'absolute path' to distinguish that it gives a complete route rather than a 'relative path' which tells you how to get from one folder to another. More on that shortly.

**2.5 ls - list the contents of a directory**

The command `ls` stands for list. The `ls` command can be used to list the contents of a directory.

To list the contents of your current working directory type:



```
ls
```

```
basic.ipynb directory_structure.png Pfalciparum Styphi
```

You should see that there are 4 items in this directory.

To list the contents of a directory with extra information about the items type:



```
ls -l
```

```
total 56
-rw-rw-r-- 1 manager manager 19512 Sep 27 10:16 basic.ipynb
-rw-rw-r-- 1 manager manager 28513 Sep 27 10:16 directory_structure.png
drwxrwxr-x 4 manager manager 4096 Sep 27 10:16 Pfalciparum
drwxrwxr-x 2 manager manager 4096 Sep 27 11:42 Styphi
```

Instead of printing out a simple list, this should have printed out additional information about each file. Note that there is a space between the command `ls` and the `-l`. There is no space between the dash and the letter `l`.

`-l` is our first example of an *option*. Many commands have options which change their behaviour but are not always required.

What do each of the columns represent?

To list all contents of a directory including hidden files and directories type:



```
ls -a -l
```

```
total 64
drwxrwxr-x 4 manager manager 4096 Sep 27 11:42 .
drwxrwxr-x 8 manager manager 4096 Sep 27 10:16 ..
-rw-rw-r-- 1 manager manager 19512 Sep 27 10:16 basic.ipynb
-rw-rw-r-- 1 manager manager 28513 Sep 27 10:16 directory_structure.png
-rw-rw-r-- 1 manager manager 0 Sep 27 10:16 .hidden1
-rw-rw-r-- 1 manager manager 0 Sep 27 10:16 .hidden2
drwxrwxr-x 4 manager manager 4096 Sep 27 10:16 Pfalciparum
drwxrwxr-x 2 manager manager 4096 Sep 27 11:42 Styphi
```

This is an example of a command which can take multiple options at the same time. Different commands take different options and sometimes (unhelpfully) use the same letter to do different things.

How many hidden files and directories are there?

Try the same command but with the `-h` option:



```
ls -alh
```



```
total 64K
drwxrwxr-x 4 manager manager 4.0K Sep 27 11:42 .
drwxrwxr-x 8 manager manager 4.0K Sep 27 10:16 ..
-rw-rw-r-- 1 manager manager 20K Sep 27 10:16 basic.ipynb
-rw-rw-r-- 1 manager manager 28K Sep 27 10:16 directory_structure.png
-rw-rw-r-- 1 manager manager 0 Sep 27 10:16 .hidden1
-rw-rw-r-- 1 manager manager 0 Sep 27 10:16 .hidden2
drwxrwxr-x 4 manager manager 4.0K Sep 27 10:16 Pfalciparum
drwxrwxr-x 2 manager manager 4.0K Sep 27 11:42 Styphi
```

You'll also notice that we've combined `-a -l -h` into what appears to be a single `-alh` option. It's almost always ok to do this for options which are made up of a single dash followed by a single letter.

What does the `-h` option do?

To list the contents of the directory called Pfalciparum with extra information type:



```
ls -l Pfalciparum/
```

```
total 24472
drwxrwxr-x 2 manager manager 4096 Sep 27 10:16 annotation
drwxrwxr-x 2 manager manager 4096 Sep 27 10:16 fasta
-rw-rw-r-- 1 manager manager 654069 Sep 27 10:16 MAL1.fa
-rw-rw-r-- 1 manager manager 962943 Sep 27 10:16 MAL2.fa
-rwxrwxr-x 1 manager manager 23241585 Sep 27 10:16 Malaria.fa
-rwxrwxr-x 1 manager manager 183279 Sep 27 10:16 Pfalciparum.bed
```

In this case we gave `ls` an *argument* describing the *relative path* to the directory Pfalciparum from our current working directory. Arguments are very similar to options (and I often use the terms interchangeably) but they often refer to things which are not prefixed with dashes.

How many files are there in this directory?

## 2.6 Tab completion

Typing out file names is really boring and you're likely to make typos which will at best make your command fail with a strange error and at worst overwrite some of your carefully crafted analysis. *Tab completion* is a trick which normally reduces this risk significantly.

Instead of typing out `ls Pfalciparum/`, try typing `ls P` and then press the `tab` character (instead of `Enter`). The rest of the folder name should just appear. If you have two folders with similar names (e.g. `my_awesome_scripts/` and `my_awesome_results/`) then you might need to give your terminal a bit of a hand to work out which one you want. In this case you would type `ls -l m`, when you press `tab` the terminal would read `ls -l my_awesome_`, you could then type `s` followed by another `tab` and it would work out that you meant `my_awesome_scripts/`

## 2.7 File permissions

Every file and directory have a set of permissions which restrict what can be done with a file or directory.

- Read (r): permission to read from a file/directory
- Write (w): permission to modify a file/directory
- Execute (x): Tells the operating system that the file contains code for the computer to run, as opposed to a file of text which you open in a text editor.

The first set of permissions (characters 2,3,4) refer to what the owner of the file can do, the second set of permissions (5,6,7) refers to what members of the Unix group can do and the third set of permissions (8,9,10) refers to what everyone else can do.

## 2.8 cd - change current working directory

The command `cd` stands for change directory.

The `cd` command will change the current working directory to another, in other words allow you to move up or down in the directory hierarchy.

To move into the `Styphi` directory type the following. Note, you'll remember this more easily if you type this into the terminal rather than copying and pasting. Also remember that you can use tab completion to save typing all of it.



```
cd Styphi/
```

(no output)

Now use the `pwd` command to check your location in the directory hierarchy and the `ls` command to list the contents of this directory.



```
pwd  
ls
```

```
/home/manager/course_data/Module1_Unix/Notebooks/Unix/basic/Styphi  
Styphi.fa Styphi.gff Styphi.noseq.gff
```

You should see that there are 3 files called: `Styphi.fa`, `Styphi.gff`, `Styphi.noseq.gff`

## 2.9 Tips

There are some short cuts for referring to directories:

- `.` Current directory (one full stop)
- `..` Directory above (two full stops)
- `~` Home directory (tilda)
- `/` Root of the file system (like C: in Windows)

Try the following commands, what do they do?



```
ls .
```

```
Styphi.fa Styphi.gff Styphi.noseq.gff
```



```
ls ..
```

```
basic.ipynb directory_structure.png Pfalciparum Styphi
```



```
ls ~
```

```
bin      data Downloads Pictures Templates  
course_admin Desktop igv Public Videos  
course_data Documents Music sf_pathogens-vm
```

Try moving between directories a few times. Can you get into the `Pfalciparum/` and then back into `Styphi/`?

## 2.10 cp - copy a file

The command `cp` stands for copy.

The `cp` command will copy a file from one location to another and you will end up with two copies of the file.

To copy the file `Styphi.gff` to a new file called `StyphiCT18.gff` type:



```
cp Styphi.gff StyphiCT18.gff
```

```
(no output)
```

Use `ls` to check the contents of the current directory for the copied file:



```
ls
```

```
StyphiCT18.gff Styphi.fa Styphi.gff Styphi.noseq.gff
```

## 2.11 mv - move a file

The `mv` command stand for move.

The `mv` command will move a file from one location to another. This moves the file rather than copies it, therefore you end up with only one file rather than two. When using the command, the path or pathname is used to tell Unix where to find the file. You refer to files in other directories by using the list of hierarchical names separated by slashes. For example, the file called `bases` in the directory `genome` has the path `genome/bases`. If no path is specified, Unix assumes that the file is in the current working directory.

To move the file `StyphiCT18.gff` from the current directory to the directory above type:



```
mv StyphiCT18.gff ..
```

(no output)

Use the `ls` command to check the contents of the current directory and the directory above to see that `StyphiCT18.gff` has been moved.



```
ls
```

```
Styphi.fa Styphi.gff Styphi.noseq.gff
```



```
cd ..  
ls
```

```
basic.ipynb directory_structure.png Pfalciparum Styphi StyphiCT18.gff
```

## 2.12 rm - delete a file

The command `rm` stands for remove.

The `rm` command will delete a file permanently from your computer so take care!

To remove the copy of the *S. typhi* file, called `StyphiCT18.gff` type:



```
rm StyphiCT18.gff
```

(no output)

Use the `ls` command to check the contents of the current directory to see that the file `StyphiCT18.gff` has been removed.



```
ls
```

```
basic.ipynb directory_structure.png Pfalciparum Styphi
```

Unfortunately there is no “recycle bin” on the command line to recover the file from, so you have to be careful.

## 2.13 find - find a file

The `find` command can be used to find files matching a given expression. It can be used to recursively search the directory tree for a specified name, seeking files and directories that match the given name.

To find all files in the current directory and all its subdirectories that end with the suffix `gff`:



```
find . -name "*.gff"
```

```
./Styphi/Styphi.noseq.gff
./Styphi/Styphi.gff
```

How many gff files did you find?

To find all the subdirectories contained in the current directory type:



```
find . -type d
```

```
.
./Styphi
./Pfalcciparum
./Pfalcciparum/annotation
./Pfalcciparum/fasta
```

How many subdirectories did you find?

These are just two basic examples of the find command but it is possible to use the following find options to search in many other ways:

- `-mtime` : search files by modifying date
- `-atime` : search files by last access date
- `-size` : search files by file size
- `-user` : search files by user they belong to

## 2.14 Exercises

Many people panic when they are confronted with a Unix prompt! Don't! All the commands you need to solve these exercises are provided above and don't be afraid to make a mistake. If you get lost ask a demonstrator. If you are a person skilled at Unix, be patient this is only a short exercise.

To begin, open a terminal window and navigate to the `basic` directory in the `Unix` directory (remember use the Unix command `cd`) and then complete the exercise below.

1. Use the `ls` command to show the contents of the `basic` directory.
2. How many files are there in the `Pfalcciparum` directory?
3. What is the largest file in the `Pfalcciparum` directory?
4. Move into the `Pfalcciparum` directory.
5. How many files are there in the `fasta` directory?
6. Copy the file `Pfalcciparum.bed` in the `Pfalcciparum` directory into the `annotation` directory.
7. Move all the `fasta` files in the directory `Pfalcciparum` to the `fasta` directory.
8. How many files are there in the `fasta` directory?
9. Use the `find` command to find all `gff` files in the `Unix` directory, how many files did you find?
10. Use the `find` command to find all the `fasta` files in the `Unix` directory, how many files did you find?

## 3 Looking inside files

A common task is to look at the contents of a file. This can be achieved using several different Unix commands, `less`, `head` and `tail`. Let us consider some examples.

But first, change directory into the `Unix/files/` directory (hint: you might need to go up a directories first using `cd ../..`). Check that the following commands give you a similar output:



```
pwd
ls
```

```
/home/manager/course_data/Module1_Unix/Notebooks/Unix/files
files.ipynb Styphi.concatenated.gff Styphi.gff
Pfalciparum.bed Styphi.fa Styphi.noseq.gff
```

### 3.1 less

The `less` command displays the contents of a specified file one screen at a time. To test this command type the following command followed by the enter key:

```
less Styphi.gff
```

The contents of the file `Styphi.gff` is displayed one screen at a time, to view the next screen press the space bar. As `Styphi.gff` is a large file this will take a while, therefore you may want to escape or exit from this command. To do this, press the `q` key, this kills the `less` command and returns you to the Unix prompt. `less` can also scroll backwards if you hit the `b` key. Another useful feature is the slash key, `/`, to search for an expression in the file. Try it, search for the gene with locus tag `t0038`. What is the start and end position of this gene?

### 3.2 head and tail

Sometimes you may just want to view the text at the beginning or the end of a file, without having to display all of the file. The `head` and `tail` commands can be used to do this.

The `head` command displays the first ten lines of a file.

To look at the beginning of the file `Styphi.gff` file type:



```
head Styphi.gff
```

```
##gff-version 3
##sequence-region AE014613 1 4791961
#!Date 2011-07-11
#!Type DNA
#!Source-version EMBOSS 6.3.1
AE014613 EMBL databank_entry 1 4791961 0.000 + . ID="AE014613.1";
  ↳ organism="Salmonella enterica subsp. enterica serovar Typhi str. Ty2";
  ↳ sub_species=enterica;strain="Ty2";mol_type="genomic DNA";serovar="Typhi
  ↳ ";db_xref="taxon:209261"
```

```

AE014613     EMBL  gene  190  255  0.000 +      .      ID="AE014613.2";gene="
↳ thrL";locus_tag="t0001"
AE014613     EMBL  CDS   190  255  0.000 +      0      ID="AE014613.3";
↳ codon_start=1;transl_table=11;gene="thrL";locus_tag="t0001";product="thr
↳ operon leader peptide";note="corresponds to STY0001 from Accession
↳ AL513382: Salmonella typhi CT18";db_xref="GOA:Q8XG12";db_xref="InterPro:
↳ IPR011720";db_xref="UniProtKB/Swiss-Prot:Q8XG12";protein_id="AAO67735
↳ .1";translation="MNRISTTTTITTTITTTGNGAG"
AE014613     EMBL  gene  337  2799 0.000 +      .      ID="AE014613.4";gene="
↳ thrA";locus_tag="t0002"
AE014613     EMBL  CDS   337  2799 0.000 +      0      ID="AE014613.5";
↳ codon_start=1;transl_table=11;gene="thrA";locus_tag="t0002";product="
↳ aspartokinase I";note="corresponds to STY0002 from Accession AL513382:
↳ Salmonella typhi CT18; homoserine dehydrogenase I";db_xref="GOA:Q8Z9R7";
↳ db_xref="HSSP:1EBF";db_xref="InterPro:IPR001048";db_xref="InterPro:
↳ IPR001341";db_xref="InterPro:IPR001342";db_xref="InterPro:IPR002912";
↳ db_xref="InterPro:IPR005106";db_xref="InterPro:IPR011147";db_xref="
↳ InterPro:IPR016040";db_xref="InterPro:IPR018042";db_xref="InterPro:
↳ IPR019811";db_xref="UniProtKB/TrEMBL:Q8Z9R7";protein_id="AAO67736.1";
↳ translation="
↳ MRVLKFGGTSVANAERFLRVADILESNSRQGVATVLSAPAKITNHLVAMIEKTIGGQDALPNI SDAERIFSDLLAGLASAQPGE
↳ "

```

The `tail` command displays the last ten lines of a file.

To look at the end of `Styphi.gff` type:



```
tail Styphi.gff
```

```

tattgagggttttccacacccttgccgacgcgctccacgatgtggattttaccgtagcgac
gacagcccgcagccgggcaaaatttcattactacgcttcgcccgctgaactggtccctt
attacaggaaaaatcacgctggatgcgctcatgccgcgctgggttttggccgtgaggattc
cggctcgaccaacgacgagctggcgctggcgatgtattgaccggcgtgccgatggcggc
ggattacccttcgctcaatctgggtcaggcggctcatggtgtattgctatcaattagcagg
tttaatgcaacagacccccggaatccggttgatattgctgatgaatcgagttacaggcggt
acgcgcgcgcttttacgcctgctaaccactctggaggcggccgatgaccacaaattaac
cgactggctacaacagcgaatcggcctgctgggacagcgagatacggcaatggtgcaccg
tttgggtccatgatattgaaaaaaaaactaacaaaataacgtggtgtaatttttaaaataat
a

```

The amount of the file that is displayed can be increased by adding extra arguments. To increase the number of lines viewed from 10 to 25 add `-n 25` to the command:



```
tail -n 25 Styphi.gff
```

```

tcgaatacatcatagccttccgcttcgaaaataacttttcaacgtggtgcggtggtaccaac
tcgcttcaacgataaagaatgtgccccgctgcatggttgctacctaaattgccaaactaa
atcgaaacaggaagtacaaaagtcctgacctgctgatgcatgtcgcaaattaacatga
tcggcgtaacatgactaaagtacgtaattgcggttcttgatgcactttccatcaacgtcaa
caacatcattagcttgggtcgtgggtactttccctctggaccgcagcagtgcaaaaacggc
tgtcatcctaaccattttaacagcaacataaacaggctaacagcgtaccggacacctataa
aactacgcttcggttgacatatatcaagttcaattgtagcacggttaacagtttgatgaaat

```

```

catcgtagctaaatgctagctttcatcacaatttgcaatattccaactagttacgtaag
ccaactaataaatgcgatgaatccaaagaacaggatctattttaattaaattatcctaa
ataaacagcaggataacgatgttctgttaacataaacagcaatagtacagatagcaata
gtgtagcgtcttttacgaaatcaaaaatgctttttcagtgatccggtaaaattttgta
aatttgcaagcgtaatatgcttacaaacgccagctaatttcctgtaaattagtcaaaaa
gagtaatgaaatgcggtgaacaatcgttcttgcgctcccgccagagcggaaaatcgg
cgcagccgcccgggctatgaagaccatgggatttactgacctgctgattgtcgacagcca
ggcgcacctagagcccgcctaccggtgggtcgacacatggatctggagatattattgataa
tattgaggttttccacaccttgccgacgcgctccacgatgtggattttaccgtagcgac
gacagcccgcagccgggcaaaaatttcattactacgcttcgcccgcgtaactggttccctt
attacaggaaaaatcacgctggatgctcatgccgcgctggttttggccgtgaggattc
cggctgaccaacgacgagctggcgctggcgatgtattgaccggcgtgccgatggcggc
ggattacccttgcctcaatctgggtcaggcggatgggtgtattgctatcaattagcagg
tttaatgcaacagaccccggaatccggtgatattgctgatgaatcgagttacaggcgtt
acgcgcgcgcttttacgcctgctaaccactctggaggcggccgatgaccacaaattaac
cgactggctacaacagcgaatcggcctgctgggacagcgagatacggcaatggtgcaccg
tttgggtccatgatattgaaaaaaaaactaacaaaataacgtggtgtaatttttaaaataat
a

```

In this case you've given `tail` an argument in two parts. In this case the `-n` says that you want to specify the number of lines to show and the `25` bit tells it how many. Unlike earlier when we merged arguments like `ls -lha` together, it's not a good idea to merge multiple two part arguments together because otherwise it is ambiguous which value goes with which argument.

`-n` is such a common argument for `tail` and `head` that it even has a shorthand: `-n 25` and `-25` mean the same thing.

### 3.3 Saving time

Saving time while typing may not seem important, but the longer that you spend in front of a computer, the happier you will be if you can reduce the time you spend at the keyboard.

- Pressing the up/down arrows will let you scroll through previous commands entered.
- If you highlight some text, middle clicking on the mouse will paste it on the command line.
- Tab completion doesn't just work on filenames, it also works on commands. Try it by typing `fin` and pressing `tab`...

`fin`

Although tab completion works on commands and filenames, unfortunately it rarely works on options or other arguments.

### 3.4 Getting help `man`

To obtain further information on any of the Unix commands introduced in this course you can use the `man` command. For example, to get a full description and examples of how to use the `tail` command type the following command in a terminal window.

```
man tail
```



There are several other useful commands that can be used to manipulate and summarise information inside files and we will introduce some of these next, `cat`, `sort`, `wc` and `uniq`.

### 3.5 Writing to files

So far we've been running commands and outputting the results into the terminal. That's obviously useful but what if you want to save the results to another file?

Type this:



```
head -1 Styphi.gff > first_Styphi_line.txt
```

(no output)

It may look like nothing has happened. This is because the `>` character has *redirected* the output of the `head` command. Instead of writing to the *standard output* (your terminal) it sent the output into the file `first_Styphi_line.txt`. Note that tab completion works for `Styphi.gff` because it exists but doesn't work for `first_Styphi_line.txt` because it doesn't exist yet.

### 3.6 cat

`cat` is another way of reading files, but unlike `less` it just throws the entire contents of the file onto your standard output. Try it on `first_Styphi_line.txt`



```
cat first_Styphi_line.txt
```

```
##gff-version 3
```

We don't need `first_Styphi_line.txt` any more so delete it by typing



```
rm first_Styphi_line.txt
```

(no output)

The `cat` command can also be given the names of multiple files, one after the other and it will just output the contents of all files. The order in which the files are displayed is determined by the order in which they appear in the command line. You can use this concept and the `>` symbol to join files together into a single file.

Having looked at the beginning and end of the `Styphi.gff` file you should notice that in the GFF file the annotation comes first, then the DNA sequence at the end. If you had two separate files containing the annotation and the DNA sequence, it is possible to concatenate or join the two together to make a single file like the `Styphi.gff` file you have just looked at.

For example, we have two separate files, `Styphi.noseq.gff` and `Styphi.fa`, that contain the annotation and DNA sequence, respectively for the *Salmonella typhi* CT18 genome. To join together these files type:



```
cat Styphi.noseq.gff Styphi.fa > Styphi.concatenated.gff
```

(no output)

The files `Styphi.noseq.gff` and `Styphi.fa` will be joined together and written to a file called `Styphi.concatenated.gff`.

The `>` symbol in the command line directs the output of the `cat` program to the designated file `Styphi.concatenated.gff`. Use the command `ls` to check for the presence of this file.



```
ls
```

```
files.ipynb Styphi.concatenated.gff Styphi.gff  
Pfalciparum.bed Styphi.fa Styphi.noseq.gff
```

### 3.7 wc - counting

The command `wc` counts lines, words or characters.

There are two ways you could use it:



```
wc -l Styphi.gff
```

```
88961 Styphi.gff
```

or



```
cat Styphi.gff | wc -l
```

```
88961
```

Both give a similar answer. In the first example you tell `wc` the file that you want it to review (`Styphi.gff`) and pass the `-l` option to say that you're only interested in the number of lines.

In the second example you use the `|` symbol which is also known as the *pipe* symbol. This *pipes* the output of `cat Styphi.gff` into the input of `wc -l`. This means that you can also use the same `wc` tool to count other things. For example to count the number of files that are listed by `ls` type:



```
ls | wc -l
```

```
6
```

You can connect as many commands as you want. For example, type:



```
ls | grep ".gff" | wc -l
```

```
3
```

What does this command do? You will learn more about the `grep` command later in this course.

### 3.8 sort - sorting values

The `sort` lets you sort the contents of the input. When you sort the input, lines with identical content end up next to each other in the output. This is useful as the output can then be fed to the `uniq` command (see below) to count the number of unique lines in the input.

To sort the contents of a BED file type:

```
sort Pfalciparum.bed
```

Now type:



```
sort Pfalciparum.bed | head
```

```
01 104936 105441 PFA0115w 1
01 107429 108580 PFA0120c -1
01 110984 116033 EBA181 -1
01 11513 12397 RNAzID:13 1
01 119275 121483 FIKK1 -1
01 124752 125719 PFA0135w 1
01 126553 128375 PFA0140c -1
01 129194 131074 PFA0145c -1
01 132320 133858 PFA0150c -1
01 134587 139491 PFA0155c -1
```



```
sort Pfalciparum.bed | tail
```

```
14 979397 979586 RNAzID:2132 1
14 981211 982551 PF14TR004 1
14 981536 981592 RNAzID:2134 -1
14 982830 982889 RNAzID:2136 -1
14 983283 984503 PF14_0232 -1
14 985307 987697 PF14_0233 -1
14 987657 987729 RNAzID:2137 1
14 989162 992872 PF14_0234 1
14 993594 994242 PF14_0235 1
14 995103 1000448 PF14_0236 -1
```

To sort the contents of a BED file on position, type the following command.

```
sort -k 2 -n Pfalciparum.bed
```

The `sort` command can sort by multiple columns e.g. 1st column and then 2nd column by specifying successive `-k` parameters in the command. Type the following commands:



```
sort -k 2 -n Pfalciparum.bed | head
```

```
06 653 1432 PFF0005c -1
14 1394 5344 PF14_0001 1
14 2215 5392 PF14TR001 1
06 3503 12835 VAR 1
```

```
09 6841 7670 RNAzID:4487 1
14 7113 7207 RNAzID:1975 1
14 7209 8539 RIF -1
11 8419 9249 RNAzID:585 1
03 8435 8527 RNAzID:2735 -1
14 8936 9033 RNAzID:1976 1
```



```
sort -k 2 -n Pfalciparum.bed | tail
```

```
14 3272513 3273783 RIF 1
14 3274613 3274669 RNAzID:2440 -1
14 3276165 3277436 RIF 1
14 3279435 3280597 RIF 1
14 3282002 3282056 RNAzID:2456 1
14 3282664 3283687 PF14_0771 1
14 3285383 3285466 RNAzID:2463 -1
14 3285835 3286938 RIF 1
14 3289946 3290002 RNAzID:2468 1
14 3290888 3291436 PF14_0773 1
```

Why not have a look at the manual for `sort` to see what these options do? Remember that you can type `/` followed by a search phrase, `n` to find the next search hit, `N` to find the previous search hit and `q` to exit.

```
man sort
```

### 3.9 uniq - finding unique values

The `uniq` command extracts unique lines from the input. It is usually used in combination with `sort` to count unique values in the input.

To get the list of chromosomes in the Pfalciparum bed file type:



```
awk '{ print $1 }' Pfalciparum.bed | sort | uniq
```

```
01
02
03
04
05
06
07
08
09
10
11
12
13
14
```

How many chromosomes are there? You will learn more about the `awk` command later in this course.

Warning: `uniq` is really stupid; it can only spot that two lines are the same if they are right next to one another. You therefore almost always want to `sort` your input data before using `uniq`.

Do you understand how this command is working? Why not try building it up piece by piece to see what it does?

```
awk '{ print $1 }' Pfalciparum.bed | less
awk '{ print $1 }' Pfalciparum.bed | sort | less
awk '{ print $1 }' Pfalciparum.bed | sort | uniq | less
```

### 3.10 Exercises

Open up a new terminal window, navigate to the `files` directory in the `Unix` directory and complete the following exercise:

1. Use the `head` command to extract the first 500 lines of the file `Styphi.gff` and store the output in a new file called `Styphi.500.gff`.
2. Use the `wc` command to count the number of lines in the `Pfalciparum.bed` file.
3. Use the `sort` command to sort the file `Pfalciparum.bed` on chromosome and then gene position.
4. Use the `uniq` command to count the number of features per chromosome in the `Pfalciparum.bed` file. Hint: use the `man` command to look at the options for the `uniq` command. Or peruse the `wc` or `grep` manuals. There's more than one way to do it!

## 4 Searching inside files with `grep`

A common task is to extract information from large files. This can be achieved using the Unix command `grep`, which stands for “Globally search for a Regular Expression and Print”. The meaning of this acronym will become clear later, when we discuss Regular Expressions. First, we will consider simpler examples.

Before we start, change into the `Unix/grep` directory and double check that the following commands gives you a similar output:



```
pwd
ls
```

```
/home/manager/course_data/Module1_Unix/Notebooks/Unix/grep
answers.md          grep.ipynb  regex_example.txt
exercises.fasta    list_example.1 sequences.fasta
gene_expression.bed list_example.2
gene_expression_sneaky.bed list_example.3
```

### 4.1 Simple pattern matching

We will use a small example file (in “BED” format), which contains the expression levels of some genes. This is a column-based file, with a tab character between each column. There can be more than 10 columns, but only the first three are required to be a valid file. The file format is described in full here: <http://genome.ucsc.edu/FAQ/FAQformat#format1>. We will use the first 5 columns:

1. Sequence name
2. start position (starting from 0, not 1)
3. end position (starting from 0, not 1)
4. feature name
5. score (which is used to store the gene expression level in our examples).

Here is the contents of the first example BED file used in this course:



```
cat gene_expression.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr2 20 35 gene3 0 +
chr2 110 200 Gene4 4 -
chr3 1000 2000 gene5 100 +
chr10 1 100 gene6 11 -
chrX 60 90 Gene7 2 +
chrY 80 120 GENE8 42 -
```

In reality, such a file could contain 100,000s of lines, so that it is not practical to read manually. Suppose we are interested in all the genes from chromosome 2. We can find all these lines using `grep`:



```
grep chr2 gene_expression.bed
```

```
chr2 20 35 gene3 0 +
chr2 110 200 Gene4 4 -
```

This has shown us all the lines that contain the string “chr2”.

We can use a pipe to then just extract the genes that are on the positive strand, using `grep` a second time:



```
grep chr2 gene_expression.bed | grep +
```

```
chr2 20 35 gene3 0 +
```

However, since `grep` is reporting a match to a string *anywhere* on a line, such simple searches can have undesired consequences. For example, consider the result of doing a similar search for all the genes in chromosome 1:



```
grep chr1 gene_expression.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr10 1 100 gene6 11 -
```

Oops! We found genes in chromosome 10, because “chr1” is a substring of “chr10”.

Or consider the following file, where the genes have unpredictable names (which is not unusual for bioinformatics data).



```
cat gene_expression_sneaky.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr1 350 500 sneaky-gene3 1000 +
chr2 20 35 gene4 0 +
chr2 110 200 gene5 4 -
chr3 1000 2000 gene6 100 +
chr8 20 100 chr11.gene1 1000 -
chr10 1 100 gene7 11 -
chr11 20 100 sneaky-gene8 1000 +
```

Now we try to find genes on chromosome 1 that are on the negative strand. We put the minus sign in quotes, to stop Unix interpreting this as an option to `grep`, as opposed to the string we are searching for:



```
grep chr1 gene_expression_sneaky.bed | grep '-'
```

```
chr1 350 500 gene2 1000 -
chr1 350 500 sneaky-gene3 1000 +
chr8 20 100 chr11.gene1 1000 -
chr10 1 100 gene7 11 -
chr11 20 100 sneaky-gene8 1000 +
```

The extra lines are found by `grep` because of matches in columns we were not expecting to match. Remember, `grep` is reporting these lines because they each contain the strings “chr1” and “-” *somewhere*.

We need a way to make searching with `grep` more specific.

## 4.2 Regular expressions

Regular expressions provide the solution to the above problems. They are a way of defining more specific patterns to search for.

### 4.2.1 Matching the start and end of lines

First, we can specify that a match must be at the start of a line using the symbol “`^`”, which means “start of line”. Without the `^`, we find any match to “chr1”:



```
grep chr1 gene_expression_sneaky.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr1 350 500 sneaky-gene3 1000 +
chr8 20 100 chr11.gene1 1000 -
chr10 1 100 gene7 11 -
chr11 20 100 sneaky-gene8 1000 +
```

However, notice the effect of searching for `^chr1` instead. Note that we put the regular expression in quotes, to avoid Unix errors. Not using quotes may or may not give an error, but it is safest to use quotes for anything but the simplest of searches.



```
grep '^chr1' gene_expression_sneaky.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr1 350 500 sneaky-gene3 1000 +
chr10 1 100 gene7 11 -
chr11 20 100 sneaky-gene8 1000 +
```

Good! We have removed the match to the badly-named gene “chr11.gene1”, which is on chromosome 8. Now we want to avoid matching chromosomes 10 and 11. This can be done by also looking for a “tab” character, which is represented by writing `\t`. For technical reasons, which are beyond the scope of this course, we must also put a dollar sign before the quotes to make any search involving a tab character work.





```
grep $'^chr1\t' gene_expression_sneaky.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr1 350 500 sneaky-gene3 1000 +
```

To find the genes on the negative strand, all that remains is to match a minus sign at the *end* of the line (so that we do not find “sneaky-gene3”). We can do this using the dollar “\$”, which means “end of line”.



```
grep $'^chr1\t' gene_expression_sneaky.bed | grep '\-$'
```

```
chr1 350 500 gene2 1000 -
```

### 4.2.2 Wildcards and alphabets

Another special character in regular expressions is the dot: “.”. This stands for any single character. For example, this finds all matches to chromosomes 1-9, and chromosomes X and Y:



```
grep $'^chr.\t' gene_expression.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr2 20 35 gene3 0 +
chr2 110 200 Gene4 4 -
chr3 1000 2000 gene5 100 +
chrX 60 90 Gene7 2 +
chrY 80 120 GENE8 42 -
```

In fact, the earlier command that found all genes on chromosome 1 that are on the negative strand, could be found with a single call to `grep` instead of two calls piped together. To do this, we need a regular expression that finds lines that:

- start with `chr1`, then a tab character
- end with a minus
- have arbitrary characters between.

The asterisk “\*” has a special meaning: it says to match any number (including zero) of whatever character is before the \*. For example, the regular expression ‘AC\*G’ will match AG, ACG, ACCG, etc. The simpler, improved command is:



```
grep $'^chr1\t.*-$' gene_expression_sneaky.bed
```

```
chr1 350 500 gene2 1000 -
```

As well as matching any character using a dot, we can define any list of characters to match, using square brackets. For example, `[12X]` means match a 1, 2, or an X. This can be used to find all genes from chromosomes 1, 2 and X:



```
grep $'^chr[12X]\t' gene_expression.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr2 20 35 gene3 0 +
chr2 110 200 Gene4 4 -
chrX 60 90 Gene7 2 +
```

Or just the autosomes may be of interest. To do this we introduce two new features:

- Ranges can be given in square brackets, for example `[1-5]` will match 1, 2, 3, 4 or 5.
- The plus sign `“+”` has a special meaning that is similar to `“*”`. Instead of any number of matches (including zero), it looks for at least one match. To avoid simply matching a plus sign, it must be preceded by a backslash: `“\+”`. For example, the regular expression `‘AC\+G’` will match `ACG`, `ACCG`, `ACCCG` etc (but will not match `AG`).

Warning: Adding a backslash is often called *escaping* (e.g. *escape the plus symbol*). Depending on the software you’re using (and the options you give it), you may need to escape the symbol to indicate that you want its special regex meaning (e.g. multiple copies of the last character please) or its literal meaning (e.g. give me a `‘+’` symbol please). If your command isn’t working as you expect, try playing with these options and always test your regular expression before assuming it gave you the right answer.

The command to find the autosomes is:



```
grep $'^chr[0-9]\+\t' gene_expression.bed
```

```
chr1 10 100 gene1 10 +
chr1 350 500 gene2 1000 -
chr2 20 35 gene3 0 +
chr2 110 200 Gene4 4 -
chr3 1000 2000 gene5 100 +
chr10 1 100 gene6 11 -
```

## 4.3 Other `grep` options

The Unix command `grep` and regular expressions are extremely powerful and we have only scratched the surface of what they can do. Take a look at the manual (by typing `man grep`) to get an idea. A few particularly useful options are discussed below.

### 4.3.1 Counting matches

A common use-case is counting matches within files. Instead of output each matching line, the option `“-c”` tells `grep` to report the number of lines that matched. For example, the number of genes in the autosomes in the above example can be found by simply adding `-c` to the command.



```
grep -c $'^chr[0-9]\+\t' gene_expression.bed
```

```
6
```

### 4.3.2 Case sensitivity

By default, `grep` is case-sensitive. It can be useful to ignore the distinction between upper and lower case using the option “`-i`”. Suppose we have a file of sequences, and want to find the sequences that contain the string `ACGT`. It is not unusual to come across files that have a mix of upper and lower case nucleotides. Consider this FASTA file:



```
cat sequences.fasta
```

```
>sequence1
aACGTaaacaca
>sequence2
TacgtAAAAA
>sequence3
AAAAAAAAA
>sequence4
agcACgtAA
```

A simple search for `ACGT` will not return all the results:



```
grep ACGT sequences.fasta
```

```
aACGTaaacaca
```

However, making the search case-insensitive solves the problem.



```
grep -i ACGT sequences.fasta
```

```
aACGTaaacaca
TacgtAAAAA
agcACgtAA
```

### 4.3.3 Searching in more than one file

So far, we have restricted to searches in one file, but `grep` can be given a list of files in which to search. As an example, we are given three files called `list_example.1`, `list_example.2`, and `list_example.3`. They are simple lists of genes, for illustrative purposes. For example, the first file looks like this:



```
cat list_example.1
```

```
gene1
gene2
gene3
gene4
gene5
```

Which files contain “gene1”?



```
grep '^gene1$' list_example.1 list_example.2
```

```
list_example.1:gene1
```

`gene1` only appears in the file `list_example.1`. The output format of `grep` has now changed, because it was given a list of files. The format is:

- filename:line\_that\_matches

ie, the name of the file has been added to the start of each matching line.

For convenience, there’s also a way of specifying all of the list examples:



```
echo list_example.*
```

```
list_example.1 list_example.2 list_example.3
```



```
grep '^gene1$' list_example.*
```

```
list_example.1:gene1
```

How about `gene42`?



```
grep '^gene42$' list_example.*
```

```
list_example.2:gene42
list_example.3:gene42
list_example.3:gene42
```

`gene42` appears once in `list_example.2` and twice in `list_example.3`.

#### 4.3.4 Inverting matches

By default, `grep` reports all lines that do match the regular expression. Sometimes it is useful to filter a file, by reporting lines that *do not* match the regular expression. Using the option “`-v`” makes `grep` “invert” the output. For example, we could exclude genes from autosomes in the BED file from earlier.



```
grep -v $'^chr[0-9]\+\t' gene_expression.bed
```

```
chrX 60 90 Gene7 2 +
chrY 80 120 GENE8 42 -
```

#### 4.4 Replacing matches to regular expressions

Finally, we show how to replace every match to a regular expression with something else, using the command “`sed`”. The general form of this is:

```
sed 's/regular expression/new string/' input_file
```

This will output a new version of the input file, with each match to the regular expression replaced with “new string”. For example:



```
sed 's/^chr/chromosome/' gene_expression.bed
```

```
chromosome1 10 100 gene1 10 +
chromosome1 350 500 gene2 1000 -
chromosome2 20 35 gene3 0 +
chromosome2 110 200 Gene4 4 -
chromosome3 1000 2000 gene5 100 +
chromosome10 1 100 gene6 11 -
chromosomeX 60 90 Gene7 2 +
chromosomeY 80 120 GENE8 42 -
```

#### 4.5 Exercises

The following exercises all use the FASTA file `exercises.fasta`. Before starting the exercises, open a new terminal and navigate to the `grep/` directory, which contains `exercises.fasta`.

Use `grep` to find the answers. Hint: some questions require you to use `grep` twice, and possibly some other Unix commands.

1. Make a `grep` command that outputs just the lines with the sequence names.
2. How many sequences are in the file?
3. Do any sequence names have spaces in them? What are their names?
4. Make a `grep` command that outputs just the lines with the sequences, not the names.
5. How many sequences contain unknown bases (an “n” or “N”)?
6. Are there any sequences that contain non-nucleotides (something other than A, C, G, T or N)?
7. How many sequences contain the 5’ cut site GCWGC (where W can be an A or T) for the restriction enzyme `AceI`?
8. Are there any sequences that have the same name? You do not need to find the actual repeated names, just whether any names are repeated. (Hint: it may be easier to first discover how many unique names there are).

## 5 File processing with AWK

AWK is a programming language named after the initials of its three inventors: Alfred Aho, Peter Weinberger, and Brian Kernighan. AWK is incredibly powerful at processing files, particularly column-based files, which are commonplace in Bioinformatics. For example, BED, GFF, and SAM files.

Although long programs, put into a separate file, can be written using AWK, we will use it directly on the command line. Effectively, these are very short AWK programs, often called “one-liners”.

Before we start, change into the `Unix/awk` directory and double check that the following commands gives you a similar output:



```
pwd
ls
```

```
/home/manager/course_data/Module1_Unix/Notebooks/Unix/awk
answers.md awk.ipynb exercises.bed genes.gff
```

### 5.1 Extracting columns from files

`awk` reads a file line-by-line, splitting each line into columns. This makes it easy to do simple things like extract a column from a file. We will use the following GFF file for our examples.



```
cat genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=gene1;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 source5 repeat 10000 14000 1 + . name=ALU
chr2 source2 gene 10000 1200 0.95 + 0
chr2 source1 gene 50 900 0.4 - 0 name=gene2;product=gene2
↪ protein
chr3 source1 gene 200 210 0.8 . 0 name=gene3
chr4 source3 repeat 300 400 1 + . name=ALU
chr10 source2 repeat 60 70 0.78 + . name=LINE1
chr10 source2 repeat 150 166 0.84 + . name=LINE2
chrX source1 gene 123 456 0.6 + 0 name=gene4;product=unknown
```

The columns in the GFF file are separated by tabs and have the following meanings:

1. Sequence name
2. Source - the name of the program that made the feature
3. Feature - the type of feature, for example gene or CDS
4. Start position
5. Stop position
6. Score
7. Strand (+ or -)
8. Frame (0, 1, or 2)

9. Optional extra information, in the form `key1=value1;key2=value2;...`

The score, strand, and frame can be set to `.` if it is not relevant for that feature. The final column 9 may or may not be present and could contain any number of key, value pairs.

We can use `awk` to just print the first column of the file. `awk` calls the columns `$1`, `$2`, ... etc, and the complete line is called `$0`.



```
awk -F"\t" '{print $1}' genes.gff
```

```
chr1
chr1
chr1
chr2
chr2
chr3
chr4
chr10
chr10
chrX
```

A little explanation is needed.

- The option `-F"\t"` was needed to tell `awk` that the columns are separated by tabs (more on this later).
- For each line of the file, `awk` does what is inside the curly brackets. In this case, we simply print the first column.

The repeated chromosome names are not nice. It is more likely to want to know just the unique names, which can be found by piping into the Unix command `sort`.



```
awk -F"\t" '{print $1}' genes.gff | sort -u
```

```
chr1
chr10
chr2
chr3
chr4
chrX
```

## 5.2 Filtering the input file

Similarly to `grep`, `awk` can be used to filter out lines of a file. However, since `awk` is column-based, it makes it easy to filter based on properties of any columns of interest. The filtering criteria can be added before the braces. For example, the following extracts just chromosome 1 from the file.



```
awk -F"\t" '$1=="chr1" {print $0}' genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=genel;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 source5 repeat 10000 14000 1 + . name=ALU
```

There are two important things to note from the above command:

1. `$1=="chr1"` means that column 1 must be *exactly* equal to "chr1". This means that "chr10" is not found.
2. The "`{print $0}`" part only happens when the first column is equal to "chr1", otherwise `awk` does nothing (the line gets ignored).

Awk commands are made up of two parts, a *pattern* (e.g. `$1=="chr1"`) and an *action* (e.g. `print $0`) which is contained in curly braces. The *pattern* defines which lines the *action* is applied to.

In fact, the action (the part in curly braces) can be omitted in this example. `awk` assumes that you want to print the whole line, unless it is told otherwise. This gives a simple method of filtering based on columns.



```
awk -F"\t" '$1=="chr1"' genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=genel;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 source5 repeat 10000 14000 1 + . name=ALU
```

You might remember using another of `awk`'s defaults in a previous exercise. In that example we supplied an action but no pattern. In this case, `awk` assumes that you want to apply the action to every line in the file. For example:



```
awk -F"\t" '{print $1}' genes.gff
```

```
chr1
chr1
chr1
chr2
chr2
chr3
chr4
chr10
chr10
chrX
```

Multiple patterns can be combined using "`&&`" to mean "and". For example, to find just the genes from chromosome 1:



```
awk -F"\t" '$1=="chr1" && $3=="gene"' genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=genel;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
```

The entire line need not be printed (remember, if not specified, `awk` assumes a `print $0`). Suppose we want only the sources of the genes on chromosome 1:





```
awk -F"\t" '$1=="chr1" && $3=="gene" {print $2}' genes.gff | sort -u
```

```
source1
source2
```

Similarly to using “&&” for “and”, there is “||” to mean “or”. To find features that are repeats or made by the tool “source2”:



```
awk -F"\t" '$2=="source2" || $3=="repeat" ' genes.gff
```

```
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 source5 repeat 10000 14000 1 + . name=ALU
chr2 source2 gene 10000 1200 0.95 + 0
chr4 source3 repeat 300 400 1 + . name=ALU
chr10 source2 repeat 60 70 0.78 + . name=LINE1
chr10 source2 repeat 150 166 0.84 + . name=LINE2
```

So far, we have only used strings for the filtering. Numbers can also be used. We could ask awk to return all the genes on chromosome 1 that start before position 1100:



```
awk -F"\t" '$1=="chr1" && $3=="gene" && $4 < 1100' genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=genel;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
```

Instead of looking for exact matches to strings, regular expressions can be used. The symbol “~” is used instead of “==”. For example, to find all the autosomes, we need to use a regular expression for matches to the first column. The regular expression is written between forward slashes.



```
awk -F"\t" '$1 ~ /^chr[0-9]+$/ ' genes.gff
```

```
chr1 source1 gene 100 300 0.5 + 0 name=genel;product=unknown
chr1 source2 gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 source5 repeat 10000 14000 1 + . name=ALU
chr2 source2 gene 10000 1200 0.95 + 0
chr2 source1 gene 50 900 0.4 - 0 name=gene2;product=gene2
  ↪ protein
chr3 source1 gene 200 210 0.8 . 0 name=gene3
chr4 source3 repeat 300 400 1 + . name=ALU
chr10 source2 repeat 60 70 0.78 + . name=LINE1
chr10 source2 repeat 150 166 0.84 + . name=LINE2
```

Like with `grep`, matches can be inverted. `grep` has the option `-v`, but with `awk` we use “!~” to mean “does not match”. This inverts the previous example:



```
awk -F"\t" '$1 !~ /^chr[0-9]+$/ ' genes.gff
```

```
chrX source1 gene 123 456 0.6 + 0 name=gene4;product=unknown
```

If we do not specify a column, `awk` looks for a match anywhere in the whole line (it assumes we wrote `$0 ~ /regex/`). So, in some sense, `awk` can be used as a replacement for `grep`:



```
awk '/repeat/' genes.gff
```

```
chr1 source5 repeat 10000 14000 1 + . name=ALU
chr4 source3 repeat 300 400 1 + . name=ALU
chr10 source2 repeat 60 70 0.78 + . name=LINE1
chr10 source2 repeat 150 166 0.84 + . name=LINE2
```

(the `-F"\t"` was omitted because the match is to the whole line, so how the columns are separated is not relevant.)



```
grep repeat genes.gff
```

```
chr1 source5 repeat 10000 14000 1 + . name=ALU
chr4 source3 repeat 300 400 1 + . name=ALU
chr10 source2 repeat 60 70 0.78 + . name=LINE1
chr10 source2 repeat 150 166 0.84 + . name=LINE2
```

However, with `awk` we can easily pull out information from the matching lines. Suppose we want to know which chromosomes have repeats. It is easy with `awk`.



```
awk -F"\t" '/repeat/ {print $1}' genes.gff | sort -u
```

```
chr1
chr10
chr4
```

### 5.3 Sanity checking files

Never, ever trust the contents of Bioinformatics files (even if you made them!). We now have enough skills to do some basic sanity checking of a GFF file. For example, to check that every gene has been assigned a strand:



```
awk -F"\t" '$3=="gene" && !($7 == "+" || $7 == "-)' genes.gff
```

```
chr3 source1 gene 200 210 0.8 . 0 name=gene3
```

Something went wrong when this file was made: `gene3` has an unknown strand.

Do the start and end coordinates of all the features make sense?



```
awk -F"\t" '$5 < $4' genes.gff
```

```
chr2  source2  gene 10000 1200 0.95  +      0
```

According to the file, this gene starts at position 10000 and ends at position 1200, which does not make sense. Also, it has no name (the final optional column is empty). We could check if there are any other genes with no name. One way to do this is to use the special variable “NF”, which is the number of columns (fields) in the current line. Since the final column is optional, each line might have 8 or 9 columns. We need to write a command that will check:

- If the feature is a gene, and if it is:
- check if the number of columns is less than 9. When there are 9 columns, check if there is a name defined.



```
awk -F"\t" '$3=="gene" && (NF<9 || $NF !~/name/)' genes.gff
```

```
chr2  source2  gene 10000 1200 0.95  +      0
```

Note the distinction between NF (the number of columns) and “\$NF” (the contents of the final column).

As promised earlier, we now consider the relevance of the option “-F“\t””, to tell awk that the columns in the input file are separated with tab characters. If we forgot to use this option, then awk will use its default behaviour, which is to separate on *any* whitespace (which usually means tabs and/or spaces). However, consider the final column of the file - it can contain whitespace, which means that messy things happen. Suppose we try to extract the optional extra final column of the file, when it is present. Compare the effect of running awk with and without “-F“\t””.



```
awk -F"\t" 'NF>8 {print $NF}' genes.gff
```

```
name=genel;product=unknown
name=recA;product=RecA protein
name=ALU
name=gene2;product=gene2 protein
name=gene3
name=ALU
name=LINE1
name=LINE2
name=gene4;product=unknown
```



```
awk 'NF>8 {print $NF}' genes.gff
```

```
name=genel;product=unknown
protein
name=ALU
protein
name=gene3
name=ALU
```

```
name=LINE1
name=LINE2
name=gene4;product=unknown
```

One more sanity check: each line should have 8 or 9 columns (remembering to use `-F"\t"`!)



```
awk -F"\t" 'NF<8 || NF>9' genes.gff
```

(no output)

There was no output, which means that every line does indeed have 8 or 9 columns.

## 5.4 Changing the output

In addition to filtering, `awk` can be used to change the output.

Every value in a column could be changed to something else, for example suppose we want to change the source column (column number 2) to something else.



```
awk -F"\t" '{ $2="new_source"; print $0 }' genes.gff
```

```
chr1 new_source gene 100 300 0.5 + 0 name=gene1;product=unknown
chr1 new_source gene 1000 1100 0.9 - 0 name=recA;product=RecA protein
chr1 new_source repeat 10000 14000 1 + . name=ALU
chr2 new_source gene 10000 1200 0.95 + 0
chr2 new_source gene 50 900 0.4 - 0 name=gene2;product=gene2 protein
chr3 new_source gene 200 210 0.8 . 0 name=gene3
chr4 new_source repeat 300 400 1 + . name=ALU
chr10 new_source repeat 60 70 0.78 + . name=LINE1
chr10 new_source repeat 150 166 0.84 + . name=LINE2
chrX new_source gene 123 456 0.6 + 0 name=gene4;product=unknown
```

This is close, but look carefully at the output. What happened? The output is not tab-separated, but is instead separated with spaces. To restore the tabs, we need to use another special variable called `OFS` (Output Field Separator), and change it before `awk` does any processing of the input file. This can be achieved by adding `"BEGIN{OFS="\t"}"`, as in the next example. Before `awk` reads any lines of the file it runs the `BEGIN` block of code, which in this case changes `OFS` to be a tab character.



```
awk -F"\t" 'BEGIN{OFS="\t"} { $2="new_source"; print $0 }' genes.gff
```

```
chr1 new_source gene 100 300 0.5 + 0 name=gene1;product=
↪ unknown
chr1 new_source gene 1000 1100 0.9 - 0 name=recA;product=RecA
↪ protein
chr1 new_source repeat 10000 14000 1 + . name=ALU
chr2 new_source gene 10000 1200 0.95 + 0
chr2 new_source gene 50 900 0.4 - 0 name=gene2;product=gene2
↪ protein
```

```
chr3 new_source gene 200 210 0.8 . 0 name=gene3
chr4 new_source repeat 300 400 1 + . name=ALU
chr10 new_source repeat 60 70 0.78 + . name=LINE1
chr10 new_source repeat 150 166 0.84 + . name=LINE2
chrX new_source gene 123 456 0.6 + 0 name=gene4;product=
↪ unknown
```

## 5.5 Processing the data

More in-depth processing is possible. For example, we could print the length of each repeat (and then sort the results numerically)



```
awk -F"\t" '$3=="repeat" {print $5 - $4 + 1}' genes.gff | sort -n
```

```
11
17
101
4001
```

Perhaps we would like to know the total length of the repeats. We need to use a variable to add up the total lengths and print the final total. In the same way that `awk` has a `BEGIN` block, it can also be given an `END` block that is only run when `awk` has finished reading all lines of the input file.



```
awk -F"\t" 'BEGIN{sum=0} $3=="repeat" \
           {sum = sum + $5 - $4 + 1} \
           END{print sum}' genes.gff
```

```
4130
```

The total repeat length was stored in a variable called `sum`. The previous `awk` command can be broken down into three parts:

1. The `BEGIN{sum=0}` sets `sum` to zero before any lines of the file are read.
2. `awk` reads each line of the file. Each time a repeat is found, the length of that repeat is added to `sum`.
3. Once all lines of the file have been read, `awk` runs the `END` block: `END{print sum}`. This prints the value of `sum`.

In fact, the command can be shortened a little. Adding a number to a variable is so common, that there is a shorthand way to write it. Instead of

```
sum = sum + $5 - $4 + 1
```

we can use

```
sum += $5 - $4 + 1
```

to get the same result.



```
awk -F"\t" 'BEGIN{sum=0} \
           $3=="repeat" {sum += $5 - $4 + 1} \
           END{print sum}' genes.gff
```

4130

Maybe we would like to know the mean score of the genes. We need to calculate the total score, and divide this by the number of genes. To keep track of the number of genes, we use a variable called `count`. Each time a new gene is found, 1 must be added to `count`. This could be done by writing

```
count = count + 1
```

but instead we will use the shorthand

```
count++
```



```
awk -F"\t" 'BEGIN{sum=0; count=0} \
           $3=="gene" {sum += $6; count++} \
           END{print sum/count}' genes.gff
```

0.691667

Finally, `awk` has a default behaviour that means we do not even need the `BEGIN` block. It can be completely omitted in this example because we are setting `sum` and `count` to zero. The first time `awk` sees a variable being used, it will set it to zero by default. For example, when `awk` reads the first line of the file, the piece of code

```
count++
```

tells `awk` to add 1 to `count`. However, if `awk` has not encountered the variable `count` before, it assumes it is zero (as if we had written `BEGIN{count=0}`), then adds 1 to it. The result is that `count` is equal to 1. Similar comments apply to the variable `sum`.



```
awk -F"\t" '$3=="gene" {sum += $6; count++} \
           END{print sum/count}' genes.gff
```

0.691667

If this confuses you, then be explicit and use the `BEGIN` block of code. The result is the same.

## 5.6 Exercises

The following exercises all use the BED file `exercises.bed`. Before starting the exercises, open a new terminal and navigate to the `awk/` directory, which contains `exercises.bed`.

Use `awk` to find the answers to the following questions about the file `exercises.bed`. Many questions will require using pipes (eg “`awk ... | sort -u`” for question 1).

1. What are the names of the contigs in the file?
2. How many contigs are there?
3. How many features are on the positive strand?
4. How many features are on the negative strand?
5. How many genes are there?
6. How many genes have no strand assigned to them (ie the final column is not there)?
7. Are any gene names repeated? (Hint: you do not need to find their names, just a yes or no answer. Consider the number of unique gene names.)
8. What is the total score of the repeats?
9. How many features are in contig-1?
10. How many repeats are in contig-1?
11. What is the mean score of the repeats in contig-1?

## 6 BASH scripts

So far, we have run single commands in a terminal. However, it is useful to be able to run multiple commands that process some data and produce output. These commands can be put into a separate file (ie a script), and then run on the input data. This has the advantage of reproducibility, so that the same analysis can be run on many input data sets.

### 6.1 First script

It is traditional when learning a new language (in this case BASH), to write a simple script that says “Hello World!”. We will do this now.

First, open a terminal and make a new directory in your home called `scripts`, by typing

```
cd
mkdir ~/scripts
```

Next open a text editor, which you will use to write the script. What text editors are available will depend on your system. For example, `gedit` in Linux. Do not try to use a word processor, such as Word! If you don’t already have a favorite, try `gedit` by running the following command:

```
gedit &
```

Type this into the text editor:

```
echo Hello World!
```

and save this to a file called `hello.sh` in your new `scripts` directory. This script will print `Hello World!` to the screen when we run it. First, in your terminal, check that the script is saved in the correct place.



```
cd scripts
ls hello.sh
```

```
hello.sh
```

If everything is OK, then next try to run the script. For now, we need to tell Unix that this is a bash script, and where it is:



```
bash hello.sh
```

```
Hello World!
```

### 6.2 Setting up a scripts directory

It would be nice if all our scripts could simply be run from anywhere in the filesystem, without having to tell Unix where the script is, or that it is a BASH script. This is how the built-in commands work, such as `cd` or `ls`.

To tell Unix that the script is a BASH script, make this the first line of the script:



```
#!/usr/bin/env bash
```

and remember to save the script again. This special line at the start of the file tells Unix that the file is a bash script, so that it expects bash commands throughout the file. There is one more change to be made to the file to tell Unix that it is a program to be run (it is “executable”). This is done with the command `chmod`. Type this into the terminal to make the file executable:



```
chmod +x hello.sh
```

(no output)

Now, the script can be run, but we must still tell Unix where the script is in the filesystem. In this case, it is in the current working directory, which is called “./”.



```
./hello.sh
```

Hello World!

The final thing to do is change our setup so that Unix can find the script without us having to explicitly say where it is. Whenever a command is typed into Unix, it has a list of directories that it searches through to look for the command. We need to add the new scripts directory to that list of directories. Try typing

```
echo $PATH
```

It returns a list of directories, which are all the places Unix will look for a command. Before we add the scripts directory to this list, check what happens if we try to run the script without telling Unix where it is:

```
hello.sh
bash: hello.sh: command not found
```

Unix did not find it! The command to run to add the scripts directory to `$PATH` is:



```
export PATH=$PATH:~/scripts/
```

(no output)

If you want this change to be permanent, ie so that Unix finds your scripts after you restart or logout and login again, add that line to the end of a file called `~/.bashrc`. If you are using a Mac, then the file should instead be `~/.bash_profile`. If the file does not already exist, then create it and put that line into it.

Now the script works, no matter where we are in the filesystem. Unix will check the scripts directory and find the file `hello.sh`. You can be *anywhere* in your filesystem, and simply running

```
hello.sh
```

will always work. Try it now.



```
hello.sh
```

```
Hello World!
```

In general, when making a new script, you can now copy and edit an existing script, or make a new one like this:

```
cd ~/scripts
touch my_script.sh
chmod +x my_script.sh
```

and then open `my_script.sh` in a text editor.

### 6.3 Getting options from the terminal and printing a help message

Usually, we would like a script to read in options from the user, such as the name of an input file. This would mean a script can be run like this:

```
my_script.sh input_file
```

Inside the script, the parameters provided by the user are given the names `$1`, `$2`, `$3` etc (do not confuse these with column names used by `awk`!). Here is a simple example that expects the user to provide a filename and a number. The script simply prints the filename to the screen, and then the first few lines of the file (the number of lines is determined by the number given by the user).



```
cat options_example.sh
```

```
#!/usr/bin/env bash

echo filename is: $1
echo

echo First $2 lines of file $1 are:
head -n $2 $1
```



```
options_example.sh test_file 2
```

```
filename is: test_file

First 2 lines of file test_file are:
test file line 1
test file line 2
```

The options have been used by the script, but the script itself is not very readable. It is better to use names instead of `$1` and `$2`. Here is an improved version of the script that does exactly the same as the previous script, but is more readable.



```
cat options_example.2.sh
```

```
#!/usr/bin/env bash
filename=$1
number_of_lines=$2

echo filename is: $filename
echo

echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

## 6.4 Checking options from the user

The previous scripts will have strange behaviour if the input is not as expected by the script. Many things could go wrong. For example:

- The wrong number of options are given by the user
- The input file does not exist.

Try running the script with different options and see what happens.

A convention with scripts is that it should output a help message if it is not run correctly. This shows anyone how the script should be run (including you!) without having to look at the code inside the script.

A basic check for this script would be to verify that two options were supplied, and if not then print a help message. The code looks like this:

```
if [ $# -ne 2 ]
then
    echo "usage: options_example.3.sh filename number_of_lines"
    echo
    echo "Prints the filename, and the given first number of lines of the file"
    exit
fi
```

You can copy this code into the start of any of your scripts, and easily modify it to work for that script. A little explanation:

- A special variable `$#` has been used, which is the number of options that were given by the user.
- The whole block of code has the form “`if [ $# -ne 2 ] then ... fi`”. This only runs the code between the `then` and `fi`, if `$#` (the number of options) is not 2.
- The line `exit` simply makes the script end, so that no more code is run.



```
options_example.3.sh
```

```
usage: options_example.3.sh filename number_of_lines

Prints the filename, and the given first number of lines of the file
```

Another check is that the input file really does exist. If it does not exist, then there is no point in trying to run any more code. This can be checked with another `if ... then ... fi` block of code:

```
if [ ! -f $filename ]

then
    echo "File '$filename' not found! Cannot continue"
    exit
fi
```

Putting this all together, the script now looks like this:



```
cat options_example.3.sh
```

```
#!/usr/bin/env bash
set -eu

# check that the correct number of options was given.
# If not, then write a message explaining how to use the
# script, and then exit.
if [ $# -ne 2 ]
then
    echo "usage: options_example.3.sh filename number_of_lines"
    echo
    echo "Prints the filename, and the given first number of lines of the file"
    exit
fi

# Use sensibly named variables
filename=$1
number_of_lines=$2

# check if the input file exists
if [ ! -f $filename ]
then
    echo "File '$filename' not found! Cannot continue"
    exit
fi

# If we are still here, then the input file was found
echo filename is: $filename
echo

echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

Two new features have also been introduced in this file:

1. The second line is “`set -eu`”. Without this line, if any line produces an error, the script will carry on regardless to the end of the script. Using the `-e` option, an error anywhere in the file will result in the script stopping at the line that produced the error, instead of continuing.

In general, it is best that the script stops at any error. The `-u` creates an error if you try to use a variable which doesn't exist. This helps to stop typos doing bad things to your analysis.

2. There are several lines starting with a hash `#`. These lines are "comment lines" that are not run. They are used to document the code, containing explanations of what is happening. It is good practice to comment your scripts!

The above script provides a template for writing your own scripts. The general method is:

1. Tell Unix that this is a BASH script, and to stop at the first error.
2. Check if the user ran the script correctly. If not, output a message telling the user how to run the script.
3. Check the input looks OK (in this case, that the input file exists).
4. Process the input.

## 6.5 Using variables to store output from commands

It can be useful to run a command and put the results into a variable. Recall that we stored the input from the user in sensibly named variables:

```
filename=$1
```

The part after the equals sign could actually be any command that returns some output. For example, running this in Unix

```
wc -l filename | awk '{print $1}'
```

returns the number of lines. In case you are wondering why the command includes `| awk '{print $1}'`, check what happens with and without the pipe to `awk`:



```
wc -l options_example.3.sh
```

```
31 options_example.3.sh
```



```
wc -l options_example.3.sh | awk '{print $1}'
```

```
31
```

With a small change, this can be stored in a variable and then used later.



```
filename=options_example.3.sh
line_count=$(wc -l $filename | awk '{print $1}')
echo There are $line_count lines in the file $filename
```

```
There are 31 lines in the file options_example.3.sh
```

## 6.6 Repeating analysis with loops

It is common in Bioinformatics to run the same analysis on many files. Suppose we had a script that ran one type of analysis, and wanted to repeat the same analysis on 100 different files. It would be tedious, and error-prone, to write the same command 100 times. Instead we can use a loop. As an example, we will just run the Unix command `wc` on each file but instead, in reality this would be a script that runs in-depth analysis. We can run `wc` on each of the files in the directory `loop_files/` with the following command.



```
for filename in loop_files/*; do wc $filename; done
```

```
2 8 28 loop_files/file.1
5 20 70 loop_files/file.2
6 24 84 loop_files/file.3
1 4 14 loop_files/file.4
0 0 0 loop_files/file.5
```

## 6.7 Exercises

1. Write a script that gets a filename from the user. If the file exists, it prints a nice human-readable message telling the user how many lines are in the file.
2. Use a loop to run the script from Exercise 1 on the files in the directory `loop_files/`.
3. Write a script that takes a GFF filename as input. Make the script produce a summary of various properties of the file. There is an example input file provided called `bash_scripts/exercise_3.gff`. Use your imagination! You could have a look back at the `awk` section of the course for inspiration. Here are some ideas you may wish to try:
  - Does the file exist?
  - How many records (ie lines) are in the file?
  - How many genes are in the file?
  - Is the file badly formatted in any way (eg wrong number of columns, do the coordinates look like numbers)?

## 7 UNIX Quick Reference Guide

### 7.1 Looking at files and moving them around

```
pwd # Tell me which directory I'm in
ls # What else is in this directory
ls .. # What is in the directory above me
ls foo/bar/ # What is inside the bar directory which is inside
           # the foo/ directory
ls -lah foo/ # Give the the details (-l) of all files and folders (-a)
           # using human readable file sizes (-h)
cd ../.. # Move up two directories
cd ../foo/bar # Move up one directory and down into the
           # foo/bar/ subdirectories
cp -r foo/ baz/ # Copy the foo/ directory into the baz/ directory
mv baz/foo .. # Move the foo directory into the parent directory
rm -r ../foo # remove the directory called foo/ from the parent directory
find foo/ -name "*.gff" # find all the files with a gff extension
                       # in the directory foo/
```

### 7.2 Looking in files

```
less bar.bed # scroll through bar.bed
grep chrom bar.bed | less -S # Only look at lines in bar.bed which have
                             # 'chrom' and don't wrap lines (-S)
head -20 bar.bed # show me the first 20 lines of bar.bed
tail -20 bar.bed # show me the last 20 lines
cat bar.bed # show me all of the lines (bad for big files)
wc -l bar.bed # how many lines are there
sort -k 2 -n bar.bed # sort by the second column in numerical order
awk '{print $1}' bar.bed | sort | uniq # show the unique entries in the
                                       # first column
```

### 7.3 Grep

```
grep foo bar.bed # show me the lines in bar.bed with 'foo' in them
grep foo baz/* # show me all examples of foo in the files immediately
              # within baz/
grep -r foo baz/ # show me all examples of foo in baz/ and every
                # subdirectory within it
grep '^foo' bar.bed # show me all of the lines begining with foo
grep 'foo$' bar.bed # show me all of the lines ending in foo
grep -i '^[acgt]$\ ' bar.bed # show me all of the lines which only have the
                             # characters a,c,g and t (ignoring their case)
grep -v foo bar.bed # don't show me any files with foo in them
```

### 7.4 Awk

```
awk '{print $1}' bar.bed # just the first column
```

```
awk '$4 ~ /^foo/' bar.bed # just rows where the 4th column starts with foo
awk '$4 == "foo" {print $1}' bar.bed # the first column of rows where
                                   # the 4th column is foo
awk -F"\t" '{print $NF}' bar.bed # ignore spaces and print the last column
awk -F"\t" '{print $(NF-1)}' bar.bed # print the penultimate column
awk '{sum+=$2} END {print sum}' bar.bed # print the sum of the second column
awk '/^foo/ {sum+=$2; count+=1} END {print sum/count}' bar.bed
                                   # ... print the average of the second value of lines starting with foo
```

## 7.5 Piping, redirection and more advanced queries

```
grep -hv '^#' bar/*.gff | awk -F"\t" '{print $1}' | sort -u
# grep => -h: don't print file names
#           -v: don't give me matching files
#           '^#': get rid of the header rows
#           'bar/*.gff': only look in the gff files in bar/
# awk => print the first column
# sort => -u: give me unique values

awk 'NR%10 == 0' bar.bed | head -20
# awk => NR: is the row number
#           NR%10: is the modulo (remainder) of dividing my 10
#           awk is therefore giving you every 10th line
# head => only show the first 20

awk '{l=($3-$2+1)}; (l<300 && $2>200000 && $3<250000)' exercises.bed
# Gives:
# contig-2 201156 201359 gene-67 24.7 -
# contig-4 245705 245932 gene-163 24.8 +
# Finds all of the lines with features less than 300 bases long which start
# after base 200,000 and end before base 250,000
# Note that this appears to have the action before the pattern. This is
# because we need to calculate the length of each feature before we use it
# for filtering. If they were the other way around, you'd get the line
# immediately after the one you want:
awk '(l<300 && $2>200000 && $3<250000) {l=($3-$2+1); print $0}' exercises.bed
# Gives:
# contig-2 201156 201359 gene-67 24.7 -
# contig-2 242625 243449 gene-68 46.5 +
```

## 7.6 A script

```
#!/usr/bin/env bash

set -e # stop running the script if there are errors
set -u # stop running the script if it uses an unknown variable
set -x # print every line before you run it (useful for debugging but annoying)

if [ $# -ne 2 ]
then
    echo "You must provide two files"
    exit 1 # exit the programme (and number > 0 reports that this is a failure)
```



```
fi

file_one=$1
file_two=$2

if [ ! -f $file_one ]
then
    echo "The first file couldn't be found"
    exit 2
fi

if [ ! -f $file_two ]
then
    echo "The second file couldn't be found"
    exit 2
fi

# Get the lines which aren't headers,
# take the first column and return the unique values
number_of_contigs_in_one=$(awk '$1 !~ /^#/ {print $1}' $file_one | sort -u | wc -l)
number_of_contigs_in_two=$(awk '/^[^#]/ {print $1}' $file_two | sort -u | wc -l)

if [ $number_of_contigs_in_one -gt $number_of_contigs_in_two ]
then
    echo "The first file had more unique contigs than the second"
    exit
elif [ $number_of_contigs_in_one -lt $number_of_contigs_in_two ]
then
    echo "The second file had more unique contigs"
    exit
else
    echo "The two files had the same number of contigs"
    exit
fi
```

## 7.7 Pro tips

- Use tab completion - it will save you time!
- Always have a quick look at files with `less` or `head` to double check their format
- Watch out for data in headers and that you don't accidentally `grep` some if you don't want them
- Watch out for spaces, especially if you're using `awk`; if in doubt, use `-F"\t"`
- Regular expressions are wierd, build them up slowly bit by bit
- If you did something smart but can't remember what it was, try typing `history` and it might have a record
- `man the_name_of_a_command` often gives you help
- Google is normally better at giving examples (prioritise `stackoverflow.com` results, they're normally good)

## 7.8 Build commands slowly

If you wanted me to calculate the sum of all of the scores for genes on contig-1 in a bed file, I'd probably run each of the following commands before moving onto the next:

```
head -20 bar.bed # check which column is which and if there are any headers
head -20 bar.bed | awk '{print $5}' # have a look at the scores
awk '{print $1}' bar.bed | sort -u | less # check the contigs don't look wierd
awk '{print $4}' bar.bed | sort -u | less # check the genes don't look wierd
awk '$4 ~ /gene-/' bar.bed | head -20 # check that I can spot genes
awk '($1 == "contig-1" && $4 ~ /gene-/)' bar.bed | head -20 # check I can find
                                                    # genes on contig-1

# check my algorithm works on a subset of the data
head -20 bar.bed | awk '($1 == "contig-1" && $4 ~ /gene-/) {sum+=$5}; END {print sum}'
# apply the algorithm to all of the data
awk '($1 == "contig-1" && $4 ~ /gene-/) {sum+=$5}; END {print sum}' bar.bed
```

## 7.9 Which tool should I use?

You should probably use awk if:

- your data has columns
- you need to do simple maths

You should probable use grep if:

- you're looking for files which contain some specific text (e.g. `grep -r foo bar/`: look in all the files in `bar/` for any with the word 'foo')

You should use find if:

- you know something about a file (like it's name or creation date) but not where it is
- you want a list of all the files in a subdirectory and its subdirectories etc.

You should write a script if:

- your code doesn't fit on one line
- it's doing something you might want to do again in 3 months
- you want someone else to be able to do it without asking loads of questions
- you're doing something sensitive (e.g. deleting loads of files)
- you're doing something lots of times

You should probably use less or head:

- always, you should always use less or head to check intermediary steps in your analysis



## HTS data formats and Quality Control

petr.danecek@sanger.ac.uk



### Data Formats

#### FASTQ

- Unaligned read sequences with base qualities

#### SAM/BAM

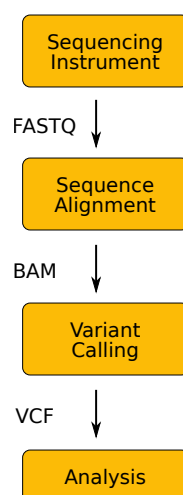
- Unaligned or aligned reads
- Text and binary formats

#### CRAM

- Better compression than BAM

#### VCF/BCF

- Flexible variant call format
- Arbitrary types of sequence variation
- SNPs, indels, structural variations



Specifications maintained by the Global Alliance for Genomics and Health

# FASTQ

- Simple format for raw unaligned sequencing reads
- Extension to the FASTA file format
- Sequence and an associated per base quality score

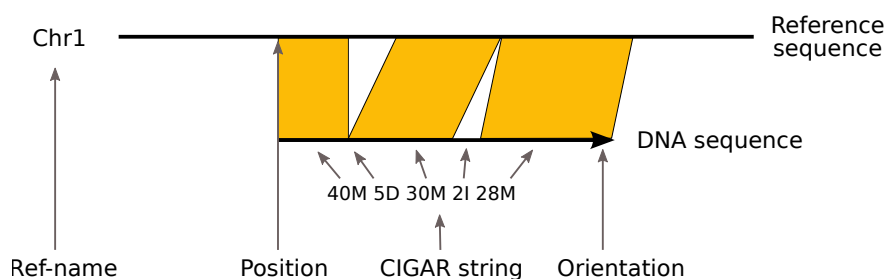
```
@ERR007731.739 IL16_2979:6:1:9:1684/1
CTTGACGACTTGAAAAATGACGAAATCACTAAAAACGTGAAAAATGAGAAATG
+
BBBCBCCCCCCCCCABBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC>@>B
@ERR007731.740 IL16_2979:6:1:9:1419/1
AAAAAAAAAAGATGTCATCAGCACATCAGAAAAGAAGCAACTTTAAAACTTTTC
+
BBABBBABABAABABABBBBAAA>@B@BBAA@4AAA>.>BAA@779:AAA@A
```

- Quality encoded in ASCII characters with decimal codes 33-126
  - ASCII code of "A" is 65, the corresponding quality is  $Q = 65 - 33 = 32$
  - Phred quality score:  $P = 10^{-Q/10}$   
`perl -e 'printf "%d\n",ord("A")-33;'`
- Beware: multiple quality scores were in use!
  - Sanger, Solexa, Illumina 1.3+
- Paired-end sequencing produces two FASTQ files

# SAM / BAM

## SAM (Sequence Alignment/Map) format

- Unified format for storing read alignments to a reference genome
- Developed by the 1000 Genomes Project group (2009)
- One record (a single DNA fragment alignment) per line describing alignment between fragment and reference
- 11 fixed columns + optional key:type:value tuples



Note that BAM can contain

- unmapped reads
- multiple alignments of the same read
- supplementary (chimeric) reads



# Flags

Hex	Dec	Flag	Description
0x1	1	PAIRED	paired-end (or multiple-segment) sequencing technology
0x2	2	PROPER_PAIR	each segment properly aligned according to the aligner
0x4	4	UNMAP	segment unmapped
0x8	8	MUNMAP	next segment in the template unmapped
0x10	16	REVERSE	SEQ is reverse complemented
0x20	32	MREVERSE	SEQ of the next segment in the template is reversed
0x40	64	READ1	the first segment in the template
0x80	128	READ2	the last segment in the template
0x100	256	SECONDARY	secondary alignment
0x200	512	QCFAIL	not passing quality controls
0x400	1024	DUP	PCR or optical duplicate
0x800	2048	SUPPLEMENTARY	supplementary alignment

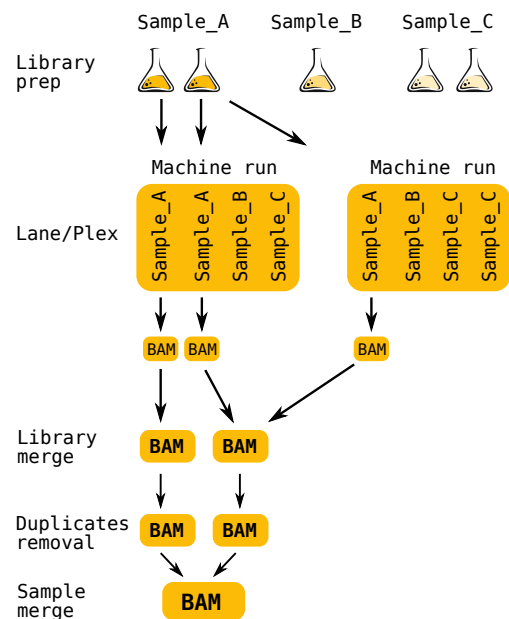
## Bit operations made easy

- python
  - `0x1 | 0x2 | 0x20 | 0x80 .. 163`
  - `bin(163) .. 10100011`
- samtools flags
  - `0xa3 163 PAIRED,PROPER_PAIR,MREVERSE,READ2`

# Optional tags

Each lane has a unique RG tag that contains meta-data for the lane  
RG tags

- ID: SRR/ERR number
- PL: Sequencing platform
- PU: Run name
- LB: Library name
- PI: Insert fragment size
- SM: Individual
- CN: Sequencing center



## BAM (Binary Alignment/Map) format

- Binary version of SAM
- Developed for fast processing and random access
  - BGZF (Block GZIP) compression for indexing

## Key features

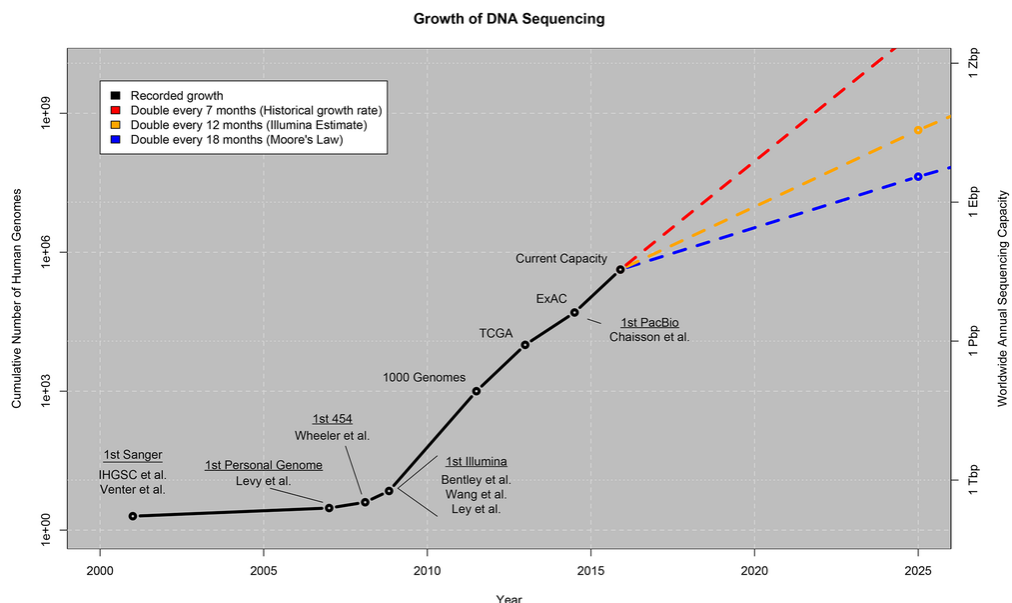
- Can store alignments from most mappers
- Supports multiple sequencing technologies
- Supports indexing for quick retrieval/viewing
- Compact size (e.g. 112Gbp Illumina = 116GB disk space)
- Reads can be grouped into logical groups e.g. lanes, libraries, samples
- Widely supported by variant calling packages and viewers

## Reference based Compression

BAM files are too large

- ~1.5-2 bytes per base pair

Increases in disk capacity are being far outstripped by sequencing technologies



## Reference based Compression

BAM files are too large

- ~1.5-2 bytes per base pair

Increases in disk capacity are being far outstripped by sequencing technologies  
BAM stores all of the data

- Every read base
- Every base quality
- Using a single conventional compression technique for all types of data

```
Reference sequence: ACGTACGTACGTACGTACGTACGTACGTACGTAC
read 1:             ACGTACGTACGTACGTACGTGC
read 2:             TACGTACGCACGTACGTGCGTA
read 3:             CGTACGCACGTACGTACGTACG
read 4:             TACGTACGTACGTGCGTACGTA
read 5:             CGCACGTACGTACGTACGTACG
read 6:             TACGTGCGTACGTACGTAC
```

## Reference based Compression

BAM files are too large

- ~1.5-2 bytes per base pair

Increases in disk capacity are being far outstripped by sequencing technologies  
BAM stores all of the data

- Every read base
- Every base quality
- Using a single conventional compression technique for all types of data

```
Reference sequence: ACGTACGTACGTACGTACGTACGTACGTACGTAC
read 1:             .....G.....
read 2:             .....C.....
read 3:             .....C.....
read 4:             .....G.....
read 5:             .....C.....
read 6:             .....G.....
```



# CRAM

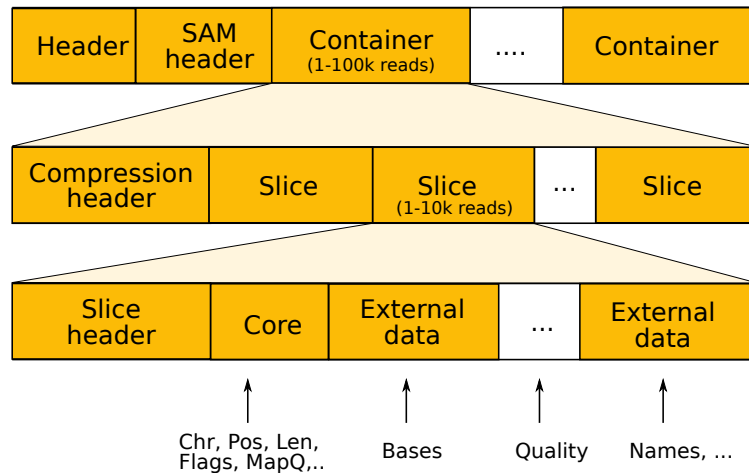
Three important concepts

- Reference based compression
- Controlled loss of quality information
- Different compression methods to suit the type of data, e.g. base qualities vs. metadata vs. extra tags

In lossless mode: 60% of BAM size

Archives and sequencing centers moving from BAM to CRAM

- Support for CRAM added to Samtools/HTSlib in 2014
- Soon to be available in Picard/GATK



# VCF: Variant Call Format

File format for storing variation data

- Tab-delimited text, parsable by standard UNIX commands
- Flexible and user-extensible
- Compressed with BGZF (bgzip), indexed with TBI or CSI (tabix)

```

##fileformat=VCFv4.0
##fileDate=20100707
##source=VCFtools
##reference=NCBI36
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality (phred score)">
##FORMAT=<ID=GL,Number=3,Type=Float,Description="Likelihoods for RR,RA,AA genotypes (R=ref,A=alt)">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##ALT=<ID=DEL,Description="Deletion">
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the variant">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT SAMPLE1 SAMPLE2
1 1 . ACG A,AT . PASS . GT:DP 1/2:13 0/0:29
1 2 rs1 C T,CT . PASS H2;AA=T GT:GQ 0|1:100 2/2:70
1 5 . A G . PASS . GT:GQ 1|0:77 1/1:95
1 100 T <DEL> . PASS SVTYPE=DEL;END=300 GT:GQ:DP 1/1:12:3 0/0:20
    
```

**Mandatory header lines** (indicated by a red arrow pointing to the first line)

**Optional header lines** (meta-data about the annotations in the VCF body) (indicated by a grey arrow pointing to the INFO and FORMAT lines)

**Reference alleles (GT=0)** (indicated by a blue arrow pointing to the first column of the body)

**Alternate alleles (GT>0 is an index to the ALT column)** (indicated by a blue arrow pointing to the second column of the body)

**Phased data** (G and C above are on the same chromosome) (indicated by a blue arrow pointing to the vertical bar in the FORMAT field)

**Deletion** (indicated by a blue arrow pointing to the ALT field '<DEL>')

**SNP** (indicated by a blue arrow pointing to the ALT field 'A,AT')

**Large SV** (indicated by a blue arrow pointing to the ALT field 'T,CT')

**Insertion** (indicated by a blue arrow pointing to the ALT field 'G')

**Other event** (indicated by a blue arrow pointing to the ALT field '<DEL>')

# VCF / BCF

VCFs can be very big

- compressed VCF with 3781 samples, human data:
  - 54 GB for chromosome 1
  - 680 GB whole genome

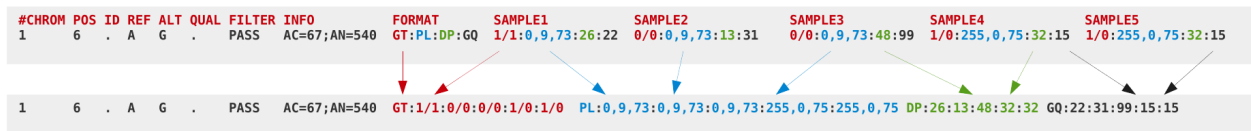
VCFs can be slow to parse

- text conversion is slow
- main bottleneck: FORMAT fields

```
##fileformat=VCFv4.0
##fileDate=20100707
##source=VCFtools
##ALT=<ID=DEL,Description="Deletion">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the variant">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT SAMPLE1 SAMPLE2
1 3 . A G . PASS AC=67;AN=5400;DP=2809 GT:PL:DP:GQ 1/1:0,9,73:26:22 0/0:0,9,73:13:31 0/0:0,9,73:48:99 1/0:255,0,75:32:15 1/0:255,0,75:32:15
1 4 . A T . PASS AC=15;AN=6800;DP=6056 GT:PL:DP:GQ 0/0:0,9,73:13:31 1/0:255,0,75:32:15 0/0:0,2,80:14:90 1/1:0,9,73:26:22 0/0:0,9,73:13:31
1 5 . C T . PASS AC=20;AN=6701;DP=5234 GT:PL:DP:GQ 1/0:255,0,75:32:15 0/0:0,2,170:14:90 1/1:0,9,73:13:31 0/0:0,6,50:13:80 0/0:0,2,80:14:90
1 6 . A G . PASS AC=67;AN=5400;DP=2809 GT:PL:DP:GQ 1/1:0,9,73:26:22 0/0:0,9,73:13:31 0/0:0,9,73:48:99 1/0:255,0,75:32:15 1/0:255,0,75:32:15
1 7 . A T . PASS AC=15;AN=6800;DP=6056 GT:PL:DP:GQ 0/0:0,9,73:13:31 1/0:255,0,75:32:15 0/0:0,2,80:14:90 1/1:0,9,73:26:22 0/0:0,9,73:13:31
```

## BCF

- binary representation of VCF
- fields rearranged for fast access



# gVCF

Often it is not enough not know *variant* sites only

- was a site dropped because of a reference call or because of missing data?
- we need evidence for both variant and non-variant positions in the genome

## gVCF

- blocks of reference-only sites can be represented in a single record using the INFO/END tag
- symbolic alleles <\*> for incremental calling
  - raw, "callable" gVCF
  - calculate genotype likelihoods only once (an expensive step)
  - then call incrementally as more samples come in

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	Sample
19	9902	.	G	<*>	.	.	END=9915;MinDP=0	PL:DP	0,0,0:0
19	9916	.	C	<*>	.	.	END=9922;MinDP=5	PL:DP	0,15,137:5
19	9923	.	G	<*>	.	.	END=9948;MinDP=10	PL:DP	0,30,214:10
19	9949	.	G	A,<*>	.	.	DP=28	PL:DP	0,60,255,78,255,255:27
19	9950	.	C	<*>	.	.	END=9958;MinDP=28	PL:DP	0,84,255:28
19	9959	.	G	T,<*>	.	.	DP=34	PL:DP	0,82,255,99,255,255:34
19	9960	.	C	<*>	.	.	END=9969;MinDP=34	PL:DP	0,102,255:34

↑ **Symbolic "unobserved" allele**  
 Represents any other possible alternate allele

↑ **A block of 10 sites** with at least 34 reference reads

↑ **Genotype likelihoods** for CC, C\*, \*\*



# Global Alliance for Genomics and Health

International coalition dedicated to improving human health

Mission

- establish a common framework to enable sharing of genomic and clinical data

Working groups

- clinical
- regulatory and ethics
- security
- data



Data working group

- beacon project .. test the willingness of international sites to share genetic data
- BRCA challenge .. advance understanding of the genetic basis of breast and other cancers
- matchmaker exchange .. locate data on rare phenotypes or genotypes
- reference variation .. describe how genomes differ so researchers can assemble and interpret them
- benchmarking .. develop variant calling benchmark toolkits for germline, cancer, and transcripts
- file formats .. CRAM, SAM/BAM, VCF/BCF

File formats

- <http://samtools.github.io/hts-specs/>

## Quality Control

Biases in sequencing

- Base calling accuracy
- Read cycle vs. base content
- GC vs. depth
- Indel ratio

Biases in mapping

Genotype checking

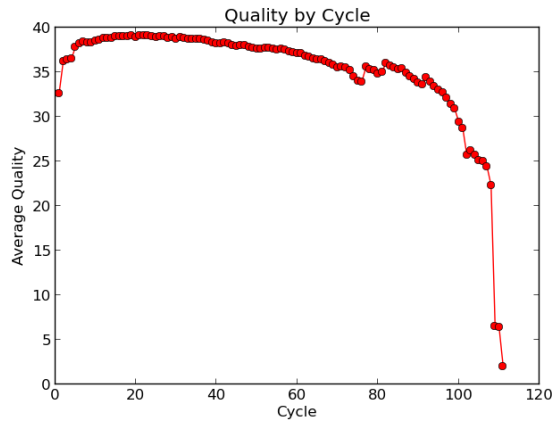
- Sample swaps
- Contaminations

# Base quality

Sequencing by synthesis: dephasing

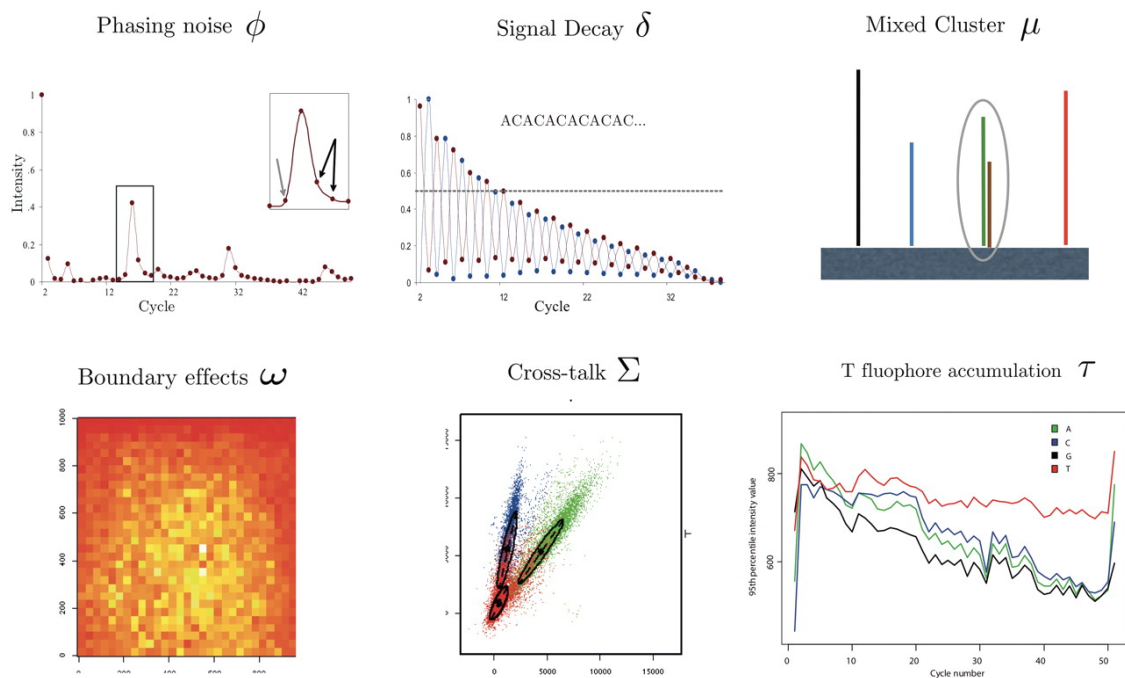
- growing sequences in a cluster gradually desynchronize
- error rate increases with read length

Calculate the average quality at each position across all reads

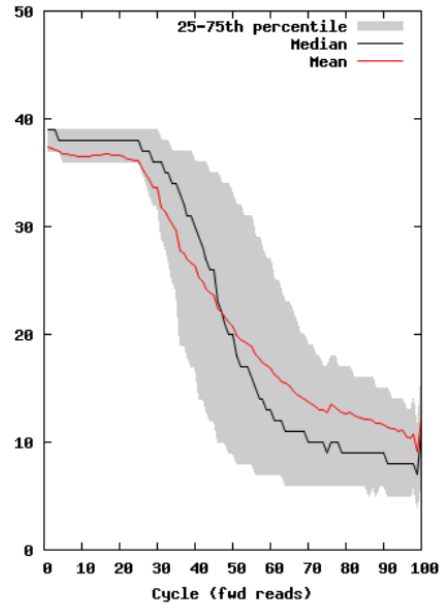
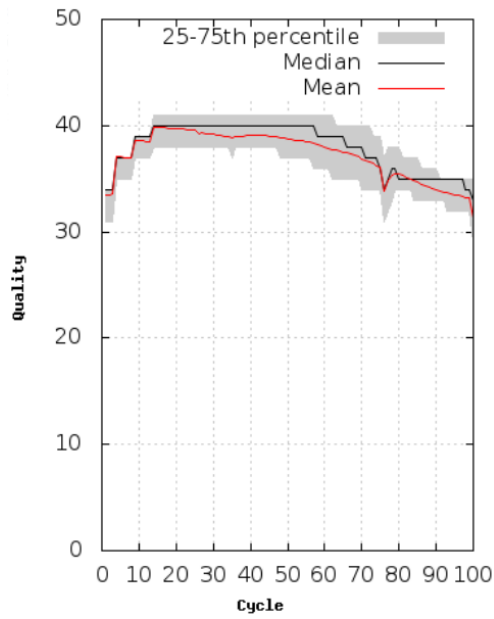


Quality	Probability of error	Call Accuracy
10 (Q10)	1 in 10	90%
20 (Q20)	1 in 100	99%
30 (Q30)	1 in 1000	99.9%
40 (Q40)	1 in 10000	99.99%

# Base calling errors



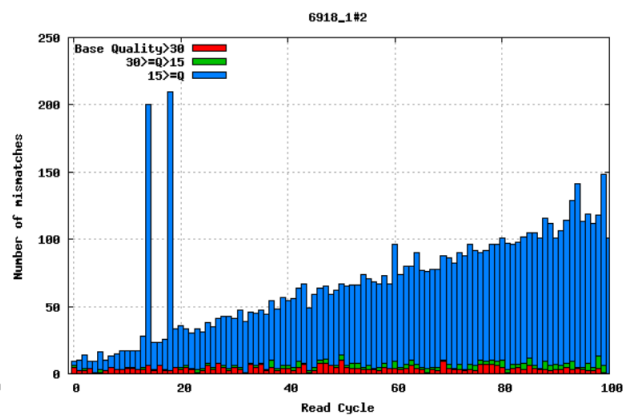
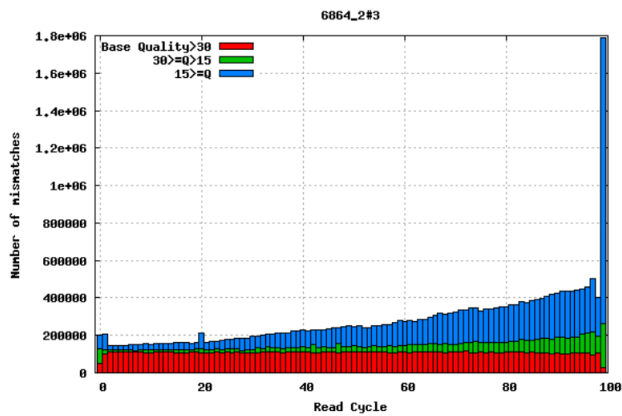
# Base quality



# Mismatches per cycle

Mismatches in aligned reads (requires reference sequence)

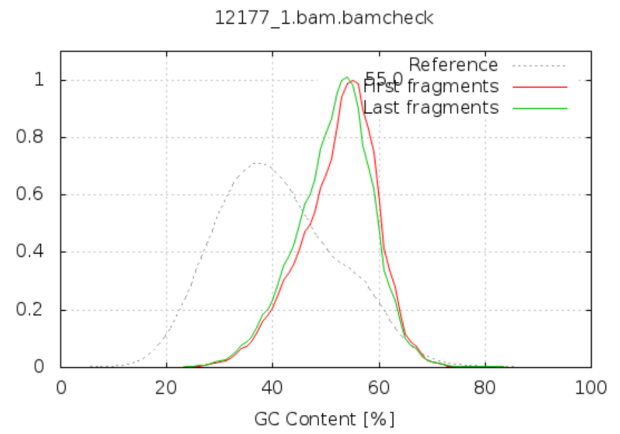
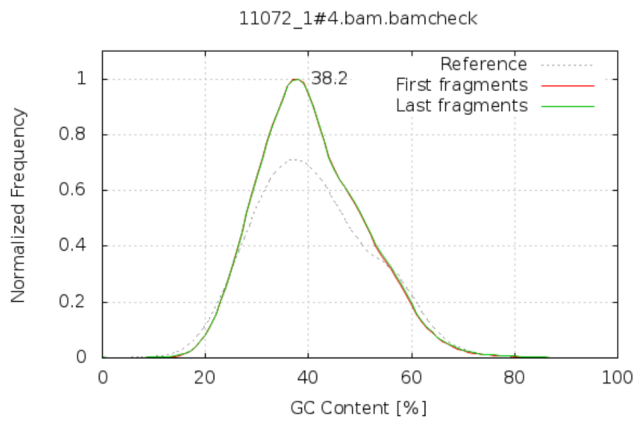
- detect cycle-specific errors
- base qualities are informative!



# GC bias

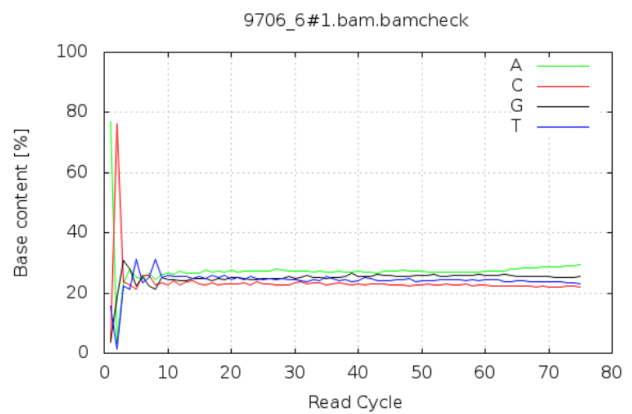
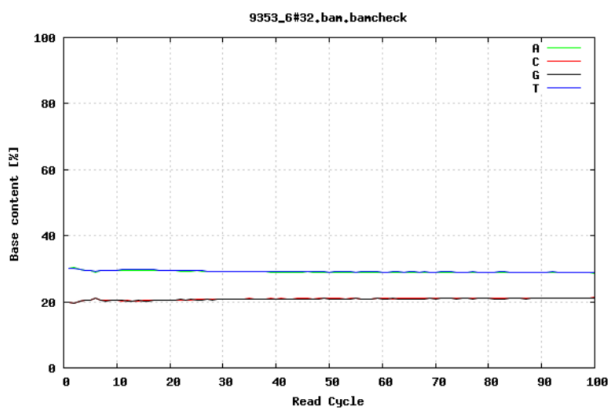
GC- and AT-rich regions are more difficult to amplify

- compare the GC content against the expected distribution (reference sequence)



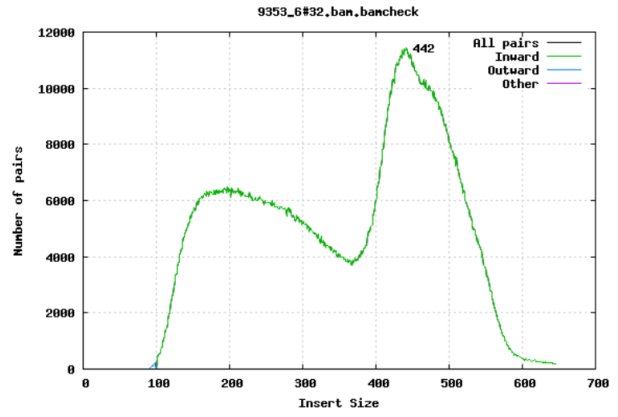
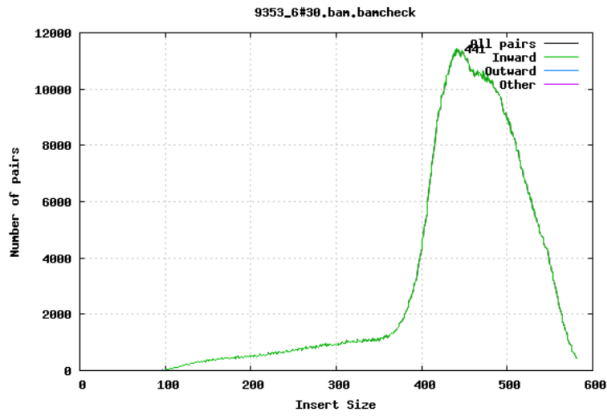
# GC content by cycle

Was the adapter sequence trimmed?

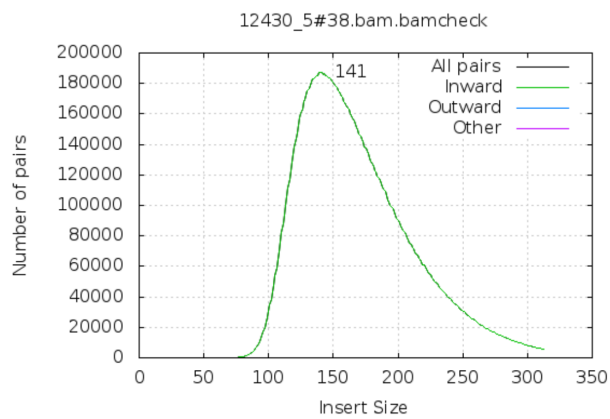


# Fragment size

Paired-end sequencing: the size of DNA fragments matters



# Quiz



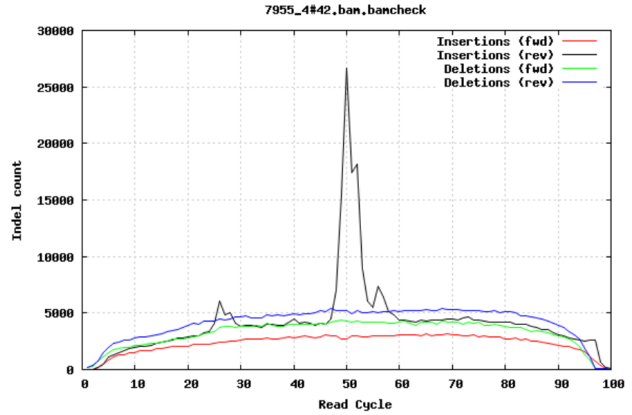
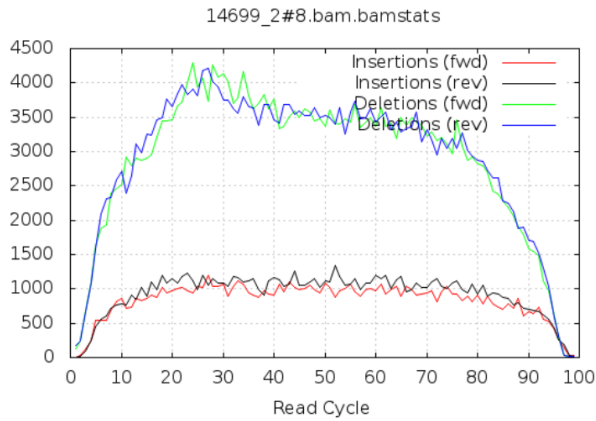
This is 100bp paired-end sequencing. Can you spot any problems??



# Insertions / Deletions per cycle

## False indels

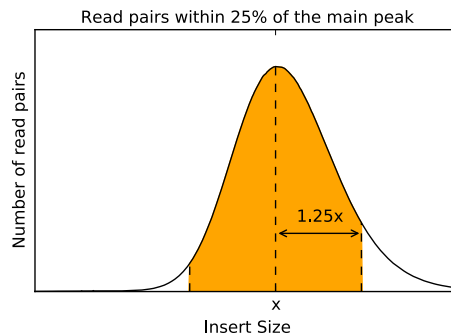
- air bubbles in the flow cell can manifest as false indels



# Auto QC tests

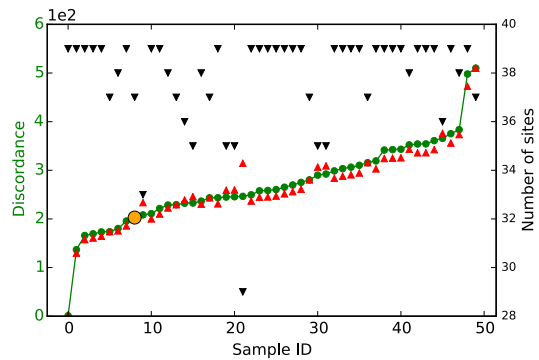
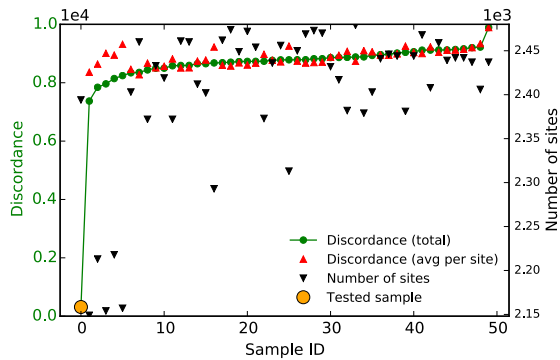
A suggestion for human data:

Minimum number of mapped bases	90%
Maximum error rate	0.02%
Maximum number of duplicate reads	5%
Minimum number of mapped reads which are properly paired	80%
Maximum number of duplicated bases due to overlapping read pairs	4%
Maximum in/del ratio	0.82
Minimum in/del ratio	0.68
Maximum indels per cycle, factor above median	8
Minimum number of reads within 25% of the main peak	80%



# Detecting sample swaps

Check the identity against a known set of variants



## Software

Software used to produce graphs in these slides

- `samtools stats` and `plot-bamstats`
- `bcftools gtcheck`
- `matplotlib`

## Practical exercises: File formats and QC

An online version of this document can be found here <https://tinyurl.com/ybyq3rk3>. Please feel free to add comments if anything is unclear or incorrect. The answers to the exercises can be found at the end of this document.

### Exercise 1: SAM header line

SAM/BAM format is the accepted standard format for storing NGS sequencing reads, base qualities, associated meta-data and alignments of the data to a reference genome. If no reference genome is available, the data can also be stored unaligned.

Download the SAM/BAM file specification document from <http://samtools.github.io/hts-specs> ([direct link](#)).

From reading page 4 of the SAM specification, look at the following line from the header of the BAM file:  
`@RG ID:ERR003612 PL:ILLUMINA LB:g1k-sc-NA20538-TOS-1 PI:2000 DS:SRP000540 SM:NA20538`

1.1 What does `RG` stand for?

1.2 What is the sequencing platform/technology used to produce the reads?

1.3 What is the lane ID?

(In sequencing terminology, a "lane" is the basic independent run of a high-throughput sequencing machine. Reads from one lane are identified by the same read group ID and the information about lanes can be found in the header in lines starting with `@RG`.)

1.4 What is the expected fragment insert size?

### Exercise 2: SAM header and samtools

Samtools comprises a set of programs for interacting with SAM/BAM files. Type `samtools` with no parameters to display the list of available commands implemented in the program. Then type `samtools view` to display a detailed usage page.

Now use the `samtools view` command to print the header of the BAM file:

```
samtools view -H NA20538.bam | less -S
```

The `-S` option makes `less` display long lines without wrapping them. (You can toggle between the wrapping and non-wrapping mode by pressing `-S` at any time.)

2.1 What version of the human assembly was used to perform the alignments? (Look for the genome assembly identifier `AS`.)

2.2 How many lanes are in this BAM file? Remember that each lane is identified by a unique read group ID. Use the commands `grep` to parse the BAM header, looking for lines starting with `@RG`, and `wc -l` to count them.

2.3 What programs were used to create this BAM file? Look up the meaning of the `@PG` lines.

## 2.4 What version of `bwa` was used to align the reads?

### Exercise 3: Alignment formats conversion

You can use `samtools` to convert between SAM<->BAM and to extract regions of a BAM file. On the command line type

```
samtools view NA20538.bam | less -S
```

As explained above, the `-S` switch causes that long lines are truncated rather than wrapped, which makes the output more readable. Alternatively, the UNIX command `cut` can be used to extract only the columns of interest. (For example, the command `cut -f1,4` prints only the first and the fourth columns of the input.)

**3.1** What is the name of the first read? Look up the QNAME field at page 5 of the SAM specification. Note that although the specification makes a distinction between a "query template" (the physical sequenced molecule) and a "read" (the actual sequence obtained by the experiment), both are often used interchangeably.

**3.2** What position does the alignment of the read start at?

**3.3** What is the mapping quality of the read?

We will convert a yeast BAM file to CRAM. In the data directory, there is a BAM file called `yeast.bam` that was created from *S. cerevisiae* Illumina sequencing data.

**3.4** Can you convert the BAM file to a CRAM file called `yeast.cram` using the `samtools view` command? First run the command without arguments to view the list of available options. For this exercise we will need `-C`, `-T` and `-o`. Note that the reference genome is stored in the file `Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa`. Name the output file `yeast.cram`.

Since CRAM files use reference based compression, we expect the CRAM file to be smaller than the BAM file. What is the size of the CRAM file?

**3.5** Is your CRAM file smaller than the original BAM file?

### Exercise 4: VCF/BCF and bcftools

VCF/BCF format is the accepted standard format for storing variant calls with supporting data. The official specification is available from <http://samtools.github.io/hts-specs>.

Bcftools comprises a set of programs for interacting with VCF/BCF files. You can use `bcftools` to convert between VCF<->BCF and to view or extract records from a region. Type `bcftools` without arguments to see the list of available commands. Then add name of any of the commands (for example, type `bcftools view`) to see the list of available options.

```
bcftools
bcftools view
```

Using the `bcftools view` command, print the header of the BCF file

```
bcftools view -h 1kg.bcf | less
```

and answer the following questions:

4.1 What version of the human assembly the coordinates refer to?

4.2 Can you convert the file called `1kg.bcf` to a compressed VCF file called `1kg.vcf.gz` using the `bcftools view` command? You will need the `--output-file` and `--output-type` options.

Similarly to BAM, the VCF/BCF format supports random access and can quickly retrieve records from any genomic region. For this, the file must be indexed.

4.3 Index the BCF, then use the `bcftools view` command to extract records from a region by adding the option `--regions 20:24042765-24043073`

Now we are able to extract complete records from any position or region. Can we extract individual fields as well? The versatile `bcftools query` command can be used to do that. Combined with standard UNIX commands, it gives a powerful tool for quick querying of VCFs. Try to answer the following questions with the help of the [manual page](#).

4.4 How many samples are in the BCF? (Hint: check the `-l` option.)

4.5 What is the genotype of the samples `HG00107` and `HG00108` at the position `20:24019472`? Use the `bcftools query` command with the following options:

```
--regions 20:24019472 to extract the VCF record at this position
--samples HG00107,HG00108 to extract the two samples
--format '%POS[ %GT]\n' to output the genotypes, printing first the position and then the
genotypes separated by a space (the square brackets loop over samples)
```

4.6 How many positions there are with more than 10 alternate alleles? First check the VCF specification and the VCF header (`bcftools view -h`). You will find that this information is encoded by the `INFO/AC` tag. Then extract all records with the `INFO/AC` value bigger than 10 using the `--include 'AC>10'` option and `wc -l` to count the lines.

4.7 List positions where the sample `HG00107` has a non-reference genotype and the read depth is bigger than 10. Similarly as above, use the `bcftools query` command with the following options:

```
--samples HG00107 to extract the sample
--include 'FORMAT/DP>10 & FORMAT/GT="alt"' to match positions with read depth
bigger than 10 and with genotype containing an alternate allele. The ampersand symbol &
requires that both conditions must be true in the same sample
--format '%POS[ %GT %DP]\n' to output position, the genotype, and the read depth.
```

Pipe the output into `head` to display only the first few lines.

## Exercise 5: Generate QC stats

We will generate QC stats for two lanes of Illumina paired-end sequencing data from yeast. We will use the `bwa` mapper to align the data to the *Saccharomyces cerevisiae* genome ([ftp://ftp.ensembl.org/pub/current\\_fasta/saccharomyces\\_cerevisiae/dna](ftp://ftp.ensembl.org/pub/current_fasta/saccharomyces_cerevisiae/dna)) and `samtools stats` to generate the stats.

5.1 Read pairs are usually stored in two separate FASTQ files so that  $n$ -th read in the first file and the  $n$ -th read in the second file constitute a read pair. Can you devise a quick sanity check that reads in these two files really form pairs? The files must have the same number of lines and the naming of the reads usually suggests if they form a pair. The location of the files is

```
60A_Sc_DBVPG6044/lane1/s_7_1.fastq
```

60A\_Sc\_DBVPG6044/lane1/s\_7\_2.fastq

First check whether the read names are suggestive of a pair. For example, the reads in the first file can have the suffix /1 and the reads in the second file should have the suffix /2. Then check whether there is the same number of reads in both files.

Run the `./align.sh` script to create the mappings. The script is very short, take a look inside using the command `less ./align.sh`. The script contains several commands, some are combined together using pipes. UNIX pipes is a very powerful and elegant concept which allows us to feed the output of one command into the next command and avoid writing intermediate files.

The script will produce the BAM file `lane1.sorted.bam`. Generate the stats including only primary alignments using the command

```
samtools stats -F SECONDARY lane1.sorted.bam > lane1.sorted.bam.bchk
```

Look at the output and answer the following questions:

**5.2** What is the total number of raw sequence reads?

**5.3** How many reads were mapped?

**5.4** How many read pairs were mapped to a different chromosome?

**5.5** What is the insert size mean and standard deviation?

Next we will create some QC plots from the output of the stats command using the command `plot-bamstats` which is of the samtools package:

```
plot-bamstats -p lane1-plots/ lane1.sorted.bam.bchk
```

In your web browser open the generated html file to view the graphs

```
firefox lane1-plots/*.html
```

**5.6** How many reads have zero mapping quality (MQ)?

**5.7** Check the "Quality per cycle" graph. Which of the first fragments or second fragments are higher base quality on average?

## Answers to exercises:

**1.1** Read Group

**1.2** Illumina, see the PL field

**1.3** ERR003612, see the ID field

**1.4** 2kbp, see the PI field

**2.1** NCBI build v37

**2.2** The command is

```
samtools view -H NA20538.bam | grep ^@RG | wc -l
```

**2.3** Use the command

```
samtools view -H NA20538.bam | grep ^@PG | less -S
```

Usually the alignments are processed with multiple programs. The @PG lines in this BAM file suggest that the reads were aligned using `bwa`, then `GATK` was used to recalibrate qualities and realign indels.

**2.4** Find VN field of the @PG line.

**3.1** The name of the first read is `ERR003814.1408899`, use for example a command like this

```
samtools view NA20538.bam | head -1 | cut -f1
```

**3.2** Chromosome 1, position 19999970

```
samtools view NA20538.bam | head -1 | cut -f3,4
```

**3.3** Q23

```
samtools view NA20538.bam | head -1 | cut -f5
```

**3.4** Use the command

```
samtools view -C -T Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa  
-o yeast.cram yeast.bam
```

**3.5** Use the command

```
ls -lh yeast.bam yeast.cram
```

**4.1** Lookup the `##reference` line

**4.2** Use the command

```
bcftools view -O z -o 1kg.vcf.gz 1kg.bcf
```

**4.3** Use the following commands:

```
bcftools index 1kg.bcf
```

```
bcftools view -H -r 20:24042765-24043073 1kg.bcf | less -S
```

**4.4** Use `bcftools query -l 1kg.bcf` to get list of samples and `wc -l` to count the lines

```
bcftools query -l 1kg.bcf | wc -l
```

**4.5** The complete command is

```
bcftools query -r 20:24019472 -s HG00107,HG00108 -f '%POS [ %GT]\n' 1kg.bcf
```

**4.6** The command can look like this:

```
bcftools query -i 'AC>10' -f '%POS\n' 1kg.bcf | wc -l
```

**4.7** The complete command is

```
bcftools query -s HG00107 -i 'FORMAT/DP>10 & FORMAT/GT="alt"' -f '%POS [ %GT %DP]\n' 1kg.bcf | head
```

**5.1** You can use the following commands

```
head 60A_Sc_DBVPG6044/lane1/s_7_1.fastq | grep ^@
```

```
head 60A_Sc_DBVPG6044/lane1/s_7_2.fastq | grep ^@
```

```
wc -l 60A_Sc_DBVPG6044/lane1/*.fastq
```

**5.2** Look inside the file and locate the field "raw total sequences". To extract the information quickly from multiple files, commands similar to the following can be used:

```
grep ^SN lane*.sorted.bam.bchk | awk -F'\t' '$2=="raw total sequences:"'
```

**5.3** Locate the field "reads mapped" or use the command

```
grep ^SN lane*.sorted.bam.bchk | awk -F'\t' '$2=="reads mapped:"'
```

**5.4** Locate the field "pairs on different chromosomes" or use the command

```
grep ^SN lane*.sorted.bam.bchk | awk -F'\t' '$2=="pairs on different  
chromosomes:"'
```

**5.5** Locate the "insert size" fields

## NGS read alignment

Thomas Keane (material by Vivek Iyer, Adams Group, WTSI)  
Team Leader, EMBL-EBI  
@drtkeane

*In the footsteps of Anthony Doran*



## NGS read alignment

*The most important first step in understanding next-generation sequencing data is the initial alignment or assembly that determines whether an experiment has succeeded and provides a first glimpse into the results.*

Flicek & Birney, 2009. *Nature Methods*

Sequence alignment in NGS is:

- *Process of determining the most likely source of the observed DNA sequencing read within the reference genome sequence.*

Principles and approaches to sequence alignment have not changed much since 80's



## Overview

**Intro**

**Methods / Aligners**

**Alignment Outputs**

**Alignment Viewers**

**NGS Workflows, QC and BAM Improvement**



## The reference is key in all that follows

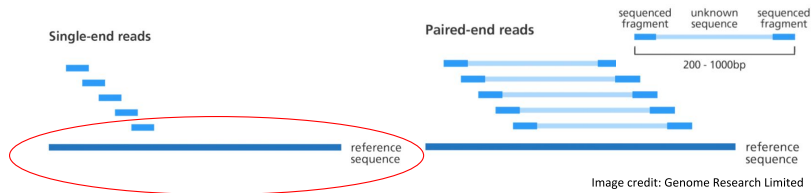
Sequence alignment in NGS is:

- *Process of determining the most likely source of the observed DNA sequencing read within the **reference genome sequence***





## The reference is key in all that follows



The view of alignment in *this* lecture is *wildly* asymmetric:

- short read sequences
- very long reference sequences. Genomes (~Gbp) or Transcriptomes (~100Mbp)

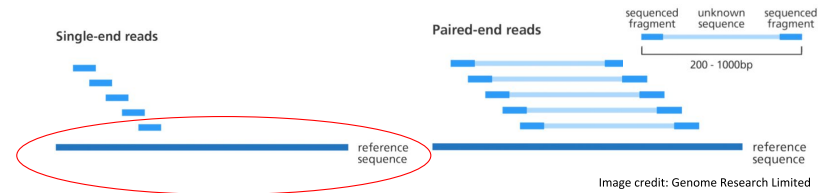
Where do references come from?

Organism => Consortium => Money => Sequencing => Assembly => Curation => Dissemination => Curation, Dissemination => ...

*Human, Mouse, Zebrafish, Chicken:*

Lots of curation / dissemination was done at various centres and distributed by NCBI. NCBI still distributes!  
Now the umbrella for curation / patching is: **Genome Reference Consortium**

## The reference is key in all that follows



### HUMAN reference sequences

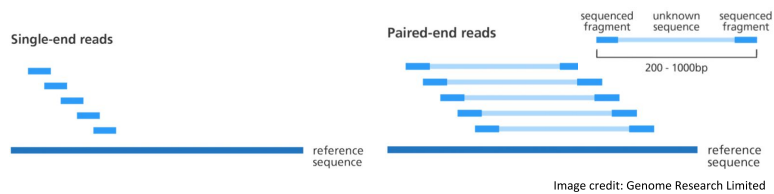
Release name	Date of release	Equivalent UCSC version
GRCh38	Dec 2013	hg38
GRCh37	Feb 2009	hg19
NCBI Build 36.1	Mar 2006	hg18
NCBI Build 35	May 2004	hg17
NCBI Build 34	Jul 2003	hg16

### MOUSE reference sequences

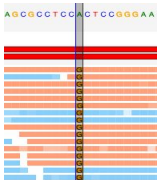
Release name	Date of release	Equivalent UCSC version
GRCm38	Dec 2011	mm10
NCBI Build 37	Jul 2007	mm9
NCBI Build 36	Feb 2006	mm8
NCBI Build 35	Aug 2005	mm7
NCBI Build 34	Mar 2005	mm6

The actual reference is a just a (big) sequence (fasta) file: `$ ls -h GRCh38.fa`  
> 3.1G GRCh38.fa

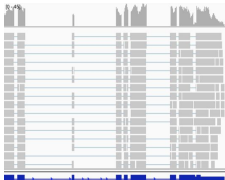
## Why align?



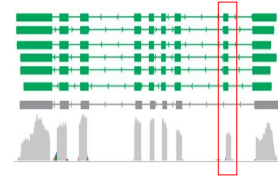
### Identify variation



### Transcript abundance



### Discover transcribed sequence



## Overview

Intro

Methods / Aligners

Alignment Outputs

Alignment Viewers

NGS Workflows, QC and BAM Improvement

## Local alignment: Smith-Waterman algorithm (1981)

Algorithm for generating the optimal pairwise alignment between two sequences

*Optimal alignment:* the alignment which exhibits the most correspondences and the least differences, i.e. the alignment with the highest mapping score

Time consuming to carry out for every read - impractical

- Aligner will use it to refine a quick approximate placement
- Variant caller might use it to correctly re-align reads with insertions/deletions

We will now get a feel for this

## Local alignment: Smith-Waterman algorithm (1981)

We get a feel for this by aligning things by eye, then doing it using S-W:

QUERY SEQUENCES:

ATG

REFERENCE SEQUENCE:

ATCG

ALIGNMENT:

AT-G

|||

ATCG

QUERY SEQUENCES:

ACCG

REFERENCE SEQUENCE:

ATCG

ALIGNMENT:

ACTG

||||

ATCG

## Local alignment: Smith-Waterman algorithm (1981)

We get a feel for this by aligning things by eye, then doing it using S-W:

QUERY SEQUENCES:

ATG

REFERENCE SEQUENCE:

ATCG

ALIGNMENT:

A-TG

|||

ATCG

QUERY SEQUENCES:

ACCG

REFERENCE SEQUENCE:

ATCG

ALIGNMENT:

ACTG

||||

ATCG

	A	C	T	G
A				
T				
G				

	A	C	T	G
A				
T				
C				
G				

## Local alignment: Smith-Waterman algorithm (1981)

Algorithm for generating the optimal pairwise alignment between two sequences:

Score every cell in matrix from top to bottom.  
Scoring is "cumulative" from previous cell:

Each cell has three possible scores:

Entry from diagonal (match or mismatch)

Entry from left (gap open)

Entry from above (gap open)

Accumulate this score to previous cell, based on

match/mismatch/gap

Match: +1

Mismatch: -1

Gap Open: -1

- 3 possible scores based on 3 previous cells

Put "top" cumulative score in current cell.

**If top score is negative, replace with 0**

Record the *source* of top score (which cell you entered from )

		A	C	T	G
	0	0	0	0	0
A	0	-1,-1,1 = 1	0,-1,-1 = 0	-1,-1,-1 = -1 = 0	-2,-1,-1 = -1 = 0
C	0	-1,0,-1 = 0	-1,-1,2 = 2	1,-1,-1 = 1	0,-1,-1 = 0
G	0	-1,-1,-1 = -1 = 0	-1,1,-1 = 1	0,0,1 = 1	-1,-1,2 = 2

## Local alignment: Smith-Waterman algorithm (1981)

Algorithm for generating the optimal pairwise alignment between two sequences:

TRACEBACK:

Start at highest score (2) and trace back to source until you get to 0 score.

WRITE OUT ALIGNMENT:

S1 A C T G  
S2 A C - G

At each point of S1/S2: either a character or "-"  
Entry from sides = gap  
Entry from diagonal = match or mismatch

		A	C	T	G
	0	0	0	0	0
A	0	-1,-1,1 = 1	0,-1,-1 = 0	-1,-1,-1 = -1 = 0	-2,-1,-1 = -1 = 0
C	0	-1,0,-1 = 0	-1,-1,2 = 2	1,-2,-1 = 1	0,-1,-2 = 0
G	0	-1,-1,-1 = -1	-2,1,-1 = 1	0,0,1 = 1	-1,-1,2 = 2



## Local alignment: Smith-Waterman algorithm (1981)

Algorithm for generating the optimal pairwise alignment between two sequences

*Optimal alignment*: the alignment which exhibits the most correspondences and the least differences, i.e. the alignment with the highest mapping score

Time consuming to carry out for every read

- Only applied to a small subset of the reads that don't have an exact match
- Important for correctly aligning reads with insertions/deletions

We will now get a feel for this (done!)

Further concepts:

**Local alignment**: Any segment of one sequence can align to an arbitrary position in the other sequence. **Segments** of query align to **segments** of target.

**Global alignment**: Alignment of two complete sequences. *If sequences are very dissimilar in size then there will be lots of gaps.*

## NGS read alignment

**NGS: Nucleotide based alignment (also known as read mapping)**

Number of 150bp reads in an 40x coverage Illumina X10 genome sequence?  
**800x10<sup>6</sup>**

These all have to be aligned against a mammalian genome (3Gbp)

**Two primary approaches:**

- Hash-based alignment
- Suffix/prefix trees



## Hash table alignment

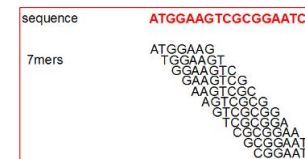
Note: K-mer is a short fixed sequence of nucleotides

First thing: "wrap up" the reference genome to bring it to the reads – build a kmer hash

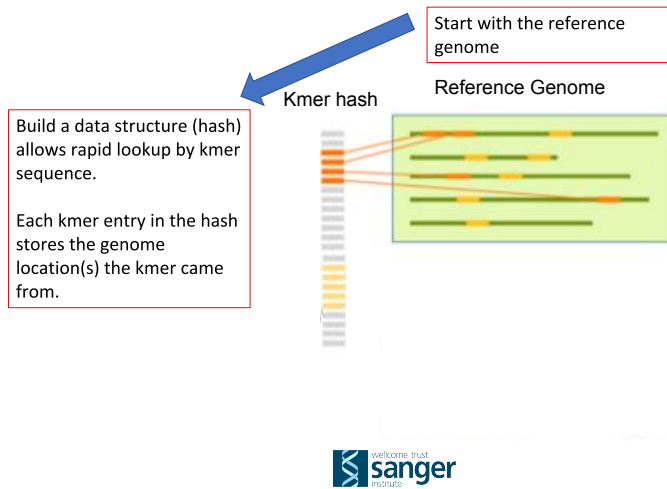
Scan the reference genome:

Build a profile (index) of all possible k-mers of length  $n$  and the locations in the reference genome they occur:

Hash table will be several Gbytes in size for human genome



## Hash table alignment



## Hash table alignment

Note: K-mer is a short fixed sequence of nucleotides

**First thing: "wrap up" the reference genome to bring it to the reads – build a kmer hash**

Scan the reference genome:

Build a profile (index) of all possible k-mers of length  $n$

and the locations in the reference genome they occur:

Hash table will be several Gbytes in size for human genome

**Next: For each sequence read:**

Split the read into k-mers of length  $n$

Lookup the locations in the reference via the index hash table (you made above)

Your hash table stores genomic locations => Pick region on the genome with most k-mer hits

Perform **Smith-Waterman** alignment to fully align the read to the region

Output the alignment of each read onto the reference in BAM (or equivalent) format.

**Hash of the reads:** MAQ, ELAND, ZOOM and SHRIMP

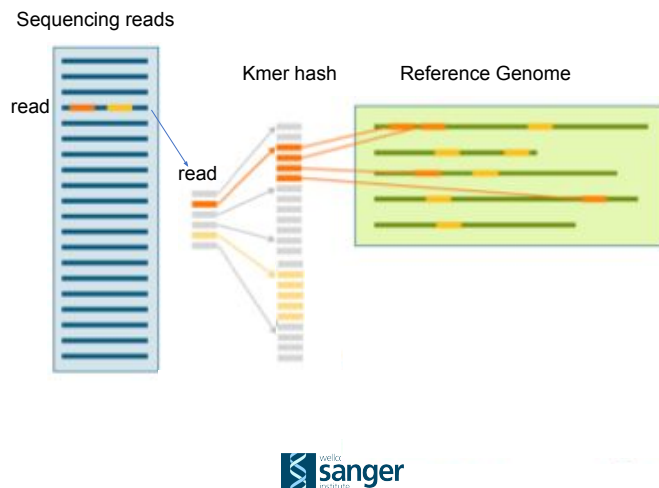
Smaller but more variable memory requirements

**Hash the reference:** SOAP, BFAST and MOSAIK

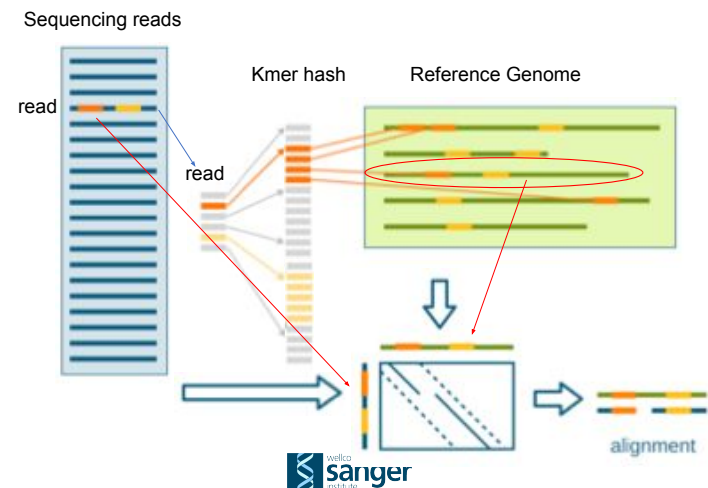
Advantage: constant memory cost



## Hash table alignment



## Hash table alignment



## Hash table alignment

Note: K-mer is a short fixed sequence of nucleotides

**First thing:** "wrap up" the reference genome to bring it to the reads – build a kmer hash

Scan the reference genome:

Build a profile (index) of all possible k-mers of length  $n$

and the locations in the reference genome they occur:

Hash table will be several Gbytes in size for human genome

**Next: For each sequence read:**

Split the read into k-mers of length  $n$

Lookup the locations in the reference via the index (**seed phase**)

Pick region on the genome with most k-mer hits

Perform **Smith-Waterman** alignment to fully align the read to the region

Output the alignment of each read onto the reference in BAM (or equivalent) format.

**Hash of the reads:** MAQ, ELAND, ZOOM and SHRIMP

Smaller but more variable memory requirements

**Hash the reference:** SOAP, BFAST and MOSAIK

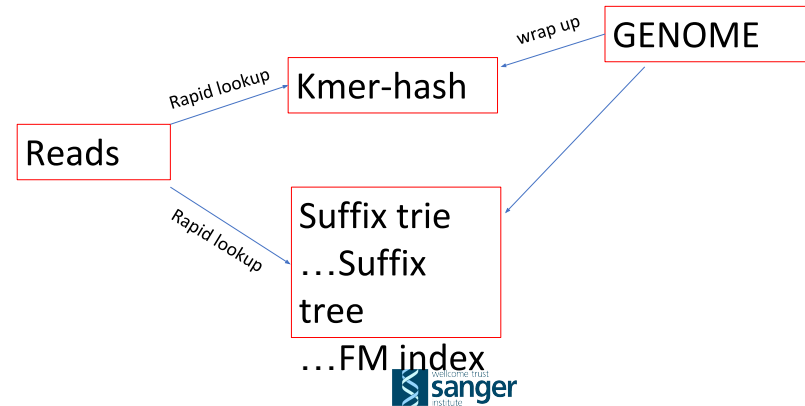
Advantage: constant memory cost



## Wrapping up the genome and bringing it to your reads

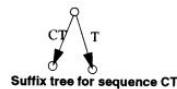
Kmer hash:

Summarises (folds up) content of genome and brings it close to the reads



## Suffix/Prefix tree based aligners

For fast string matching: A suffix trie, or simply a trie, is a data structure that stores all the suffixes of a string, enabling fast string matching. To establish the link between a trie and an FM-index, a data structure based on Burrows-Wheeler Transform (BWT)



Build a suffix-trie for the genome:

- a different structure to bring the genome to the query

- But the memory requirements are huge

More terms: BW transform, FM-index ...

- Methods of reducing memory & time footprint

Examples: **BWA**, bowtie2, MUMmer

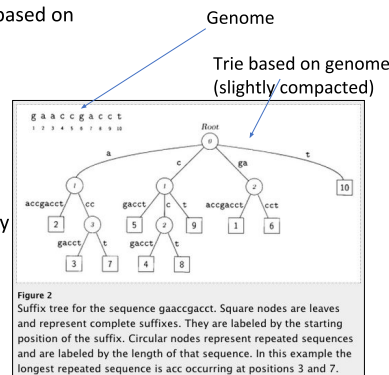


Figure 2  
Suffix tree for the sequence gaaccgacct. Square nodes are leaves and represent complete suffixes. They are labeled by the starting position of the suffix. Circular nodes represent repeated sequences and are labeled by the length of that sequence. In this example the longest repeated sequence is acc occurring at positions 3 and 7.

Delcher et al (1999) NAR



## Mapping qualities

**What if there are several possible places in the genome to align your sequencing read?**

Genomes contain many different types of repeated sequences

- Transposable elements (40-50% of vertebrate genomes)
- Low complexity sequence
- Reference errors and gaps
- Plenty of closely paralogous genes!

**The aligner will issue a MAPPING QUALITY to each alignment**

Mapping quality is a measure of how confident the aligner is that the read corresponds to this location in the reference genome

Typically represented as a phred score (log scale)

Q0 = read placed with identical score elsewhere

Q10 = 1 in 10 incorrect

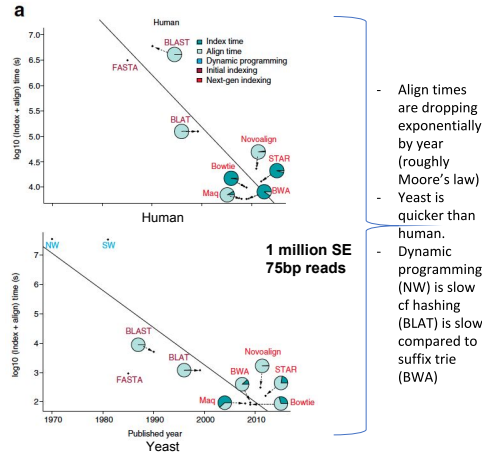
Q20 = 1 in 100 incorrect

Paired-end sequencing is useful - if one end maps inside a repetitive elements and one outside in unique sequence: **the aligner will use this**

- Hence prefer paired-end sequencing



## Aligner choice – some considerations



Muir et al. Genome Biology (2016)  
DOI 10.1186/s13059-016-0917-0

- Align times are dropping exponentially by year (roughly Moore's law)
- Yeast is quicker than human.
- Dynamic programming (NW) is slow compared to suffix trie (BWA)

## Aligner choice – some considerations

In the event that you have a choice of aligner to use

### Read manual, publication, vignette, etc

Sometimes, defaults may not be appropriate for your study

Does an aligner/mapper support:

- my platform (Illumina, PacBio etc)?
- my sequence type (DNA, RNA, etc)?
- paired end reads?
- my read lengths (e.g. PACBio are long 10's of thousands)
- Consider speed, and memory requirements and your resources
- Does it allow for mismatches (SNPs)? (*Configurable*)
- Support for gapped alignments? (*Yes*)
- How does it handle multi-mapping reads? (*Configurable*)
- Does it allow spliced alignment (RNA seq)? (*Purpose build aligners for this*)



Muir et al. Genome Biology (2016)  
DOI 10.1186/s13059-016-0917-0

- Align times are dropping exponentially by year (roughly Moore's law)
- Yeast is quicker than human.
- Dynamic programming (NW) is slow compared to suffix trie (BWA)

## Some alignment options (there are lots more)

careful

Unspliced:  
No  
intron-sized  
gaps

Spliced:  
Allow for  
big gaps  
inside  
reads, are  
aware of  
gene-structure

Aligner	Gapped alignment	SE and/or PE	Input format	Output format	Trimming?	BS-s eq	Note
BWA-MEM	Yes	SE, PE	FASTQ FASTA	SAM	No	No	Seed + extend: Local alignment
<del>Bowtie</del>	<del>No</del>	<del>SE, PE</del>	<del>FASTQ FASTA</del>	<del>SAM</del>	<del>Yes</del>	<del>No</del>	<del>Mismatches &lt; 2</del> Don't use old tech
Bowtie2	Yes	SE, PE	FASTQ FASTA qseq	SAM	Yes	No	Seed + extend: Local alignment
TopHat	Yes	SE, PE	FASTQ FASTA qseq	SAM	Yes	No	Uses bowtie or bowtie2 as base aligner
STAR	Yes	SE, PE	FASTQ FASTA	SAM	Yes	No	Fastest and most accurate for RNAseq Independently performs spliced aligns (no preprocessing)

Spliced aligns (RNASeq)

## Alignments – scaling up

Question: IF: 50Mbp bwa mem = 5 CPU minutes.

AND: 150 Gbp per HiSeqX10 lane – 1 human genome sequenced at 45x

THEN: How long will it take to align your X10 human genome sequencing?



## Alignments – scaling up

Question: IF: 50Mbps bwa mem = 5 CPU minutes.  
AND: 150 Gbp per HiSeqX10 lane – 1 human genome sequenced at 45x  
THEN: How long will it take to align your X10 human genome sequencing?

How many lots of 50Mb in 150Gb ?  
 $150\text{Gb} / 50\text{Mb} = (150 \times 10^9) / (50 \times 10^6) = 3 \times 10^3$  lots

Each lot takes 5 minutes => total time is  $5 \times 3 \times 10^3$  CPU minutes  
15000 CPU minutes = 250 hours ~ 10 days **ouch**.



## Alignments – scaling up

Question: IF: 50Mbps bwa mem = 5 CPU minutes.  
AND: 150 Gbp per HiSeqX10 lane – 1 human genome sequenced at 45x  
THEN: How long will it take to align your human genome sequencing?

How many lots of 50Mb in 100Gb ?  
 $150\text{Gb} / 50\text{Mb} = (150 \times 10^9) / (50 \times 10^6) = 3 \times 10^3$

Each lot takes 5 minutes => total time is  $5 \times 3 \times 10^3$  CPU minutes  
15000 CPU minutes = 250 hours ~ 10 days **ouch**.

Compute Cluster with N nodes: Time of problem is reduced by N.

- E.g. if you have dedicated 200 nodes, the time for alignment drops to ~ 1.25 hrs
- Which is ok for a single sample.
  - If you have 1000 samples, you need more nodes + better optimisation.
  - At WTSI, alignment is more much more efficient (dedicated resources)



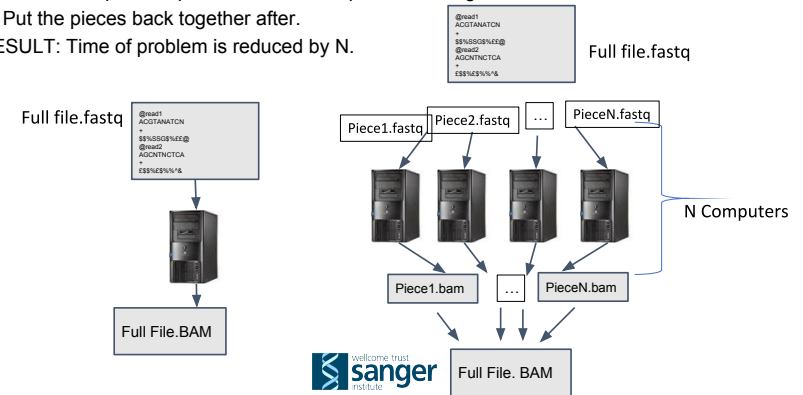
## Alignments – scaling up

This is why we like compute clusters!

**AS ALWAYS:** The strategy is to:

- Chop up the problem into small pieces and
- Solve each piece in parallel, with N computers working **at same time**
- Put the pieces back together after.

RESULT: Time of problem is reduced by N.



## Overview

Intro

Methods / Aligners

Alignment Outputs

Alignment Viewers

NGS Workflows, QC and BAM Improvement



## Pre-alignment file formats

Data generation



FASTQ

```
@LL16_4408:3:5:17860:13258
CTGGCTCACAATACAGGCCAGTATAAAGGCTCTCCTTAA
+
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
@LL16_4408:3:14:13276:1210
GTAGAAACAATATAAGAGCCTTAGGACAGAAACCCCTAA
+
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
@LL16_4408:3:83:5210:21157
AGGCAGCAGGAACCTCCAGTTAGGCCATAGTTCATCTTGC
```

Read ID  
Read sequence  
Read ID  
Qualities

The same 4 line format for paired reads (PE sequencing)

```
@LL16_4408:3:5:17860:13258
CAGTTTTTCAGAGCAGTAGCCATTAGGCACAATGTGATT
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
@LL16_4408:3:14:13276:1210
AGTCAACAGATGCTCTTAGGCTTAAGAATTCAGCAGAAG
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
@LL16_4408:3:83:5210:21157
GGCCAGTGTGCCCTCTGCCACTGAAGACATGTGATT
```



## Post-alignment file formats

**SAM (Sequence Alignment/Map) format**

- Single unified format for storing read alignments to a reference genome
- Developed by the 1000 Genomes group in 2009

**BAM (Binary Alignment/Map) format**

- Binary equivalent of SAM
- Developed for fast processing/indexing
  - Block GZIP compression for random access of regions

**Key features**

- Can store alignments from most aligners
- Supports multiple sequencing technologies
- Supports indexing for quick retrieval/viewing
- Compact size (e.g. 112Gbp Illumina = 116Gbytes disk space)
- Reads can be grouped into logical groups e.g. lanes, libraries, samples
- Widely support by variant calling software packages

Specification maintained by the Global Alliance for Genomics and Health



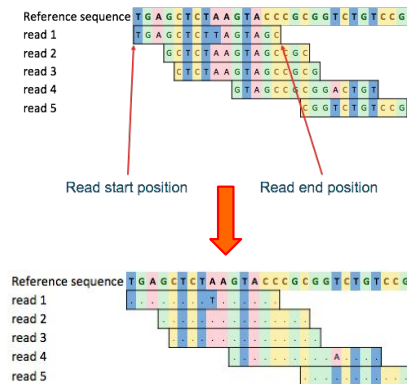
## Reference based compression

**BAM stores all of the data**

- Every read base
- Every base quality
- Using a single conventional compression technique for all types of data

**CRAM: Important concepts**

- Reference based compression
- Controlled loss of quality information
- Different compression methods to suit the type of data, e.g. base qualities vs. meta-data vs. extra tags

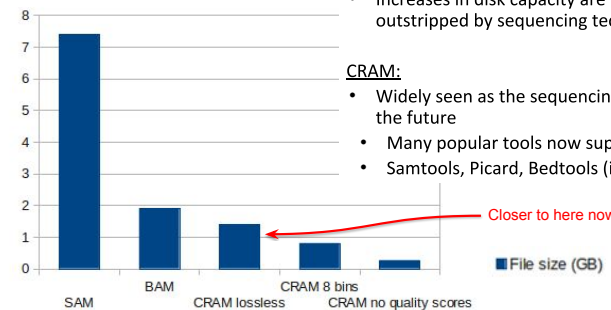


## CRAM format

- BAM files are too large
- ~1.5-2 bytes per base pair
- Increases in disk capacity are being far outstripped by sequencing technologies

**CRAM:**

- Widely seen as the sequencing format of the future
- Many popular tools now support CRAM
- Samtools, Picard, Bedtools (in the future)





## SAM/BAM format

### SAM fields

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	[!~?A~]{1,254}	Query template NAME
2	FLAG	Int	[0,2 <sup>16</sup> -1]	bitwise FLAG
3	RNAME	String	\*  [!~?A~]{1,~}	Reference sequence NAME
4	POS	Int	[0,2 <sup>31</sup> -1]	1-based leftmost mapping POSITION
5	MAPQ	Int	[0,2 <sup>8</sup> -1]	MAPping Quality
6	CIGAR	String	\*  ([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	\*  [!~?A~]{1,~}	Ref. name of the mate/next read
8	PNEXT	Int	[0,2 <sup>31</sup> -1]	Position of the mate/next read
9	TLEN	Int	[-2 <sup>31</sup> +1,2 <sup>31</sup> -1]	observed Template LENgth
10	SEQ	String	\*  [A-Za-z=.]+	segment SEQUENCE
11	QUAL	String	[!~?A~]{1,~}	ASCII of Phred-scaled base QUALity+33

### CIGAR string

Cigar has been traditionally used as a compact way to represent a sequence alignment: to go "FROM one string TO another you perform these operations".

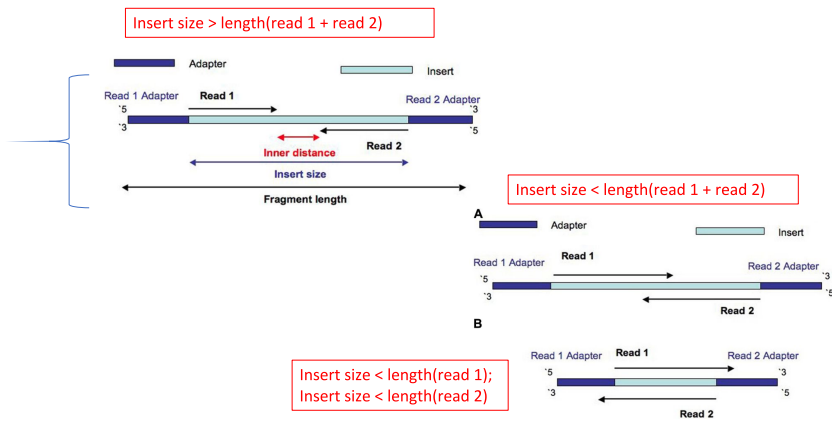
### Operations include

M - match or mismatch  
I - insertion  
D - deletion  
S - soft clip (ignore these bases)  
H - hard clip (ignore and remove these bases)

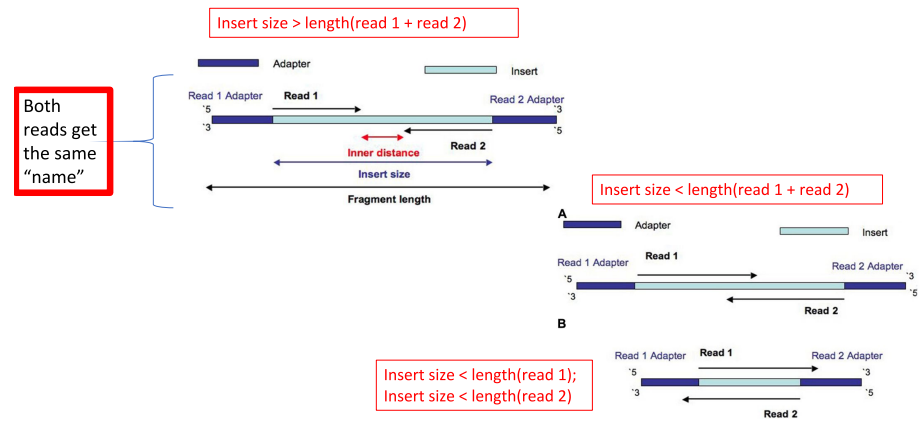
E.g. Read: **ACGCA-TGCAGTtagacgt**  
Ref: **ACTCAGTG--GT**  
Cigar: **5M 1D 2M 2I 2M 7S**

## INTERLUDE – PAIRED END SEQUENCING PARAMETERS

### Fragment length, insert size and inner mate distance



### Fragment length, insert size and inner mate distance



## NOW WE RESUME OUR NORMAL PROGRAMMING



## SAM/BAM/CRAM format in practice

```
HX8_24050:5:2103:20770:9449 83 7 4189207 0 151M = 4188954 -404 GTTG <JJJ MC:Z:151M MD:Z:151 NM:i:0 AS:i:151 XS:i:151
...
HX8_24050:5:2103:20770:9449 163 7 4188954 0 151M = 4189207 404 TTGG AAFF MC:Z:151M MD:Z:151 NM:i:0 AS:i:151 XS:i:151
...
HX8_24050:3:2212:32319:38772 99 7 4188947 0 151M = 4189300 436 ATGG AAFF MC:Z:68S83M MD:Z:151 NM:i:0 AS:i:151
XS:i:151
...
HX8_24050:3:2212:32319:38772 147 7 4189300 0 68S83M = 4188947 -436 ACAG --7- MC:Z:151M MD:Z:5C10T4T61 NM:i:3 AS:i:68
```

QNAME: read id  
- matches fastq

FLAG:  
> samtools flags 83: PAIRED, PROPER\_PAIR, REVERSE, READ1  
> samtools flags 163: PAIRED, PROPER\_PAIR, MREVERSE, READ2

RNAME, POS – chromosome & base pair position

MAPQ mapping quality  
CIGAR M? S? I? D?

RNEXT, PNEXT - paired read chr & position  
RNEXT “=” means same as first read  
PNEXT: read2 aligns 5’ of read1.  
TLEN: insert size

SEQUENCE  
QUALity (alignment doesn’t use them, but variant callers will)

ALIGNMENT TAGS:  
<http://www.samformat.info/sam-format-alignment-tags>

MC:mate cigar  
MD:string for mismatching positions  
NM>Edit distance to the reference (no clipping)  
AS:Aligner’s Alignment score  
XS:suboptimal alignment score (aha!!)

## SAM/BAM/CRAM format in practice

```
HX8_24050:5:1217:22455 83 7 87483848 60 118M1I32M = 87483669 -329 TG.. FJ.. MC:Z:151M MD:Z:150 NM:i:1 AS:i:143 XS:i:0
```

QNAME:  
FLAG  
RNAME  
POS  
MAPQ  
CIGAR  
RNEXT, PNEXT  
RNEXT  
PNEXT  
TLEN: can we work this out from the POS and PNEXT?  
SEQ:  
QUAL:  
MC: how does the mate match?  
MD: Does the mate have a mismatch?  
NM: This read: what is the edit distance?  
AS: what the current alignment score?  
XS: what is the align score of the next-best-hit?

## Overview

Intro

Methods / Aligners

Alignment Outputs

Alignment Viewers

NGS Workflows, QC and BAM Improvement



## Three viewers, in order of sophistication

### Context:

You have a mouse.  
 You extracted DNA from mouse kidney and performed paired-end Whole-Genome Sequencing.  
 Core sequencing facility (and/or an informatician and / or you) produces the alignments and summary stats.

They do variant calling! You find some variants you like, or some differentially expressed genes.  
**YOU HAVE TO LOOK AT YOUR ALIGNMENTS.**

### You will need:

Genome file (ie the reference)  
 Genome file: GRCm38.fa. Remember, there's always a reference back there ...  
 Gene annotations.  
 Alignment file: **MySample.bam**

### Moral

... You need to have a look at it. You cannot just trust a spreadsheet of variants given to you by someone else.



## Viewer 1: samtools

You could just use samtools **view** at the command line.

- This shows you the bam header: chromosomes/contigs, read-groups, programs which have acted
- This shows you each read id and CIGAR string, as well as mate structure (as we saw before)

```
$ samtools view -H MySample.bam
```

```
@HD VN:1.5 SO:coordinate
@SQ SN:1 LN:195471971 AS:GRCm38
@SQ SN:10 LN:130694993 AS:GRCm38
...
@SQ SN:X LN:171031299 AS:GRCm38
@SQ SN:Y LN:91744698 AS:GRCm38
...
@RG ID:332872 PL:ILLUMINA SM:MD5638a DS:WGS_ILLUMINA_short PU:24050_1
@RG ID:332873 PL:ILLUMINA SM:MD5638a DS:WGS_ILLUMINA_short PU:24050_2
...
@PG ID:basecalling_0 PN:Unknown PP:SCS_0 DS:Basecalling Package VN:Unknown
@PG ID:bambi_0 PN:bambi CL:/software/solexa/pkg/bambi/0.9.11/bin/bambi i2b ....
@PG ID:bamadapterfind_0 ...
@PG ID:bwa_0 PN:bwa CL:/software/solexa/pkg/bwa/ ...
```

```
HX8_24050:4:2105:31456:16147 99 1 3000000 40 20M11104M26S = 3000301 45 ...
```



## Viewer 1: samtools

You could just use samtools at the command line.

- This shows you the bam header.
- This shows you each read id and CIGAR string, as well as mate structure.

- But otherwise it's a pretty poor way of visualizing anything.

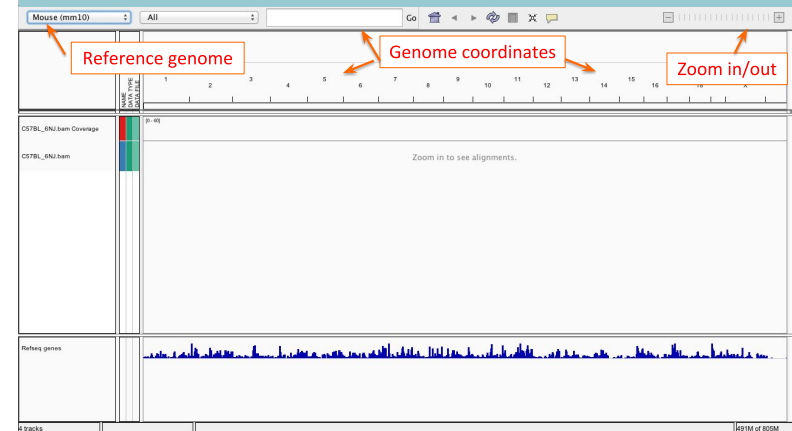
- No feel for relationship of pairs or how multiple reads show the same signal

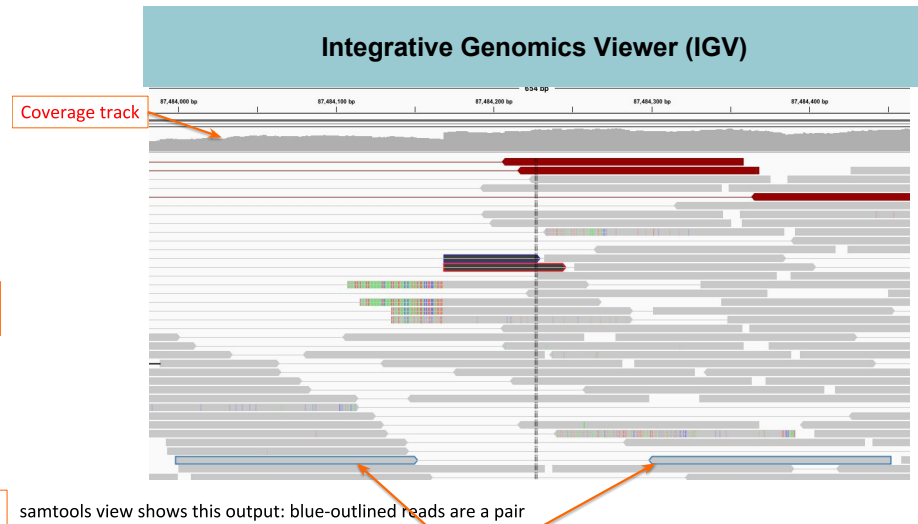
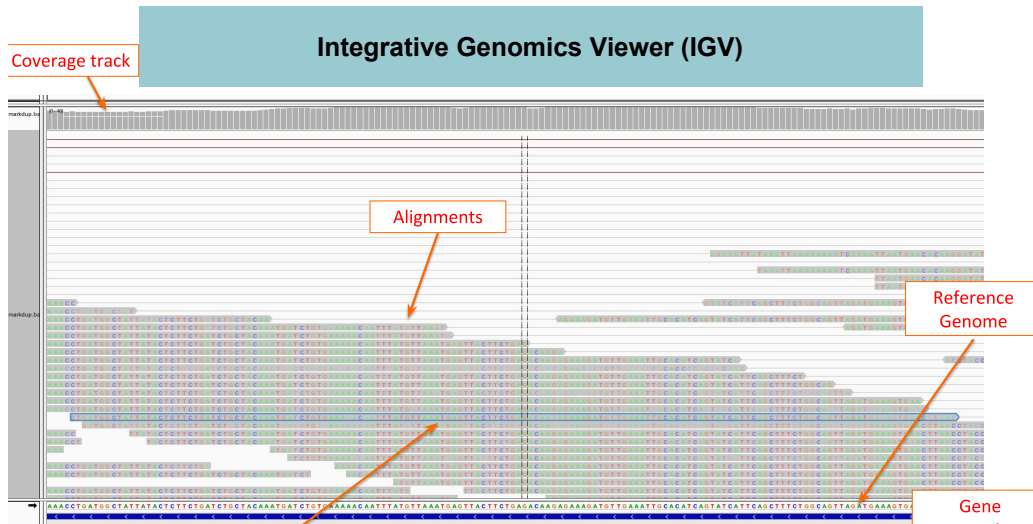
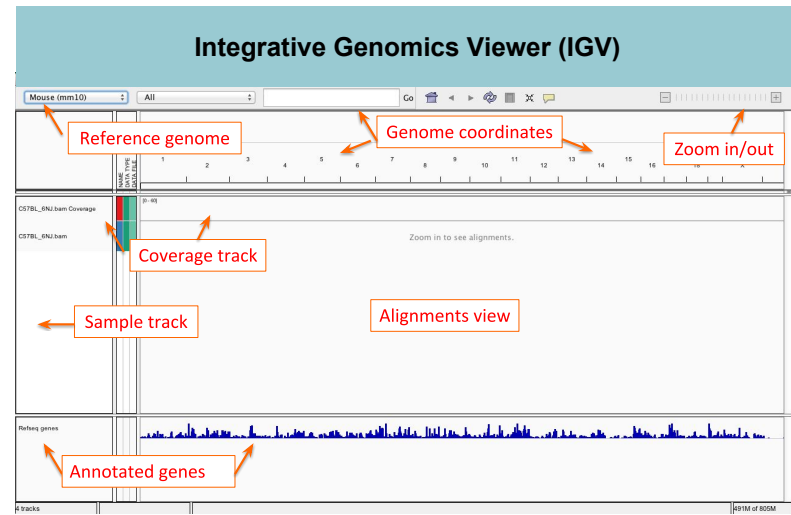
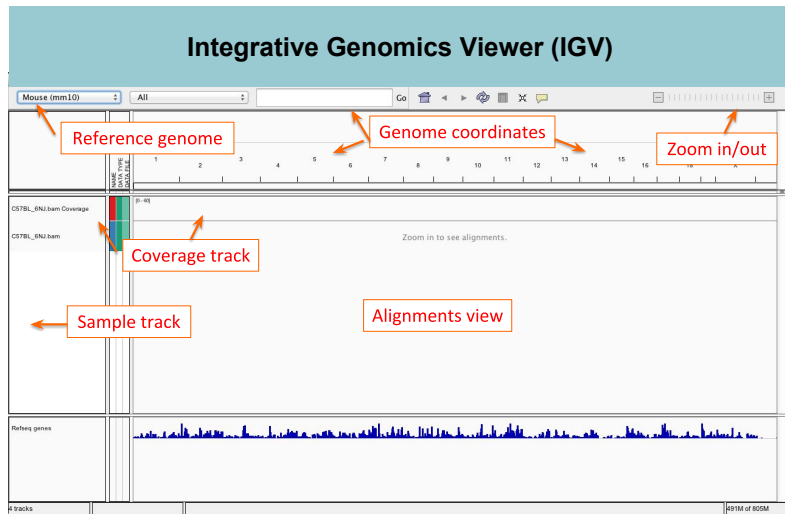
The whole point of short-read sequencing is to infer variation etc from PILEUP – the cooperative action of lots of reads together, all having the same variation at the same point  
 Samtools doesn't cut it.

```
HX8_24050:5:2103:20770:9449 83 7 4189207 0 151M = 4188954 -404 GTTG <JJJ MC:Z:151M MD:Z:151 NM:i:0 AS:i:151 XS:i:151
...
HX8_24050:5:2103:20770:9449 163 7 4188954 0 151M = 4189207 404 TTGG AAFP MC:Z:151M MD:Z:151 NM:i:0 AS:i:151 XS:i:151
...
HX8_24050:3:2212:32319:38772 99 7 4188947 0 151M = 4189300 436 ATGG AAAP MC:Z:68S83M MD:Z:151 NM:i:0 AS:i:151
XS:i:151
...
HX8_24050:3:2212:32319:38772 147 7 4189300 0 68S83M = 4188947 -436 ACAG --7- MC:Z:151M MD:Z:5C10T4T61 NM:i:3 AS:i:68
XS:i:175
```



## Option 3: Integrative Genomics Viewer (IGV)





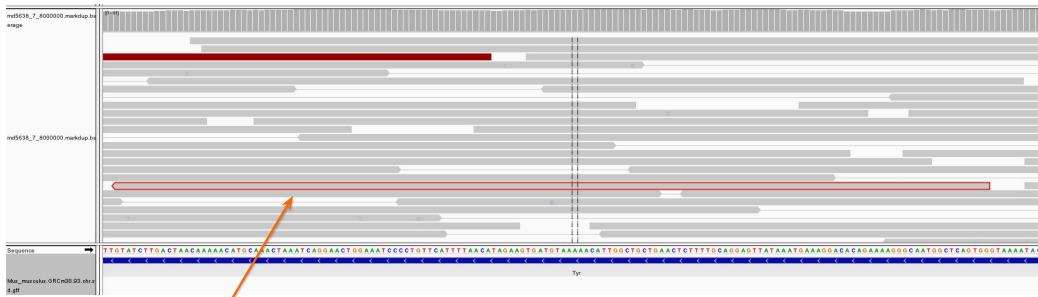
samtools view shows this output: blue-outlined read is the first line

```
HX8_24050:3:2122:30411:65986 163 7 87483999 60 151M = 87484301 453 CTGATGGCT
HX8_24050:3:2122:30411:65986 83 7 87484301 60 151M = 87483999 -453 TGTCACTAG
```

samtools view shows this output: blue-outlined reads are a pair

```
HX8_24050:3:2122:30411:65986 163 7 87483999 60 151M = 87484301 453 CTGATGGCT
HX8_24050:3:2122:30411:65986 83 7 87484301 60 151M = 87483999 -453 TGTCACTAG
```

## Integrative Genomics Viewer (IGV)

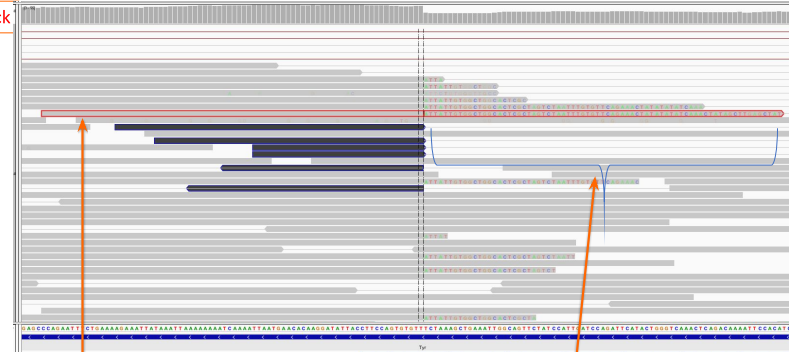


samtools view shows this output: red-outlined read is the 3' read

```
HX8_24050:1:2112:29315:29173 147 7 87484452 60 151M = 87483756 -847 GTAT|HX8_24050:1:2112:29315:29173 99 7 87483756 60 78M73S = 87484452 847 CCAG
```

## Integrative Genomics Viewer (IGV)

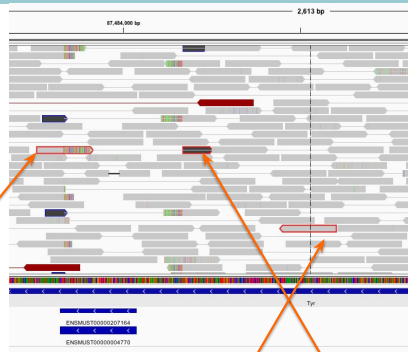
Coverage track



samtools view shows this output: red-outlined read is the 5' read

Notice the CIGAR string is 78M 73S – softclipped means that the alignment STOPS at bp 78, but the rest of the read is NOT trimmed. Result is that EVERY BASE PAIR of the read will mismatch ref => bases are coloured!

## Integrative Genomics Viewer (IGV)



But wait! There's more

samtools view will actually show three lines in the file with the same read name.

We have examined the left read (softclipped) and the right read (perfectly aligned).

The middle read alignment is the 'right half' of the left read. It is hard-clipped and marked as 'secondary' – see flags

```
HX8_24050:1:2112:29315:29173 99 7 87483756 60 78M73S = 87484452 847 CCAG
```

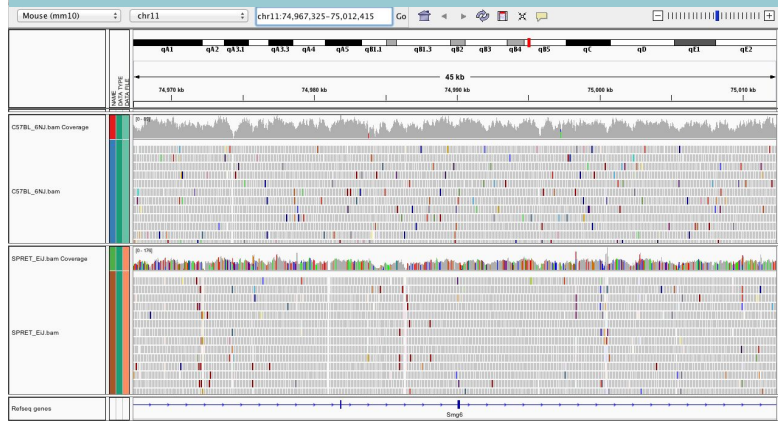
```
HX8_24050:1:2112:29315:29173 2147 7 87484159 60 76H75M = 87484452 434 TTATT
```

```
HX8_24050:1:2112:29315:29173 147 7 87484452 60 151M = 87483756 -847 GTAT
```

## Viewing reads as pairs

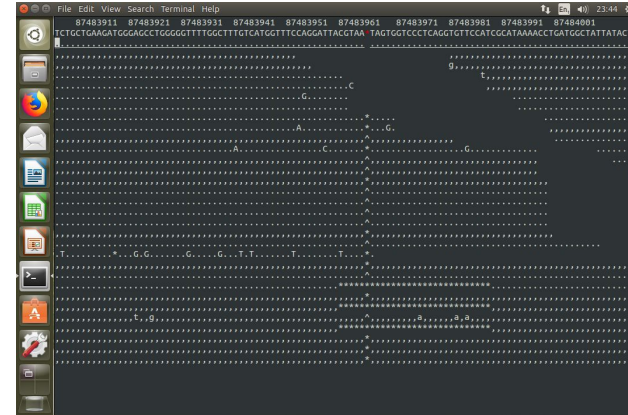


## Two samples



## Option 2: samtools tview

If you were stuck on a desert island, this would do the job: and you can try it for fun during the practical!  
`samtools tview md5638.markdup.bam ../ref/GRCm38.68.dna.toplevel.fa`



## Overview

Intro

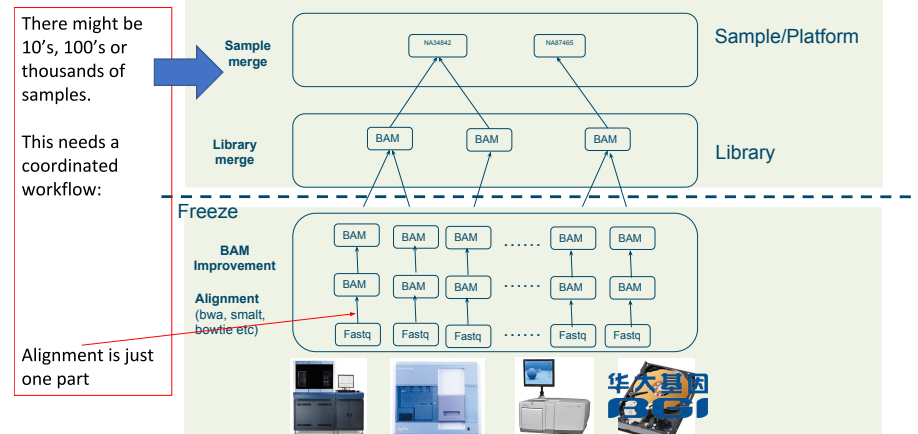
Methods / Aligners

Alignment Outputs

Alignment Viewers

NGS Workflows, QC and BAM Improvement

## A typical NGS workflow

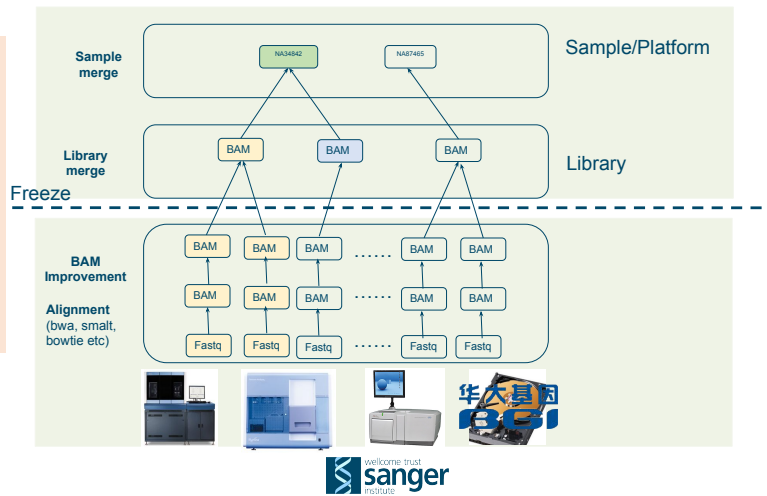


## A typical NGS workflow

Typically in a production workflow:

One sample spread over multiple seq libraries.

One library spread over multiple seq runs (lanes)

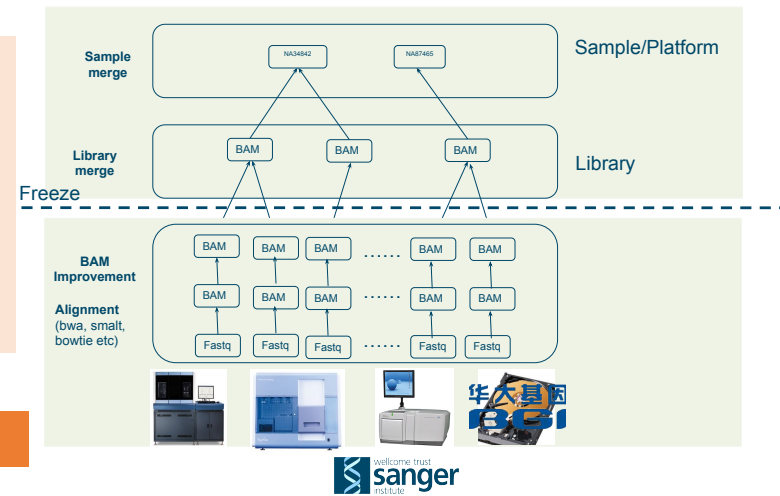


## Data production workflow

Typically in a production workflow:

One sample spread over multiple seq libraries.

One library spread over multiple seq runs (lanes)



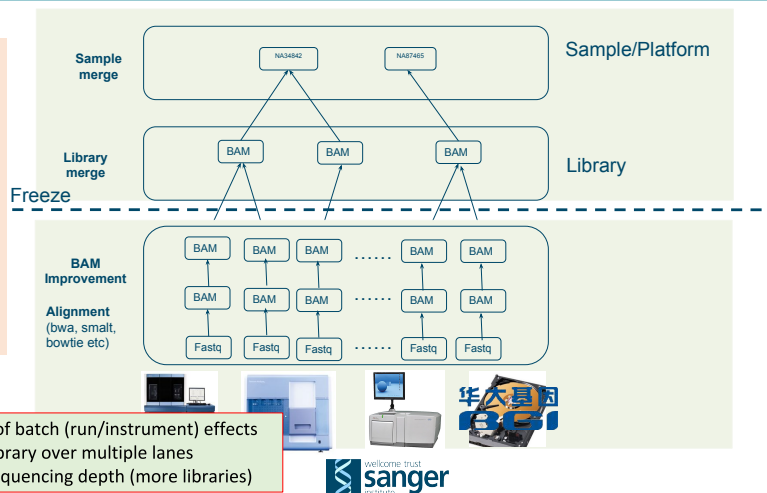
WHY?

## Data production workflow

Typically in a production workflow:

One sample spread over multiple seq libraries.

One library spread over multiple seq runs (lanes)

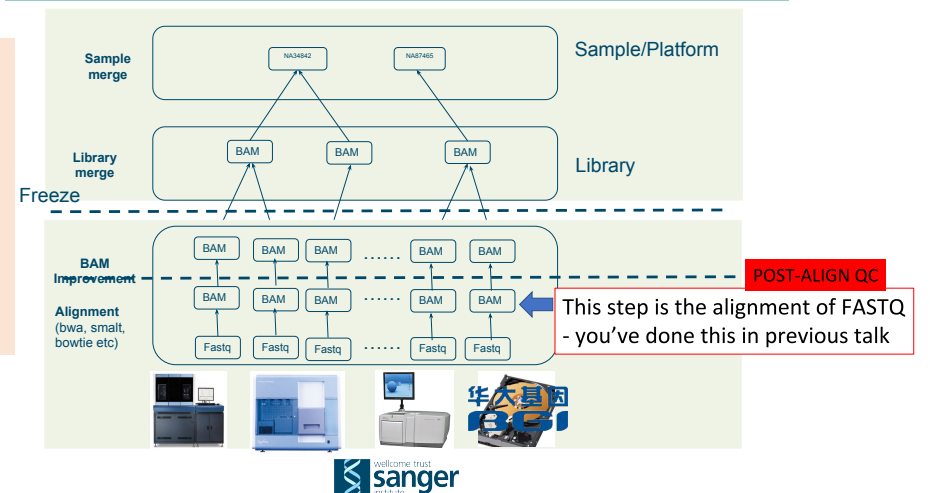


## Data production workflow

Typically in a production workflow:

One sample spread over multiple seq libraries.

One library spread over multiple seq runs (lanes)



## Data QC from alignments – a brief revision

```
samtools stats my_aligned_reads.bam > my_aligned_reads.stats.txt  
plot-bamstats my_aligned_reads.stats.txt
```

Already covered by the QC lecture: several useful metrics to assess the quality of your data and alignments produced:

- Mapping rates – should be > 90%
- Duplicate marked rates – some want 5%, I see anectdotally around 15%
- GC content vs read depth – which does need mapping (why?). ~ Flat.
- Fragment size distribution. *Depends on read length and insert size.*
- Indels by cycle. *No weird spikes.*

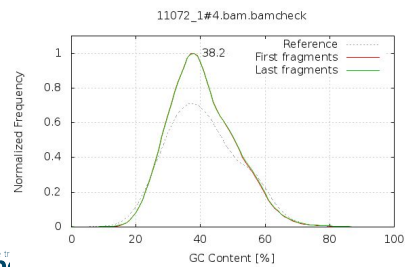
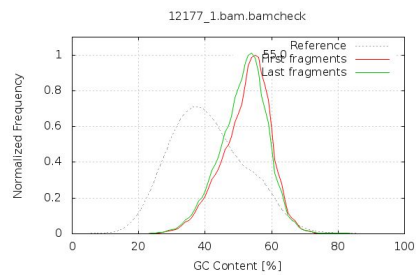


## Data QC from alignments – a brief revision

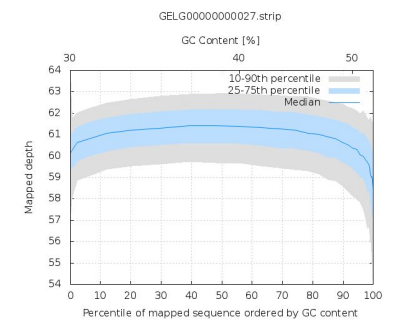
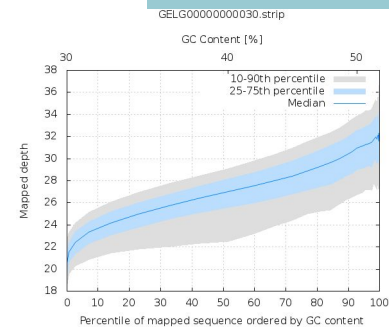
# And now, a quiz!



## GC of reads

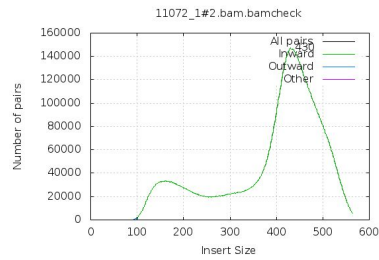
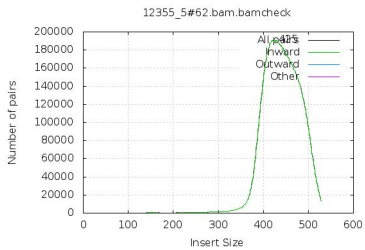


## GC vs depth

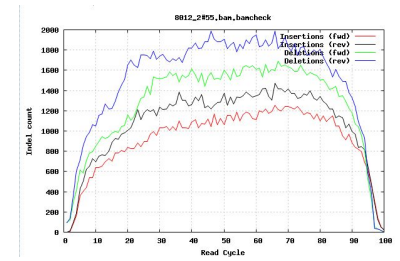
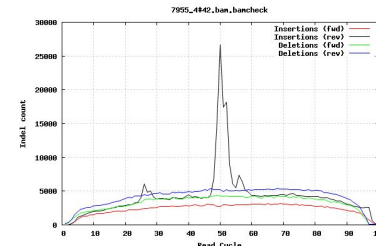




## Fragment size



## Indels per cycle



## Data QC from alignments – a brief revision

**QUIZ IS OVER**

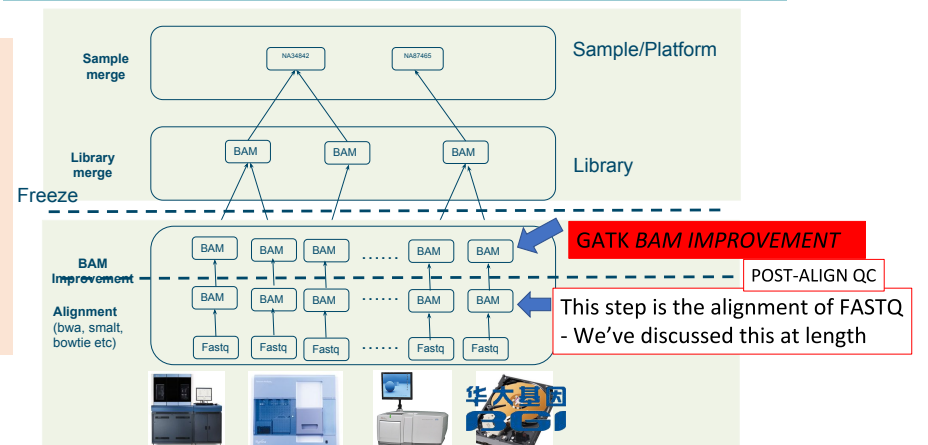


## Data production workflow

Typically in a production workflow:

One sample spread over multiple seq libraries.

One library spread over multiple seq runs (lanes)



## BAM improvement

FIRST - align each separate FASTQ: library x lane

Input: BAM

~~Process 1: INDEL (local) realignment~~

Process 2: Base quality recalibration

Output: BAM

BAM improvement

Part of ...

“GATK Best

Practice”

Merge independent lanes BAMs in same library together

Process 3: library - level mark-duplication



## BAM Improvement - Indel Realignment

*THIS IS NOW DEPRECATED (even by GATK!).*

I will mention it briefly because you may see it in different contexts.

PROBLEM: read aligners act on each read individually.

If the read had an indel,

THEN the aligner might get it right.

OR – if the indel happened near the edge of a read – it may get alignment wrong and miss the indel.

THEN the read would have lots of MISMATCHES instead of a single indel.



## Indel Realignment

NA12878, chr1:1,510,530-1,510,589

M.A. DePristo et al., Nature Genetics 2011

### INDEL Realignment algorithm

FOR indel sites and a BAM file

At each site, model the **indel haplotype** vs the **reference haplotype**

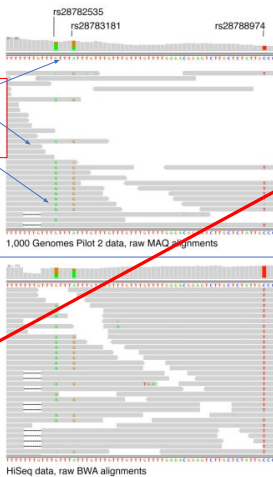
Which alignment minimises ALL the observed mismatches?  
indel or no indel?

Result: Group alignment for *all* the reads (not one-by-one)

Result: New BAM file produced with read cigar lines modified where indels have been introduced by the realignment process

1. "...GTTTGTTTATT..."
2. These reads mis-aligned
3. These SNPs are false

Before Indel Realign



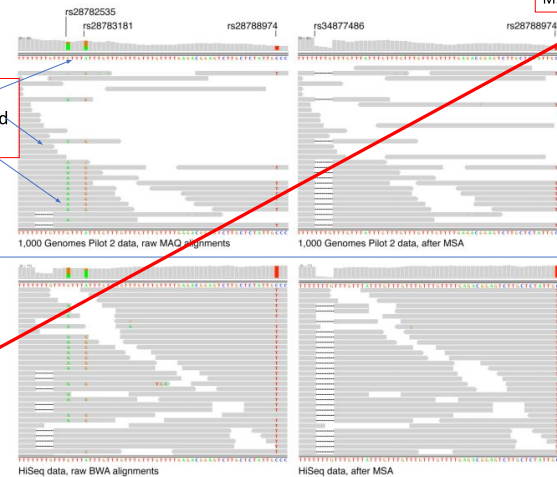
## Indel Realignment

NA12878, chr1:1,510,530-1,510,589

M.A. DePristo et al., Nature Genetics 2011

1. "...GTTTGTTTATT..."
2. These reads mis-aligned
3. These SNPs are false

Before Indel Realign



After Indel Realign

## BAM Improvement - Indel Realignment

*THIS IS NOW DEPRECATED (even by GATK!).*

Reads are getting longer and aligners are getting better – the problem is going away.

Local realignment now automatically performed by variant callers (downstream) ANYWAY.

So GATK have removed it from their "best practice" docs.



## BAM Improvement - Base Quality Score Recalibration

Each base call has an associated base call quality

- What is the chance that the base call is incorrect?
  - Illumina evidence: intensity values + cycle
- Phred values (log scale)
  - Q10 = 1 in 10 chance of base call incorrect
  - Q20 = 1 in 100 chance of base call incorrect
- Accurate base qualities essential measure in variant calling

**Rule of thumb: Anything less than Q20 is not useful data**

**BUT:**

The base quality scores produced by a sequencer can be influenced by *systematic technical error*:

They can vary with sequence context, position in read etc.

Therefore the quality score can be off

Therefore variant calling can be influenced.

BQSR: adjusts the quality of each base to adjust for these systematic errors



<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3409179/>

## Base quality recalibration

The idea of recalibration:

- Use the alignment of your reads to a human reference
- **Remove** all known variants dbSNP+1000G etc SNP sites
- **Assume** all other mismatches in your data **are sequencing errors**
- Now you can infer the mean observed ("empirical") error rate for these bins:
  - Position of base in read (1 => length of read)
  - Dinucleotide Sequence Context: AA, AT ..., CA, CT ...
  - Empirical quality = number of mismatches / total number in bin.
- Compare empirical rate to *reported* sequencer base quality in each of these categories.
- Derive an adjustment for each category, to be applied to each reported base quality value

e.g. Sequencer reports Q25 base calls for a "T" in context "AT"

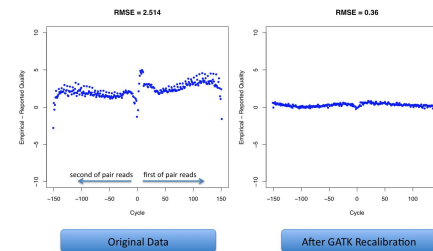
- After alignment - it may be that "T" in context "AT" actually mismatches the reference at a 1 in 100 rate, so are actually Q20
- Adjustment: for every T in "AT" – subtract 5 from BQ. (30=>25, 21=> 16 etc)

**NOTE:** requires a reference genome and a catalog of variable sites

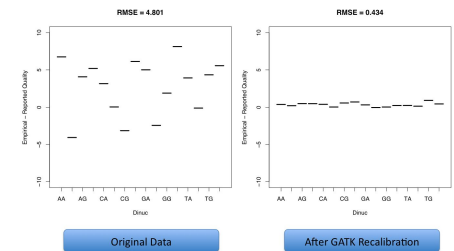


## Base quality recalibration effects

### Residual Error by Machine Cycle

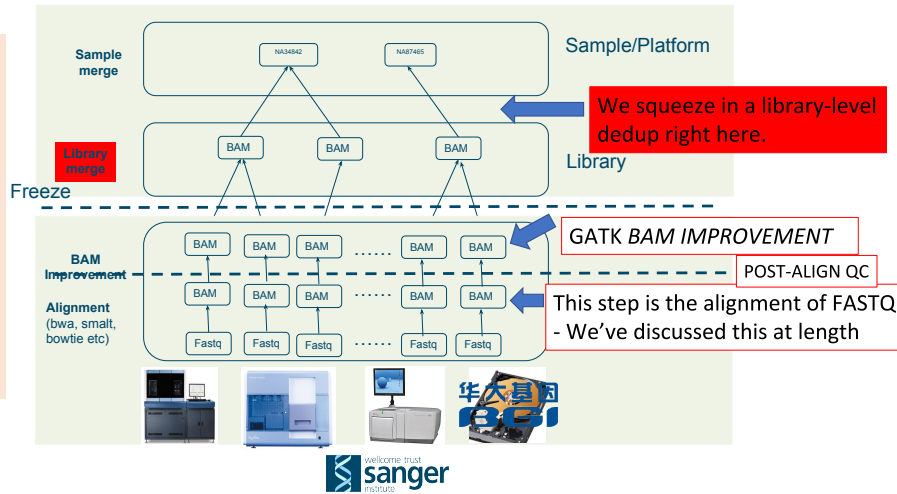


### Residual Error by Dinucleotide



## Data production workflow

One sample  
...spread over...  
Multiple libraries  
One library  
...spread over...  
Multiple runs (lanes)



## Library duplicates

All second-gen sequencing platforms are NOT single molecule sequencing

- PCR amplification step in library preparation
- Can result in duplicate DNA fragments in the final library prep.
- PCR-free protocols do exist – require larger volumes of input DNA

Problem: can result in false SNP calls

- Duplicates manifest themselves as high read depth support

Solution:

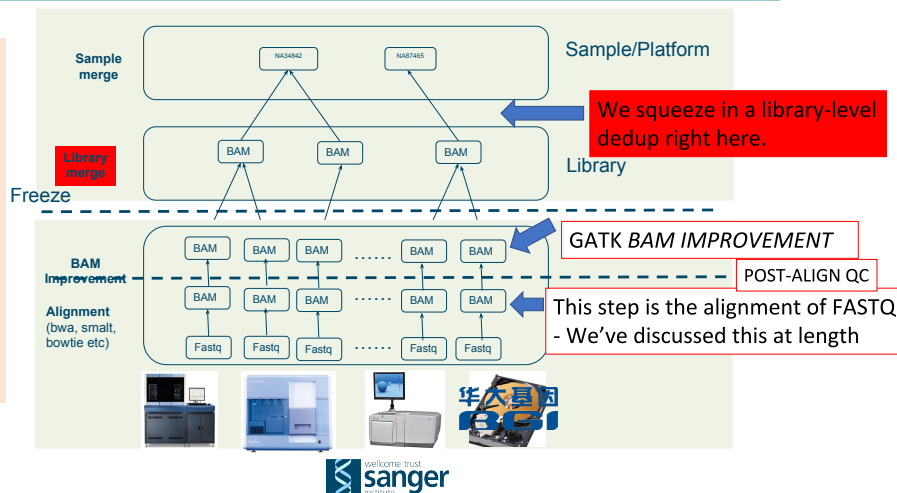
- Align reads to the reference genome
- Identify read-pairs where the **outer ends** map to the same position on the genome and remove all but 1 copy
  - Samtools: samtools rmdup
  - Picard/GATK: MarkDuplicates

Generally low number of duplicates in good libraries (<5%). *But I see ~15%*



## Data production workflow

One sample  
...spread over...  
Multiple libraries  
One library  
...spread over...  
Multiple runs (lanes)



## Overview

Intro

Methods / Aligners

Alignment Outputs

Alignment Viewers

Alignment summary statistics and QC – BRIEF REVISION

After alignment: BAM file improvement before variant calling

If you are an consumer then you should have a grasp of this

An informatics group should have control of this, but you should be aware that it's being done.



# WTAC Next Generation Sequencing Bioinformatics Course

## Read Alignment Practical Exercises

### Introduction

We will first “hand-run” a Smith Waterman (local) alignment between two sequences.

We will then align whole-genome sequencing from a mouse zygote which was subject to CRISPR-induced mutagenesis. We will find the resulting engineered alleles, and track down some other alleles in this mouse. This section will be “cued”.

We will then align sequences to a yeast genome, this time with less prompting!

### Required Data

For this lab, some pre-prepared datasets have been installed on the Virtual Machine (VM) for you. Double-click the “Module 3 Read Alignment” icon on your desktop to open a terminal window. This window should open in the Read Alignment module directory containing all of the materials needed to complete this practical.

In this directory, there are two exercise directories (Exercise2 and Exercise4) which contain the various data used throughout the practical. There is also a directory titled ‘ref’ which contains the reference genome (FASTA format) of *M. musculus* and *S. Cerevisiae* which will be used in Exercises 2 and 4.

## Exercise 1: Do a Smith-Waterman alignment

### 1.1: Here are the rules and an example of aligning “CTGAG” vs “TAG”.

#### Scoring:

Match score: +1

Mismatch score: -1

Gap penalty: -1

**Starting cells:** fill in first row & first column with “0”s.

#### Rules for cell scoring:

For any given cell, you can enter from the left side, top or diagonally.

If you entered from top or side:

$$\text{Score in cell} = \text{score in prior cell} + \text{gap penalty} = \text{score in prior cell} - 1$$

If you entered diagonally:

$$\text{Score in cell} = \text{score in prior cell} + 1 \text{ if sequences match}$$

OR

score in prior cell - 1 if sequences mismatch

**Choose the highest of these scores!** And remember *which* entry cell generated the highest score!

If score in cell is negative, replace score with 0!

#### Rules for traceback:

Start with highest scoring cell, and trace back to cell of entry (ie did the high score arise when you enter your current cell diagonally or from the sides).

#### Writing out alignment.

If cell is entered diagonally, write out match or mismatch.

If cell is entered from sides, then write out a gap in the appropriate sequence.

Example of scoring:

		C	T	G	A	G
	0	0	0	0	0	0
C	0	1 -1 -1 1	-1 -1 0 0	-1 -1 -1 0	-1 -1 -1 0	-1 -1 -1 0
G	0	-1 0	0 -1	1 -1	-1 -1	-1 -1

		-1 0	-1 0	-1 1	0 0	-1 0
A	0	-1 -1	-1 -1	-1 0	2 -1	-1 -1
		-1 0	-1 0	-1 0	-1 2	1 1

Now choose the highest score in the matrix and trace-back , choosing the cell which resulted in the highest score.

		C	T	G	A	G
	0	0	0	0	0	0
C	0	-1 -1	-1 -1	-1 -1	-1 -1	-1 -1
		-1 1	0 0	-1 0	-1 0	-1 0
G	0	-1 0	0 -1	1 -1	-1 -1	-1 -1
		-1 0	-1 0	-1 1	0 0	-1 0
A	0	-1 -1	-1 -1	-1 0	2 -1	-1 -1
		-1 0	-1 0	-1 0	-1 2	1 1

Alignment: If a cell was entered diagonally, write a match/mismatch between two sequences. If it was entered horizontally / vertically, enter a gap in one sequence or the other:

C T G A G  
C - G A -

**1.2: Repeat with these two sequences: "CTGAG" and "TAG"**

CTGAG  
And  
TAG

		C	T	G	A	G
	0	0	0	0	0	0
T	0					
A	0					
G	0					

Solution:

		C	T	G	A	G
	0	0	0	0	0	0
T	0	-1 -1 -1 0	-1 -1 -1 1	-1 -1 0 0	-1 -1 -1 0	-1 -1 -1 0
A	0	-1 -1 -1 0	-1 0 -1 0	0 -1 -1 0	1 -1 -1 1	-1 -1 0 0
G	0	-1 -1 -1 0	-1 -1 -1 0	1 -1 -1 1	-1 0 0 0	2 -1 -1 2

C T G A G  
- T - A G



## Exercise 2: BWA Alignment / inspection of mouse variation

We will use the BWA aligner to align one small region of illumina sequencing data to the *Mus Musculus* genome. You will align genomic sequence (from Whole-Genome Sequencing) from a mouse embryo which has been mutagenised while the one-cell stage using CRISPR-Cas9 and a gRNA targeting an exon of the *Tyr* gene. The successful mutation of the gene will delete one or both alleles. A bi-allelic null *Tyr* mouse will be albino, but otherwise healthy.

### 2.1: View the the reference genome

**Goto the 'ref' directory** that contains the fasta files of the reference genomes:  
~/course\_data/Module3\_ReadAlignment/ref

Fasta files (.fa) are used to store raw sequencing information before aligning data. The mouse genome file is here: GRCm38.68.dna.toplevel.fa

**View the file with less:**  
\$ less GRCm38.68.dna.toplevel.fa

**Question:** What is the length of chromosome 1 of the mouse genome? (*Look at the fasta header for chromosome 1*)

**Question:** Can you *quickly* check if there other sequences in the assembly other than the 'standard' chromosomes? (*Try grep '>' GRCm38.68.dna.toplevel.fa*)

Similar to a BAM file, to allow fast retrieval of data, and index file is often required. In this case we have already created both the fasta index for the genome:

GRCm38.68.dna.toplevel.fa.fai – allows rapid sequence retrieval with samtools  
GRCm38.68.dna.toplevel.fa.amb ... GRCm38.68.dna.toplevel.fa.sa – created by bwa: suffix trees, bwt transform etc etc.

### 2.2: Align the paired fastq files with bwa

Goto the '~/course\_data/Module3\_ReadAlignment/Exercise2/fastq/' directory. We will align the fastq files to the mouse reference genome using bwa.

**Use the 'bwa mem' command to align the fastq files.** Bwa outputs sam files by default, so you will have to pipe the result into a sam file.

```
bwa mem ~/course_data/Module3_ReadAlignment/ref/GRCm38.68.dna.toplevel.fa md5638a_7_87000000_R1.fastq md5638a_7_87000000_R2.fastq > md5638.sam
```

### 2.3: Convert a SAM file to a BAM file

Now we need to convert the SAM file ('md5638.sam') from the previous step into a BAM file.

**Convert the SAM file into a BAM file** called 'md5638.bam' using samtools.

**Hint:** to do this conversion use 'samtools view'. What options are required to input a SAM file and output a BAM file?

```
samtools view -O BAM -o md5638.bam md5638.sam
```

**How much space is saved by using a bam file instead of sam?**

## 2.4: Sort and index the BAM file

The BAM files produced by BWA are sorted by read name (same order as the original fastq files). However, most viewing and variant calling software required the BAM files to be sorted by reference co-ordinate position and indexed for rapid retrieval:

**Therefore, use 'samtools sort' to produce a new BAM file** ('md5638.sorted.bam') that is sorted by position.

**Finally, can you index the sorted BAM file** using 'samtools-1.5 index' command?

Note: indexing a BAM file is also a good way to check that the BAM file has not been truncated (e.g. your disk becomes full when writing the BAM file). At the end of every BAM file, a special end of file (EOF) marker is written. The Samtools index command will first check for this and produce an error message if it is not found.

```
samtools sort -T temp -O bam -o md5638.sorted.bam md5638.bam
```

```
samtools index md5638.sorted.bam
```

## 2.5: Unix pipes to combine the commands together

To produce the sorted BAM file in 2.1-2.3 we had to carry out several separate commands and produce intermediate files. The Unix pipe command allows you to feed the output of one command into the next command.

**Combine all of these commands together using unix pipes**, and do all of this data processing together and avoid writing intermediate files.

```
bwa mem ~/course_data/Module3_ReadAlignment/ref/GRCm38.68.dna.toplevel.fa  
md5638a_7_87000000_R1.fastq md5638a_7_87000000_R2.fastq | samtools view -O BAM  
- | samtools sort -T temp -O bam -o md5638.sorted.bam -
```

## 2.5: Mark PCR Duplicates

We will use a program called 'MarkDuplicates' that is part of Picard tools (<http://picard.sourceforge.net>) to remove PCR duplicates that may have been introduced during the library construction stage. To find the options for 'MarkDuplicates' – type:

```
picard-tools MarkDuplicates
```

**Now run MarkDuplicates** using the 'I=' option to specify the input BAM file and the 'O=' option to specify the output file (e.g. md5638.markdup.bam). You will also need to specify the duplication metrics output file using 'M=' (e.g. md5638.markdup.metrics). Don't forget to index your final bam file using 'samtools index'.

From looking at the output metrics file - how many reads were marked as duplicates? What was the percent duplication?

```
picard-tools MarkDuplicates I= md5638.sorted.bam O=md5638.markdup.bam  
M=md5638.metrics.txt
```

**Generate an index for the bam file** using samtools.

```
samtools index md5638.markdup.bam
```

## 2.5: Generate QC Stats

Use samtools to collect some statistics about the alignments in the BAM file from the last step. To run the 'stats' command - type:

```
samtools stats md5638.markdup.bam > md5638.markdup.stats
```

Then look at the output and answer the following questions:

What is the total number of reads?

What proportion of the reads were mapped?

How many reads were paired correctly/properly?

How many reads mapped to a different chromosome?

What is the insert size mean and standard deviation?

Next we will create some QC plots from the output of the stats command. Make sure you have saved the output of the stats command to a file (e.g. lane1.stats.txt). We will use the 'plot-bamstats' command that is part of Samtools:

```
plot-bamstats -p md5639_plot md5638.markdup.stats
```

Now in your web browser open the file called `md5639_plot.html` to view the QC information.

How many reads have zero mapping quality?

Do any of the graphs look odd to you?

Which of the first fragments or second fragments are higher base quality on average? Note: Look at the first of the 'Quality per cycle' graphs.

## 2.6: BAM Visualisation

***Congratulations! You made it to the Good Bit!** I find actually seeing the seeing sequence variation – natural or engineered - and pondering its relation to biological effects quite compelling. I hope you do too!*

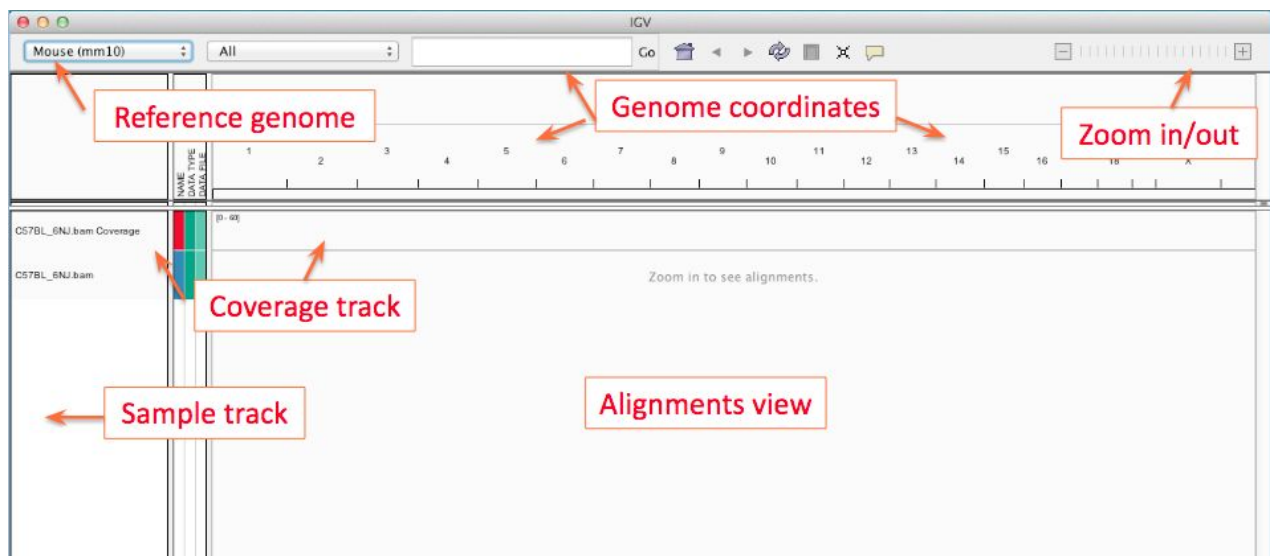
IGV (<http://www.broadinstitute.org/igv/>) is a very useful visualisation tool for looking at the alignments of reads onto a reference genome from BAM files.

In the 'fastq' directory, you can start IGV by typing:

```
igv.sh
```

Here's a great IGV tutorial and refresher:

<https://github.com/sanger-pathogens/pathogen-informatics-training/blob/master/Notebooks/IGV/IGV.pdf>



**2.6.1 Load the reference genome**, on the top menu bar find the genomes dropdown (top-left) and select “mouse mm10”. This is a synonym for GRCm38, which is the current mouse assembly.

If - for whatever reason, this fails - you can load the genome from a file (Genomes=>Load From File using the genome in the “ref” directory:

**GRCm38.68.dna.toplevel.fa**

and load gene annotations from file File=>Load from file using this file:

**Mus\_musculus.GRCm38.93.chr.sorted.gtf**

**2.6.2 Load your BAM file**, on the top menu bar goto ‘File -> Load from File...’ and select the md5638.markdup.bam file that you created in the previous step.

**2.6.3 Set up basic view preferences**, using:

**Popup preferences** – use the little “speech bubble” button on the top icon list to set popups on *click only* (*or you will go mad, I promise*).

**Track preferences** (Command-click on the track at left). Sort alignments by insert size. Colour alignments by insert size.

**View preferences** (View menu item): View=>preferences=>Alignments. Show soft-clipped bases. *This colour highlighting emphasises soft-clips on the read itself.*

**2.6.4 Inspect an interesting region of the mouse Tyr locus.**

Go to chromosome 7, positions 87,483,625-87,484,330 using the navigation bar across the top. *Take in the glorious view of a genome pileup. Stop and smell the roses! Click on stuff! Scroll around, zoom in and out a bit!*

**2.6.5 Questions about the pileup and visible variation:**

Go back to chromosome 7:87,483,625-87,484,330. What is the (rough) coverage across this region?

There are three mutant variants (two small and one larger) in this region: Can you spot them, state what the evidence is for them, and work out their allele fraction? Can you venture a guess as to what happened here? Why are these mutations present? Why might they be subclonal?

Hints:

1. Look around 87,483,960 for an insertion. How large is it? How many reads does it occur in?
2. Look around 87,483,960 for a deletion. How large is it? How many reads does it occur

in?

3. Zoom out slightly and watch the coverage track between 87,483,700 - 87,484,200. Once you've spotted the large change look at reference sequence the edges of the mutation to hazard a guess as to its mechanism.

**What mutations can you see?** There is a 1bp insertion (at "T") with VAF ~0.3 at position 7:87,483,965. There is a 28bp deletion with VAF ~0.15 starting at 7:87,483,960. There is a 338bp deletion with a VAF of about 0.25 starting at position 7:87,483,831. **Why are they there?** The CRISPR-Cas9 has acted on the zygote at this locus to create Non-Homologous-End-Join-based damage around 87,483,960: that resulted in a subclonal 1bp insertion and a 28bp deletion. Microhomology-induced-end-joining resulted in the 338bp deletion (can you see the "TTT" motif on the 5' end of the deletion, and just inside the 3' end of the deletion? You are watching the zygote DNA-repair machinery panicking and grabbing at straws). **Why are these alleles subclonal?** Because the action of the CRISPR-Cas9 occurred both at the single-cell and the two-cell stage.

### 2.6.6 Looking for natural SNV's and Indels

At each of the following genomic locations, write down the variant, its allele fraction, and whether you can find it in the Mouse Genomes Search facility:

(<https://www.sanger.ac.uk/sanger/Mouse/SnpViewer/>)

Location1. 7:87258490

Location2. 7:87834414

Location3. 7:87251720

Location4. 7:87303315

Location5. 7:87392116

Location6. 7:87428859

Answer:

1. Hom SNV/no,

2. Hom SNV/no,

3. Hom 2bp and Hom 4bp deletion/sort of: The MGP shows a 4p deletion only, and this is one region which may have benefited from indel-realignment! Note excess repeats in the area surrounding.

4. Hom single bp deletion/yes.

5. Hom single bp insertion/yes.

6. Hom (?) 2b deletion/yes.

## Exercise 4: BWA Alignment / lane merging / inspection with YEAST

We will use the BWA aligner to align 2 lanes of illumina sequencing data to the *Saccromyces cerevisiae* genome

([ftp://ftp.ensembl.org/pub/current\\_fasta/saccharomyces\\_cerevisiae/dna/](ftp://ftp.ensembl.org/pub/current_fasta/saccharomyces_cerevisiae/dna/)).

### 4.1: Index the reference genome with bwa

Goto the 'ref' directory that contains the fasta files of the reference genome. Fasta files (.fa) are used to store raw sequencing information before aligning data. Similar to a BAM file, to allow fast retrieval of data, and index file is often required. You can use the 'bwa index' command to create a reference genome index that bwa can use.

Do you see any new files? How many?:

**Note:** Generally, when a tool creates an index for a file, the index will have almost the same name as the original but include a new ending. For example, do you see a file ending in .bwt?:

### 4.2: Align the lane fastq files with bwa

Goto the 'Exercise4/60A\_Sc\_DBVPG6044/library1/lane1/' directory and we will align the fastq files using bwa.

When aligning data, we often want include additional information of relevance to a project in the header of a file. Next create a lane SAM file called 'lane1.sam' with the following SAM header (**Hint:** type 'bwa mem' and look for the "Input/output options" section of the printed help page):

```
'@RG\tID:lane1\tSM:60A_Sc_DBVPG6044'
```

Use the 'bwa mem' command to align the fastq files and use the appropriate option to include the about header information. Don't forget to use the -M option (mark shorter split hits as secondary).

**Hint:** This will create a SAM so needs to be output to a file called lane1.sam (> lane1.sam).

### 4.3: Convert a SAM file to a BAM file

Now we need to convert the SAM file ('lane1.sam') from the previous step into a BAM file. Convert the SAM file into a BAM file called 'lane1.bam' using samtools.

**Hint:** to do this conversion use 'samtools view'. What options are required to input a SAM file and output a BAM file?

#### 4.4: Sort and index the BAM file

The BAM files produced by BWA are sorted by read name (same order as the original fastq files). However, most variant calling software required the BAM files to be sorted by reference co-ordinate position to allow rapid retrieval of data. Therefore, use 'samtools sort' to produce a new BAM file ('lane1.sorted.bam') that is sorted by position.

Look at the first line of the header of the BAM file, is it coordinate sorted?:

Finally, can you index the sorted BAM file using 'samtools index' command?

Note: indexing a BAM file is also a good way to check that the BAM file has not been truncated (e.g. your disk becomes full when writing the BAM file). At the end of every BAM file, a special end of file (EOF) marker is written. The Samtools index command will first check for this and produce an error message if it is not found.

#### 4.5: Unix pipes to combine the commands together

To produce the sorted BAM file in 2.1-2.3 we had to carry out several separate commands and produce intermediate files. The Unix pipe command allows you to feed the output of one command into the next command.

So using Unix pipes, we can combine all of these commands together and do all of this data processing together and avoid writing intermediate files.

```
bwa mem -M -R '@RG\tID:lane1\tSM:60A_Sc_DBVPG6044'
../../../../ref/Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa s_7_1.fastq
s_7_2.fastq | samtools view -bS - | samtools sort -T temp -O bam -o
lane1.sorted.bam -
```

#### 4.6: Generate QC Stats

We will use samtools to collect some statistics about the alignments in the BAM file from the last step (remember to output this to a stats file, e.g. a file ending in .stats.txt). To run the 'stats' command - type:

```
samtools stats lane1.sorted.bam
```

Then look at the output and answer the following questions (**Hint** look for rows beginning with SN):

What is the total number of reads?

What proportion of the reads were mapped?



How many reads were paired correctly/properly?

How many reads mapped to a different chromosome?

What is the insert size mean and standard deviation?

Next we will create some QC plots from the output of the stats command. Make sure you have saved the output of the stats command to a file (e.g. lane1.stats.txt). We will use the 'plot-bamstats' command that is part of Samtools:

```
plot-bamstats -p plot lane1.stats.txt
```

(Aside: If you have problems running `plot-bamstats`, you can download the results from here: <http://tinyurl.com/h37d8bx>).

Now in your web browser open the file called plots.html to view the QC information.

How many reads have zero mapping quality?

Which of the first fragments or second fragments are higher base quality on average? Note: Look at the 'Quality per cycle' graphs.

#### 4.7: Align Lane 2

There is a second lane in the 'library1' directory called 'lane2'. We want to also align this lane also to produce a BAM file.

Goto the 'Exercise4/60A\_Sc\_DBVPG6044/library1/lane2' directory. Now repeat exercise 2 using the fastq files in the lane2 directory to produce a sorted BAM file. **Note:** This time when you use the 'bwa mem' command use the following header option to specify lane2 as the read group ID:

```
'@RG\tID:lane2\tSM:60A_Sc_DBVPG6044'
```

#### 4.8: Merge the lane BAMS

Go to the '60A\_Sc\_DBVPG6044/library1' directory. Use 'ls' to get a listing of the files and directories contained in this directory.

You will notice that there are two directories called 'lane1' and 'lane2'. There were two sequencing lanes produced from this sequencing library. In order to mark library PCR duplicates, we need to merge the two lane BAM files together to produce a single BAM file.

We will use the picard tool called 'MergeSamFiles' (<http://picard.sourceforge.net>) to merge the lane BAM files. Picard-tools is a collection of commands (with their own options) for

interacting with BAM files. Look at the list of commands available for Picard-tools by running 'picard-tools'. To find the options for 'MergeSamFiles' command, type:

```
picard-tools MergeSamFiles
```

Now use the 'I=' option to specify both the input BAM files and the 'O=' option to specify the output file (e.g. library1.bam). **Note:** Multiple input files can be specified using 'I='

#### 4.6: Mark PCR Duplicates

We will use a program called 'MarkDuplicates' that is part of Picard tools (<http://picard.sourceforge.net>) to remove PCR duplicates that may have been introduced during the library construction stage. To find the options for 'MarkDuplicates' – type:

```
picard-tools MarkDuplicates
```

Now use the 'I=' option to specify the input BAM file and the 'O=' option to specify the output file (e.g. library1.markdup.bam). You will also need to specify the duplication metrics output file using 'M=' (e.g. library1.markdup.metrics).

Don't forget to index your final bam file using 'samtools index'.

From looking at the output metrics file - how many reads were marked as duplicates? What was the percent duplication?

#### 4.9: BAM Visualisation

IGV (<http://www.broadinstitute.org/igv/>) is a very useful visualisation tool for looking at the alignments of reads onto a reference genome from BAM files.

In the 'library1' directory, you can start IGV by typing:

```
Igv.sh &
```

To load the reference genome, on the top menu bar goto 'Genomes -> Load Genome From File...' and select the reference genome in the 'ref' directory.

Next to load your BAM file, on the top menu bar goto 'File -> Load from File...' and select the library BAM file that you created in the previous step.

Now goto Chromosome IV and position 764,293 using the navigation bar across the top.

What is the reference base at this position?

Do the reads agree with the reference base?

What about the adjacent position (IV:764,292)? What is the reference base at this position? Is it supported by the reads?

Now goto Chromosome IV and position 766,588 using the navigation bar across the top.

What sort of mutation are the alignments indicating might be present?

Now goto Chromosome IV and position 770,137 using the navigation bar across the top.

What sort of mutation are the alignments indicating might be present? Is there anything in the flanking sequence of the reference genome that might make you suspicious about this mutation?

### Software URLs

<b>Name</b>	<b>URL</b>
Burrows-Wheeler Aligner (BWA)	<a href="http://bio-bwa.sourceforge.net/">http://bio-bwa.sourceforge.net/</a> <a href="http://github.com/lh3/bwa">http://github.com/lh3/bwa</a>
Samtools	<a href="http://www.htslib.org">http://www.htslib.org</a>
Picard Tools	<a href="https://broadinstitute.github.io/picard/">https://broadinstitute.github.io/picard/</a>
IGV	<a href="http://software.broadinstitute.org/software/igv/">http://software.broadinstitute.org/software/igv/</a>



## Variant Calling - SNPs and short indels

petr.danecek@sanger.ac.uk



1 / 52

### HTS workflow

#### Library preparation

- DNA extraction
- fragmentation
- adapter ligation
- amplification

#### Sequencing

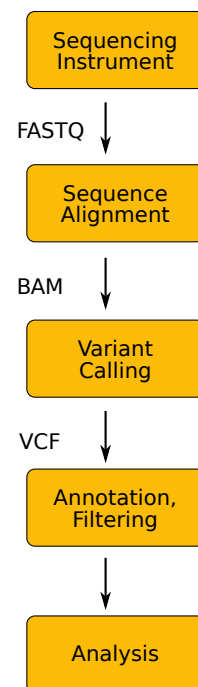
- base calling
- de-multiplexing

#### Data processing

- read mapping
- variant calling
- variant filtering

#### Analysis

- Variant annotation
- ...



2 / 52



# Germline vs somatic mutation

## Germline mutation

- heritable variation in the germ cells

## Somatic mutation

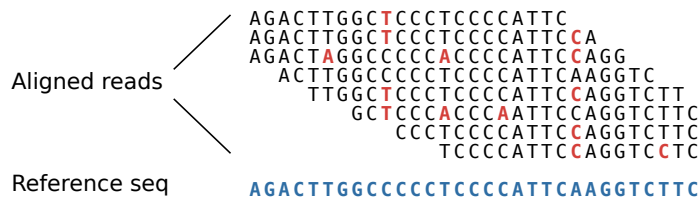
- variation in non-germline tissue, tumors. . .

## Germline variant calling

- expect the following fractions of alternate alleles in the pileup:
  - 0.0 for RR genotype (plus sequencing errors)
  - 1.0 for AA (plus sequencing errors)
  - 0.5 for RA (random variation of binomial sampling)

## Somatic

- any fraction of alt AF possible - subclonal variation, admixture of normal cells in tumor sample



6 / 52

# Naive variant calling

Use fixed allele frequency threshold to determine the genotype

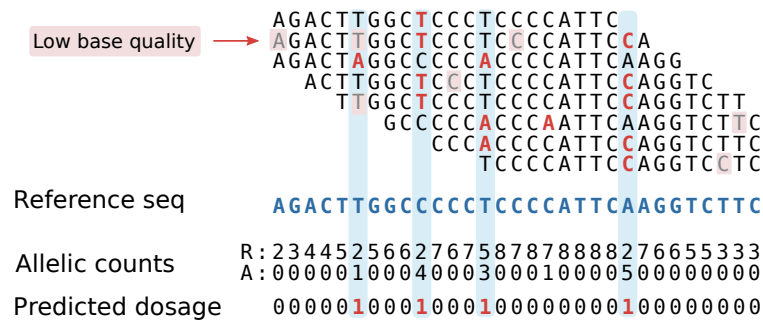


alt AF	genotype
[0, 0.2)	RR .. homozygous reference
[0.2, 0.8]	RA .. heterozygous
(0.8, 1]	AA .. homozygous variant

7 / 52

# Naive variant calling

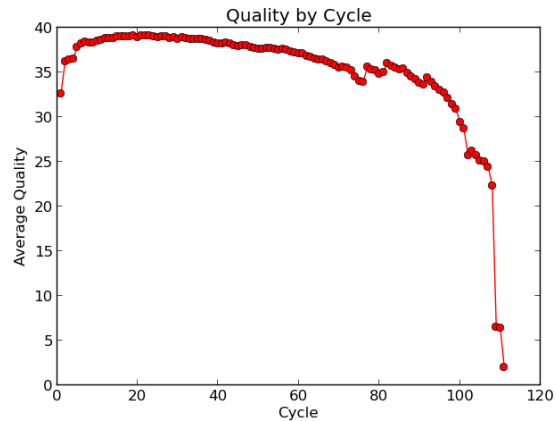
Use fixed allele frequency threshold to determine the genotype



1) Filter base calls by quality  
e.g. ignore bases  $Q < 20$

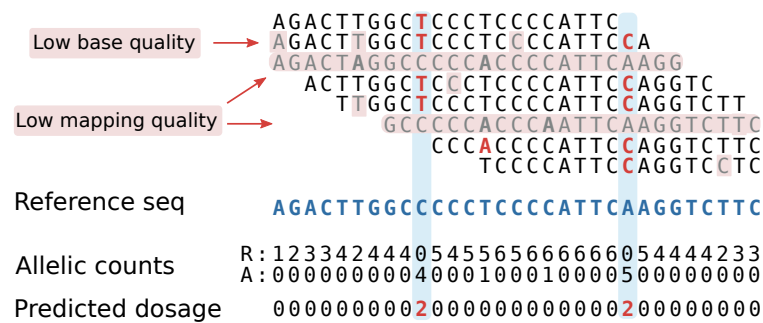
**Phred quality score**  
 $Q = -10 \log_{10} P_{err}$

Quality	Error probability	Accuracy
10 (Q10)	1 in 10	90%
20 (Q20)	1 in 100	99%
30 (Q30)	1 in 1000	99.9%
40 (Q40)	1 in 10000	99.99%



# Naive variant calling

Use fixed allele frequency threshold to determine the genotype



1) Filter base calls by quality  
e.g. ignore bases  $Q < 20$

2) Filter reads with low mapping quality

alt AF	genotype
[0, 0.2)	RR .. homozygous reference
[0.2, 0.8]	RA .. heterozygous
(0.8, 1]	AA .. homozygous variant

Problems:

- undercalls hets in low-coverage data
- throws away information due to hard quality thresholds
- gives no measure of confidence

More sophisticated models apply a statistical framework

$$P(G|D) = \frac{P(D|G) P(G)}{P(D)}$$

Labels: Likelihood (blue), Prior (orange), Posterior (green), Normalization (red)

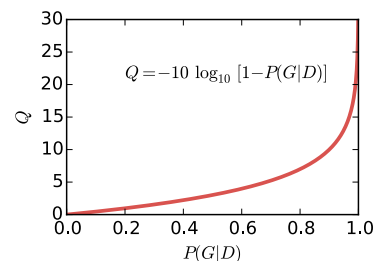
to determine:

1. the most likely genotype  $g \in \{RR, RA, AA\}$  given the observed data  $D$

$$g = \underset{G}{\operatorname{argmax}} P(G|D)$$

2. and the genotype quality

$$Q = -10 \log_{10}[1 - P(G|D)]$$



11 / 52

## Important terms you may encounter

### Genotype likelihoods

- which of the three genotypes RR, RA, AA is the data most consistent with?
- calculated from the alignments, the basis for calling
- takes into account:
  - base calling errors
  - mapping errors
  - statistical fluctuations of random sampling
  - local indel realignment (base alignment quality, BAQ)

### Prior probability

- how likely it is to encounter a variant base in the genome?
- some assumptions are made
  - allele frequencies are in Hardy-Weinberg equilibrium  
 $P(RA) = 2f(1 - f)$ ,  $P(RR) = (1 - f)^2$ ,  $P(AA) = f^2$
- can take into account genetic diversity in a population

$$P(G|D) = \frac{P(D|G) P(G)}{P(D)}$$

12 / 52



## Variant calling example

### Inputs

- alignment file
- reference sequence

### Outputs

- VCF or BCF file

### Example

```
bcftools mpileup -f ref.fa aln.bam | bcftools call -mv
```

### Tips

```
bcftools mpileup
```

- increase/decrease the required number (-m) and the fraction (-F) of supporting reads for indel calling
- the -Q option controls the minimum required base quality (30)
- BAQ realignment is applied by default and can be disabled with -B
- streaming the uncompressed binary BCF (-Ou) is much faster than the default text VCF

```
bcftools call
```

- decrease/increase the prior probability (-P) to decrease/increase sensitivity

### General advice

- take time to understand the options
- play with the parameters, see how the calls change

13 / 52

## Factors to consider in calling

Many calls are not real, a **filtering step is necessary**

False calls can have many causes

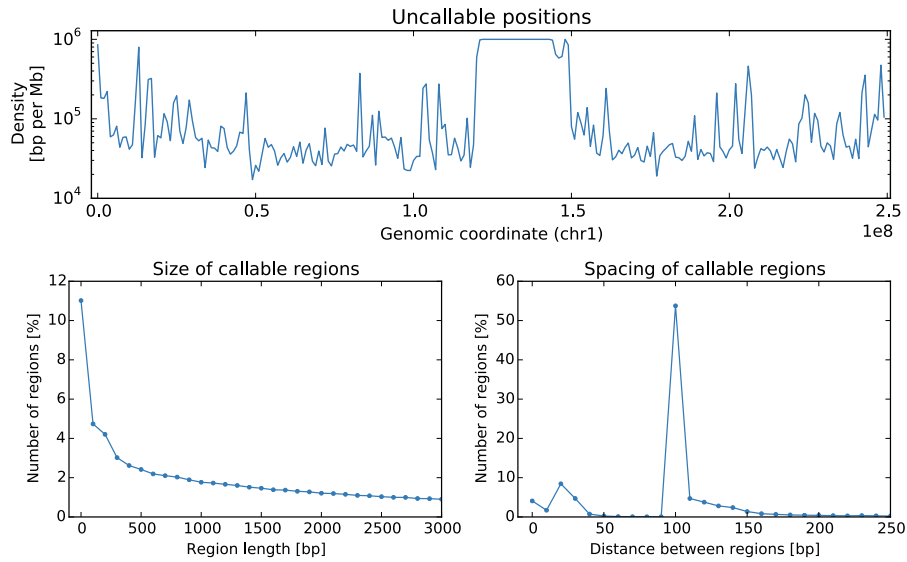
- contamination
- PCR errors
- sequencing errors
  - homopolymer runs
- mapping errors
  - repetitive sequence
  - structural variation
- alignment errors
  - false SNPs in proximity of indels
  - ambiguous indel alignment

14 / 52

# Callable genome

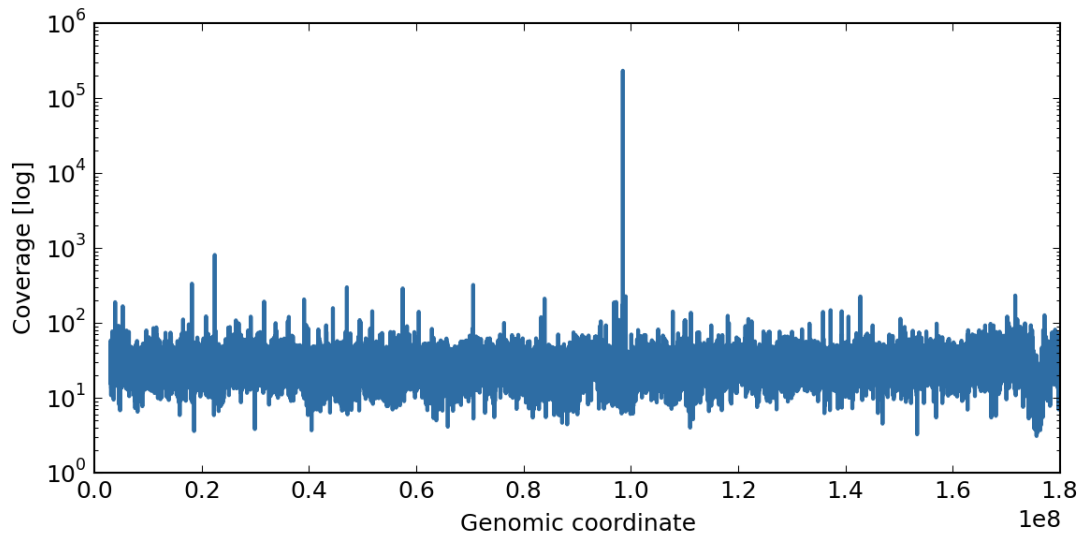
Large parts of the genome are still inaccessible

- the Genome in a Bottle high-confidence regions:
  - cover 89% of the reference genome
  - are short intervals scattered across the genome



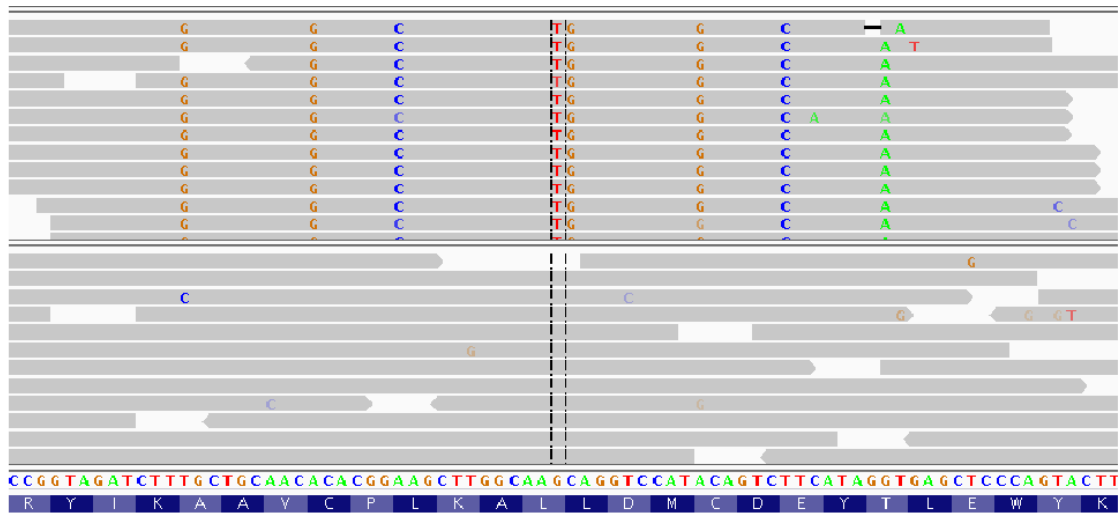
If possible, include only "nice" regions: for many analyses (e.g. population genetics studies) difficult regions can be ignored

# Maximum depth



Q: Why is the sequencing depth thousandfold the average in some regions?

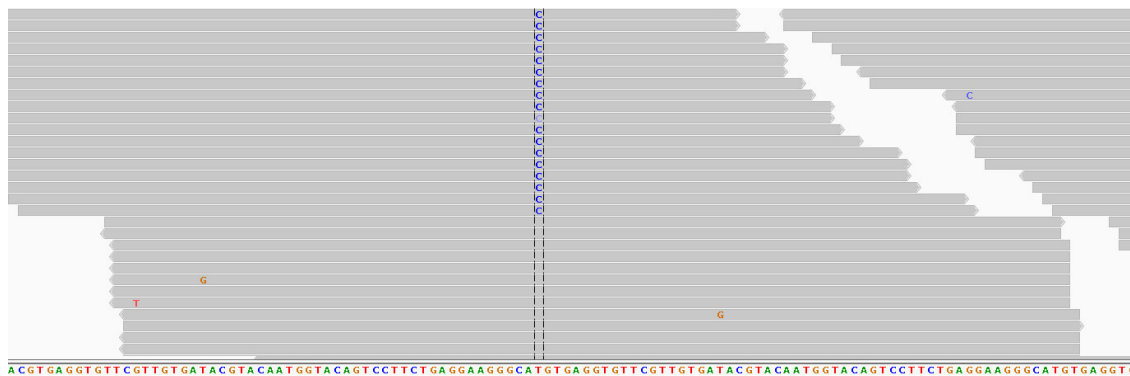
## Mapping errors



Q: RNA-seq (top) and DNA data (bottom) from the same sample has been mapped onto the reference genome. Can you explain the novel SNVs?

20 / 52

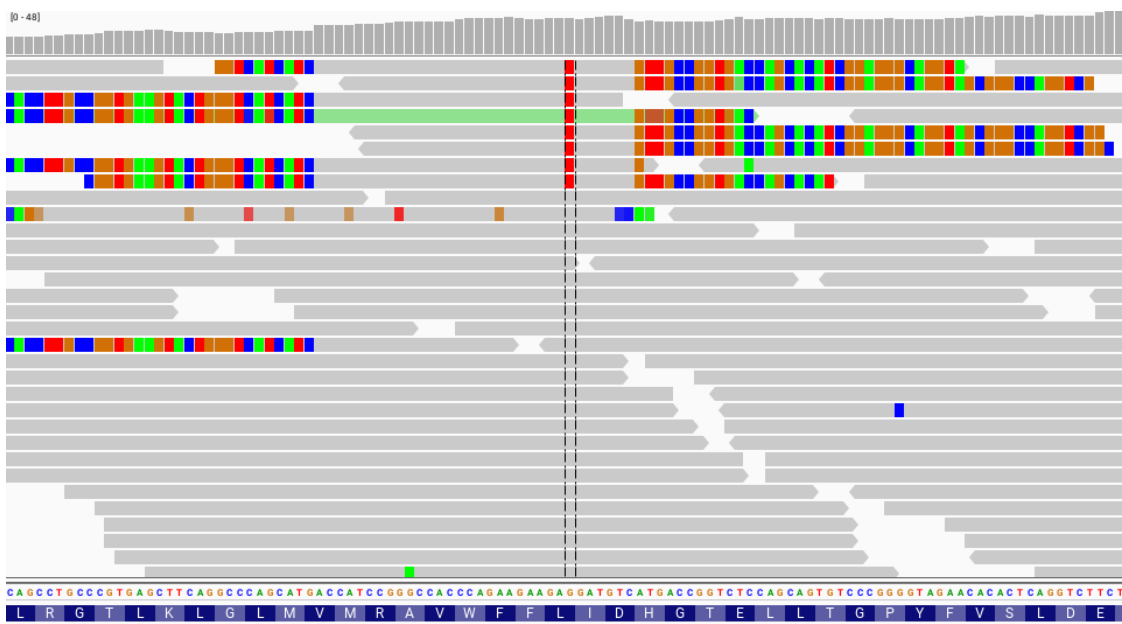
## Strand bias



Q: Is this a valid call?

23 / 52

## Change the display in IGV to reveal artefacts



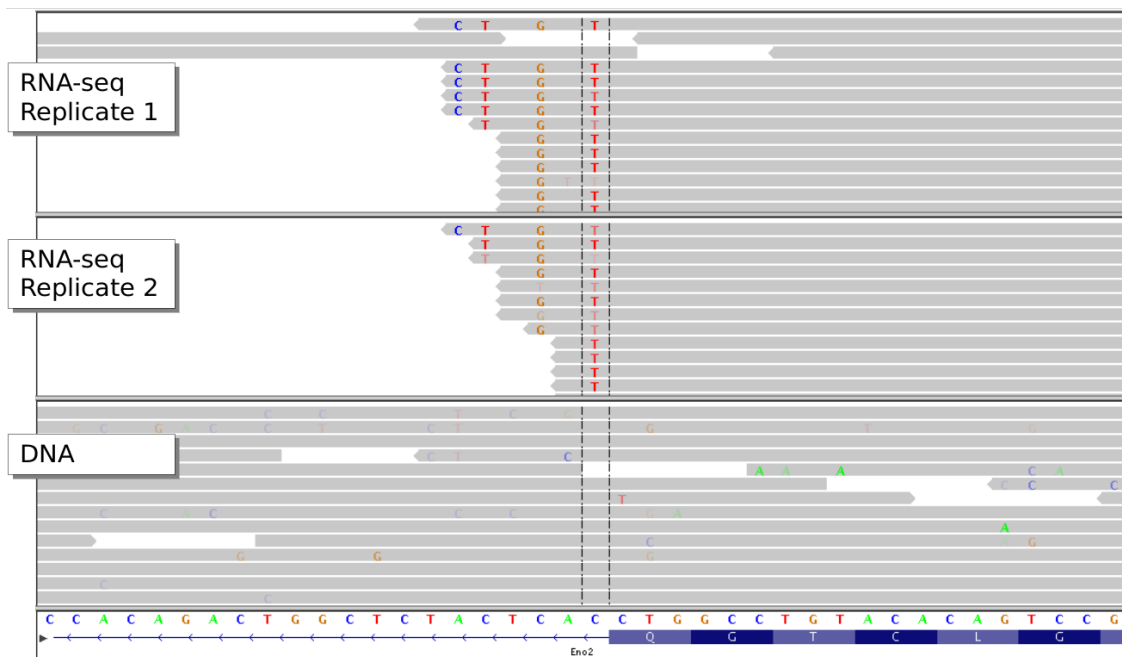
Display soft-clipped bases...



Too many soft-clipped reads in a region suggest mapping errors, beware!

29 / 52

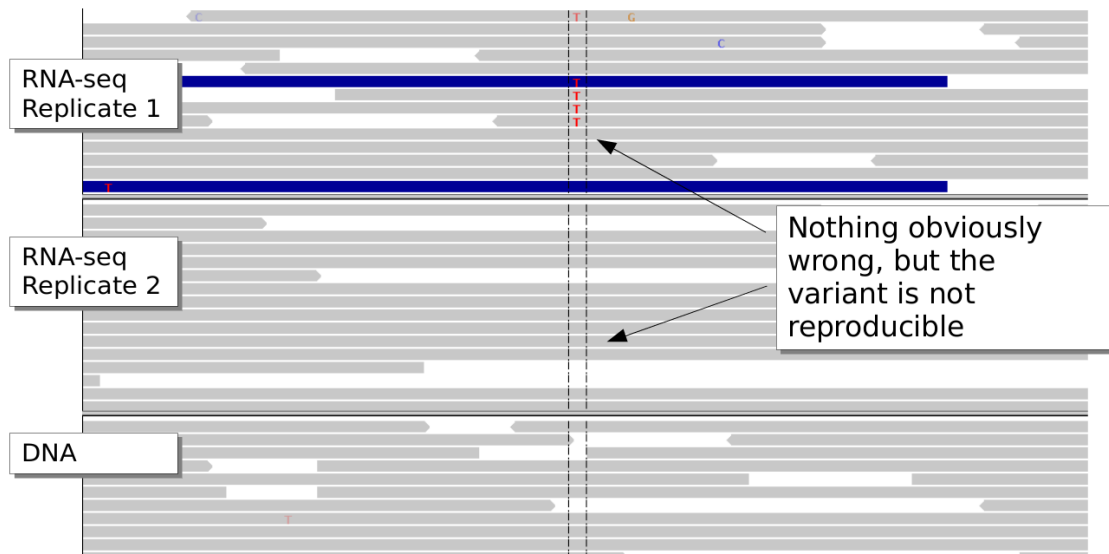
## Variant distance bias



Q: Can you explain what happened here?

30 / 52

# Reproducibility



Mind the biological variability. If possible, validate and replicate.

34 / 52

# False SNPs caused by incorrect alignment

Pairwise alignment artefacts can lead to false SNPs

- multiple sequence alignment is better, but very expensive
- instead: base alignment quality (BAQ) to lower quality of misaligned bases

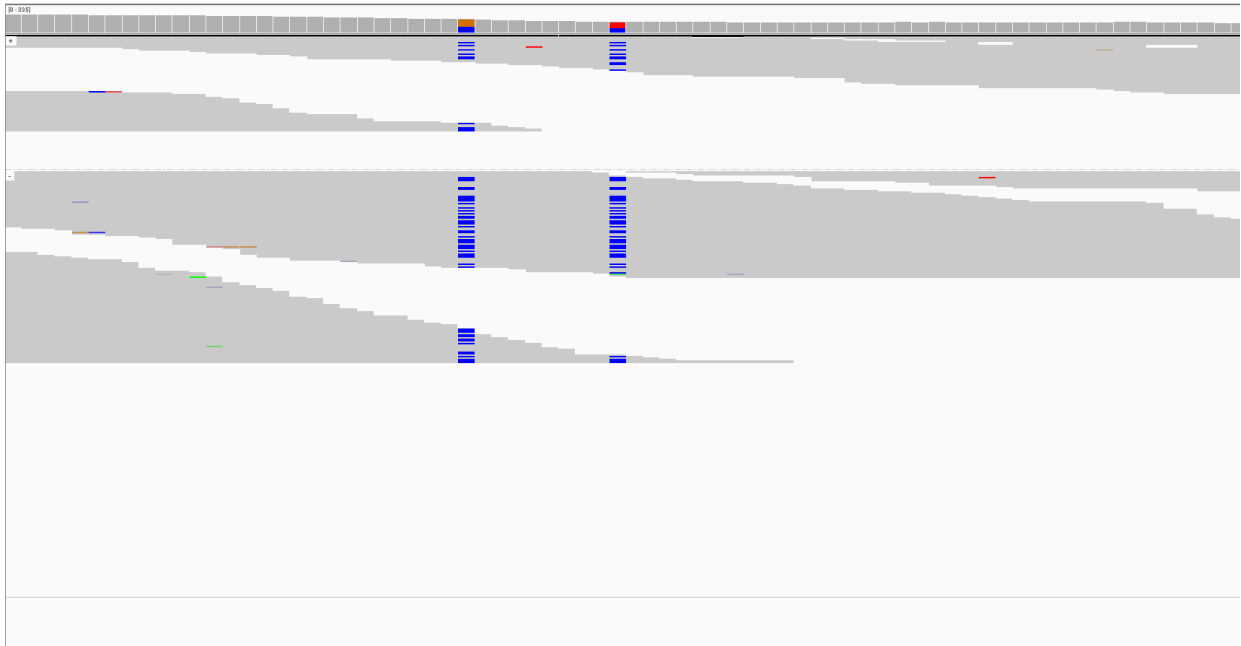
```

Aligned reads
  /
 / aggtttataaac----aaataa
 / gggtttataaac----aaataatt
 / ttataaaacaaataattaagtctaca
 / caaat----aattaagtctacagagcaac
 / aat----aattaagtctacagagcaact
 / t----aattaagtctacagagcaacta
Reference seq aggtttataaac----aattaagtctacagagcaacta
  
```

Q: How many SNPs are real?

35 / 52

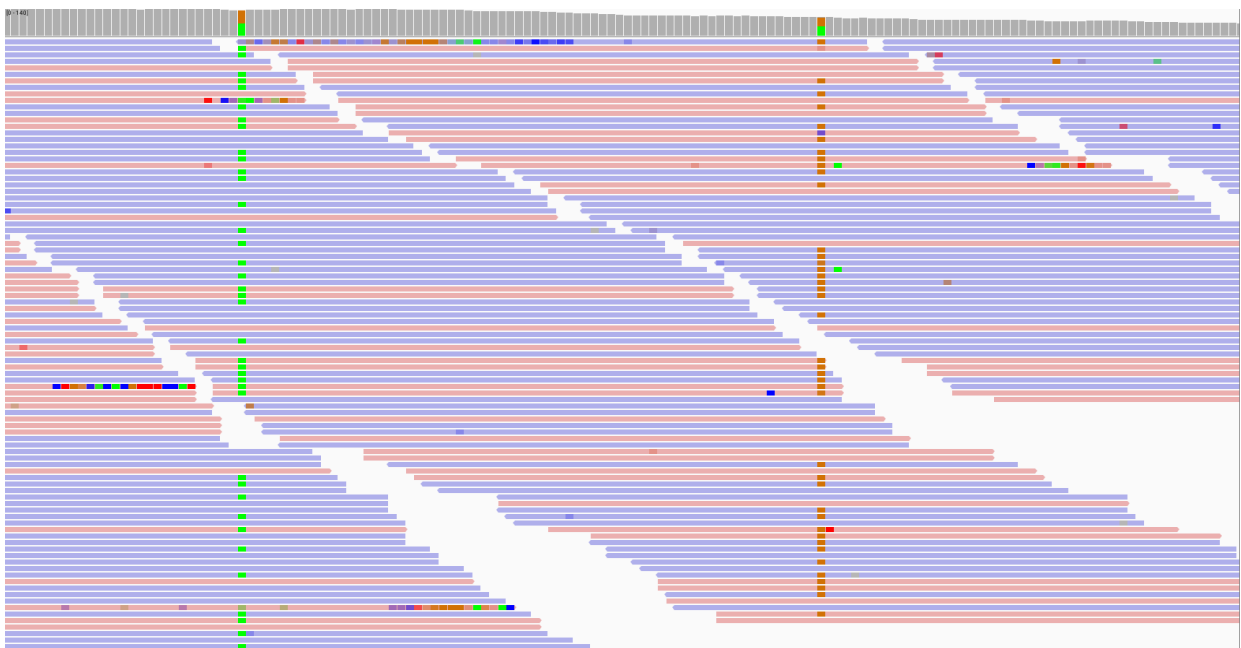
## What good SNPs look like?



Change the view IGV to inspect possible biases. Here the reads were squished and grouped by read strand to confirm two clean unbiased calls.

39 / 52

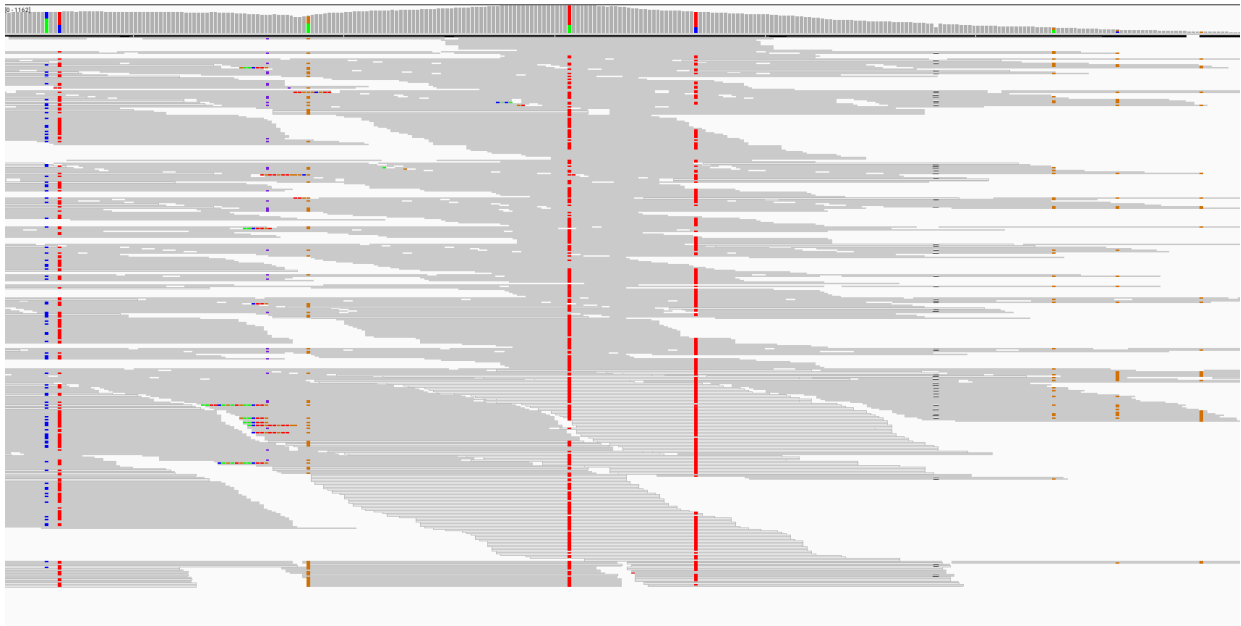
## What good SNPs look like?



Change the view IGV to inspect possible biases. Here the reads were colored by read strand to confirm another two clean unbiased calls.

41 / 52

# What good SNPs look like?



Q: Is this call real? There are many reads with MQ=0.

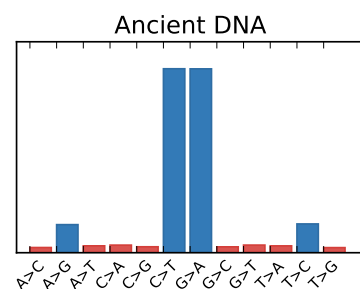
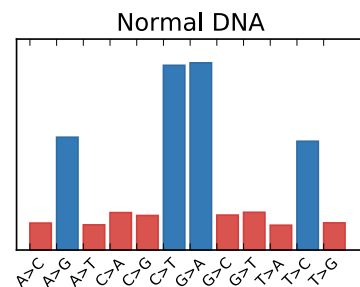
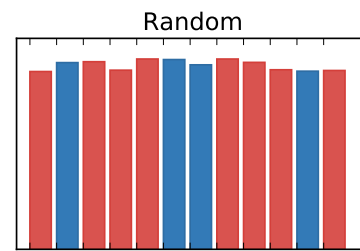
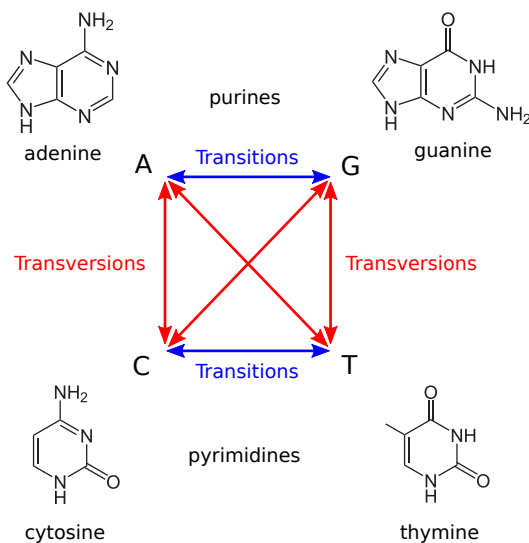


Sorting the reads by MQ reveals the variant is also supported by many high-quality reads.

# How to estimate the quality of called SNPs?

Transitions vs transversions ratio, known as ts/tv

- transitions are 2-3x more likely than transversions



## Indel calling challenges

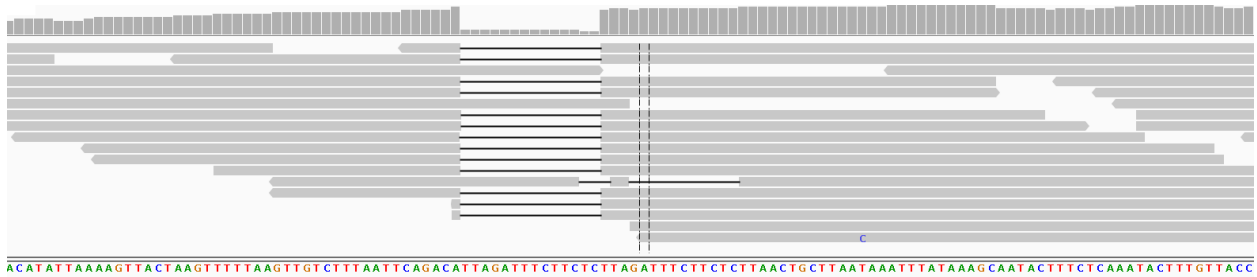
The sequencing error rate is elevated in microsatellites

Low reproducibility across callers

- 37.1% agreement between HapCaller, SOAPindel and Scalpel  
Narzisi et al. (2014) Nat Methods, 11(10):1033

Reads with indels are more difficult to map and align

- the aligner can prefer multiple mismatches rather than a gap
- indel representation can be ambiguous



```
CTTTAATTCAGACATTAGATTTCTTCTC
CTTTAATTCAGACATTAGATTTCTTCTTA
CTTTAATTCAGACA-----TTAGATTTCTTCTCTTAACTGCTT
CTTTAATTCAGACATTAGATTTCTTC--TA-----TTAACTGCTT

CTTTAATTCAGACATTAGATTTCTTCTCTTAGATTTCTTCTCTTAACTGCTT
```

48 / 52

## Future of variant calling

Current approaches

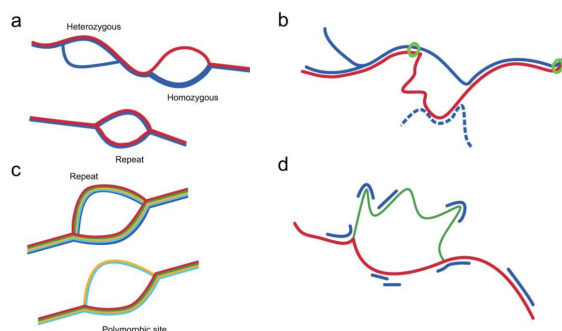
- rely heavily on the supplied alignment, but aligners see one read at a time
- largely site based, do not examine local haplotype and linked sites

Local *de novo* assembly based variant callers

- call SNPs, indels, MNPs and small SV simultaneously
- can remove alignment artefacts
- eg GATK haplotype caller, Scalpel, Octopus

Variation graphs

- align to a graph rather than a linear sequence



Iqbal et al. (2012) Nat Gen 44(2):226

50 / 52



# Single vs multi-sample and gVCF calling

VCF files can be **very** big, therefore we often store only variant sites<sup>1</sup>

- however, variant-only VCFs are difficult to compare - was a site dropped because of a reference call or because of low coverage?
- we need evidence for both variant and non-variant positions in the genome

gVCF

- represents blocks of reference-only calls in a single record using the END tag
- symbolic allele in raw "callable" gVCFs allows to calculate genotype likelihoods only once (an expensive step), then do calling repeatedly as more samples come in

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	Sample
19	9902	.	G	<*>	.	.	END=9915;MinDP=0	PL:DP	0,0,0:0
19	9916	.	C	<*>	.	.	END=9922;MinDP=5	PL:DP	0,15,137:5
19	9923	.	G	<*>	.	.	END=9948;MinDP=10	PL:DP	0,30,214:10
19	9949	.	G	A,<*>	.	.	DP=28	PL:DP	0,60,255,78,255,255:27
19	9950	.	C	<*>	.	.	END=9958;MinDP=28	PL:DP	0,84,255:28
19	9959	.	G	T,<*>	.	.	DP=34	PL:DP	0,82,255,99,255,255:34
19	9960	.	C	<*>	.	.	END=9969;MinDP=34	PL:DP	0,102,255:34

Symbolic "unobserved" allele  
 Represents any other possible alternate allele

A block of 10 sites with at least 34 reference reads

Genotype likelihoods for CC, C\*, \*\*

<sup>1</sup>Annotated VCF with 3,781 samples, variant sites only, UK10k project . . . 680GB

# Functional annotation

VCF can store arbitrary INFO tags (per site) and FORMAT tags (per sample)

- describe genomic context of the variant (e.g. coding, intronic, UTR)
- predict functional consequence (e.g. synonymous, missense, start lost)

Several tools for annotating a VCF, only few are haplotype-aware

BCFtools/csq <http://github.com/samtools/bcftools>

VEP Haplosaurus <http://github.com/willmclaren/ensembl-vep>

A)

	R	Q	R	L	L										
	C	G	T	C	A	G	T	A	G	C	T	G	C	T	G
	C	G	T	C	A	G	T	A	G	C	T	G	C	T	G
✗	R	Q	W	L	L										
✗	C	G	T	C	A	G	T	A	G	C	T	G	C	T	G
✗	R	Q	Q	L	L										
✗	C	G	T	C	A	G	T	A	G	C	T	G	C	T	G
✓	R	Q	*	L	L										
✓	C	G	T	C	A	G	T	A	G	C	T	G	C	T	G

B)

	S	S	S	P	P	P	P	P	L	Q	F	H																						
	T	C	A	T	C	C	T	C	C	C	C	T	C	C	T	C	T	C	T	G	C	A	G	T	T	T	C	A	T					
	T	C	-	-	C	C	T	C	-	-	-	-	-	-	C	C	C	C	T	C	C	T	C	T	G	C	A	G	T	T	T	C	A	T
✗	S	-	-	L	P	S	P	S	S	S	S	A	V	S																				
✗	T	C	A	T	C	C	T	C	-	-	-	-	-	-	C	C	C	C	T	C	C	T	C	T	G	C	A	G	T	T	T	C	A	T
✓	S	-	-	L	-	-	-	-	-	-	-	-	-	-	P	P	P	P	P	L	Q	F	H											
✓	T	C	-	-	C	C	T	C	-	-	-	-	-	-	C	C	C	C	T	C	C	T	C	T	G	C	A	G	T	T	T	C	A	T

C)

	D	G	T	Q	P	G	H														
	G	A	T	G	G	A	A	C	C	C	A	G	C	C	T	G	G	C	A	C	G
	G	A	T	G	G	A	A	C	C	C	A	G	C	C	T	G	G	C	A	C	A
✗	D	G	T	Q	P	G	H	A													
✗	G	A	T	G	G	A	A	C	C	C	A	G	C	C	T	G	G	C	A	C	A
✓	D	G	T	Q	P	G	H														
✓	G	A	T	G	G	A	A	C	C	C	A	G	C	C	T	G	G	C	A	C	A

intron

	D	I	S	D	S	L	T													
	A	T	A	T	T	T	C	A	G	A	T	T	C	G	C	T	A	A	C	A
	A	G	A	T	T	T	C	A	G	A	T	T	C	G	C	T	A	A	C	A
✗	N	I	S	D	S	L	T													
✗	A	T	A	T	T	T	C	A	G	A	T	T	C	G	C	T	A	A	C	A
✓	K	I	S	D	S	L	T													
✓	A	G	A	T	T	T	C	A	G	A	T	T	C	G	C	T	A	A	C	A

## Practical exercises: Variant calling

An online version of this document can be found here <https://tinyurl.com/yb5ak2xb>. Please feel free to add comments if anything is unclear or incorrect. The answers to the exercises can be found at the end of this document.

### Exercise 1: Making sense of the input data

We will need the aligned sequences in SAM, BAM or CRAM format and the reference genome in fasta format. To list the files in the current directory, type

```
ls -lh
```

The listing shows two mouse data fragments, strains A/J and NZO, and the chromosome 19 of the mouse reference genome.

Before variant calling, it is important to check the data we'll be working with. Using the command below, get some statistics from the bams

```
samtools stats -r GRCm38_68.19.fa A_J.bam > A_J.stats
samtools stats -r GRCm38_68.19.fa NZO.bam > NZO.stats
plot-bamstats -r GRCm38_68.19.fa.gc -p A_J.graphs/ A_J.stats
plot-bamstats -r GRCm38_68.19.fa.gc -p NZO.graphs/ NZO.stats
```

**1.1** Open the plots `A_J.graphs/index.html` and `NZO.graphs/index.html` in your firefox browser (e.g., `firefox A_J.graphs/index.html`). What is the percentage of mapped reads in both strains? Check the insert size, GC content, per-base sequence content and quality per cycle graphs. Do all look reasonable?

### Exercise 2: Generating pileup

The command `samtools mpileup` prints the read bases that align to each position in the reference genome. On the command line, try this

```
samtools mpileup -f GRCm38_68.19.fa A_J.bam | less -S
```

Each line corresponds to a position on the genome. The columns are: chromosome, position, reference base, read depth, read bases (dot `.` and comma `,` indicate match on the forward and on the reverse strand; `ACGTN` and `acgtn` a mismatch on the forward and the reverse strand) and the final column is the base qualities encoded into characters. The caret symbol `^` marks the start of a read, the dollar sign `$` the end of a read, deleted bases are represented by asterisk `*`.

**2.1** What is the read depth at position `10001994`? (Rather than scrolling to the position, use the substring searching capabilities of `less`: press `/`, then type `10001994` followed by `enter` to find the position.)

**2.2** What is the reference and the alternate base at the position? How many reference and how many non-reference bases there are?

This output can be used for a simple consensus calling. One rarely needs this type of output. Instead, for a more sophisticated variant calling method, go to the next section.

### Exercise 3: Generating genotype likelihoods and calling variants

The `bcftools mpileup` command can be used to generate genotype likelihoods. (Beware: the command `mpileup` is present in both `samtools` and `bcftools`, but in both they do different things. While `samtools mpileup` produces the text pileup output from the previous exercise, `bcftools mpileup` generates a VCF with genotype likelihoods.)

Run the following command (when done, press `q` to quit the viewing mode):

```
bcftools mpileup -f GRCm38_68.19.fa A_J.bam | less -S
```

This generates an intermediate output which contains genotype likelihoods and other raw information necessary for variant calling. This output is usually streamed directly to the caller like this

```
bcftools mpileup -f GRCm38_68.19.fa A_J.bam | bcftools call -m | less -S
```

**3.1** The output above contains both variant and non-variant positions. Check the `Input/output options` section of the `bcftools call` usage page and see if there is an option to print out only variant sites.

The `INFO` and `FORMAT` fields of each entry tells us something about the data at the position in the genome. It consists of a set of key-value pairs with the tags being explained in the header of the VCF file (see the `##INFO` and `##FORMAT` lines in the header).

Let `mpileup` output more information. For example, we can ask it to add the `FORMAT/AD` tag which informs about the number of high-quality reads that support alleles listed in REF and ALT columns. The list of all available tags can be printed with the command

```
bcftools mpileup -a ?
```

Now let's run the variant calling again, this time adding the `-a AD` option. We will also add the `-Ou` option so that it streams a binary uncompressed BCF into call. This is to avoid the unnecessary CPU overhead of formatting the internal binary format to plain text VCF only to be immediately formatted back to the internal binary format again

```
bcftools mpileup -a AD -f GRCm38_68.19.fa A_J.bam -Ou | bcftools call -mv -o out.vcf
```

Examine the VCF file output using the unix command `less`

```
less -S out.vcf
```

**3.2** What is the reference and the alternate base at the position 10001994?

**3.3** What is the total read depth at the position 10001994? Note that this number may be different from the values we obtained earlier, because some low quality reads or bases might have been filtered.

**3.4** What is the number of high-quality reads supporting the SNP call at position 10001994? How many reads support the reference allele and how many support the alternate allele? Look up the `AD` tag in the `FORMAT` column: the first value gives the number of reference reads and the second gives the number of non-reference reads.

**3.5** What sort of event is at position 10003649?

## Exercise 4: Variant filtering

In the series of commands we will learn how to extract information from VCFs and how to filter the raw calls. We will use the `bcftools` commands again. Most of the commands accept the `-i`, `--include` and

`-e`, `--exclude` options (<http://samtools.github.io/bcftools/bcftools.html#expressions>) which will come handy when filtering using fixed thresholds. We will estimate the quality of the callset by calculating the ratio of transitions and transversions (<http://en.wikipedia.org/wiki/Transversion>).

When drafting commands, it is best to build them gradually. This prevents errors and allows to verify that they work as expected. Let's start with printing a simple list of positions from the VCF using the `bcftools query` command (<http://samtools.github.io/bcftools/bcftools.html#query>) and pipe through the `head` command to limit the printed output to the first few lines:

```
bcftools query --format 'POS=%POS\n' out.vcf | head
```

As you could see, the command expanded the formatting expression `POS=%POS\n` in the following way: for each VCF record the string `POS=` was copied verbatim, the string `%POS` was replaced by the VCF coordinate stored in the `POS` column, and then the newline character `\n` ended each line. (Without the newline character, positions from the entire VCF would be printed on a single line.)

Now add the reference and the alternate allele to the output. They are stored in the `REF` and `ALT` column in the VCF, and let's separate them by a comma:

```
bcftools query -f'%POS %REF,%ALT\n' out.vcf | head
```

In the next step add also the quality (`%QUAL`), genotype (`%GT`) and sequencing depth (`%AD`) to the output. Note that `FORMAT` tags must be enclosed within square brackets `[...]` to iterate over all samples in the VCF. (Check the [Extracting per-sample tags](http://samtools.github.io/bcftools/howtos/query.html) section in the manual <http://samtools.github.io/bcftools/howtos/query.html> for a more detailed explanation why the square brackets are needed.)

```
bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' out.vcf | head
```

Now we are able to quickly extract important information from the VCFs. Now let's filter rows with `QUAL` smaller than 30 by adding the filtering expression `--exclude 'QUAL<30'` or `--include 'QUAL>=30'` like this:

```
bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' -i'QUAL>=30' out.vcf | head
```

Now compare the result with the output from the previous command, were the low-quality lines removed? In the next step limit the output to SNPs and ignore indels by adding the `type="snp"` condition to the filtering expression. Because both conditions must be valid at the same type, we request the `AND` logic using the `&&` operator:

```
bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' -i'QUAL>=30 && type="snp"' out.vcf | head
```

**4.1** Can you print SNPs with `QUAL` bigger than 30 and require at least 25 alternate reads in the `AD` tag?

Remember, the first value of the `AD` tag is the number of reference reads, the second is the number of alternate reads, therefore you will need to query the second value of the `AD` tag. The first value can be queried as `AD[0]` and the second as `AD[1]` (the allele indexes are zero-based). Thus add to the expression the condition `AD[1] >= 25`.

Now we are able to filter our callset. In order to evaluate the quality, we will use `bcftools stats` to calculate the ratio of transitions vs transversions. We start by checking first what is the `ts/tv` of the raw unfiltered callset. The `stats` command produces a text output, we extract the field of interest as follows:

```
bcftools stats out.vcf | less
bcftools stats out.vcf | grep TSTV
bcftools stats out.vcf | grep TSTV | cut -f5
```

**4.2** Calculate ts/tv of the set filtered as above by adding `-i 'QUAL>=30 && AD[*:1]>=25'` to the `bcftools stats` command. (Here the asterisk followed by a colon tells the program to apply the filtering to all samples. At least one sample must pass in order for a site to pass.) After applying the filter, you should observe an increased ts/tv value.

**4.3** Can you do the reverse and find out the ts/tv of the **removed** sites? Use the `-e` option instead of `-i`. The ts/tv of the removed low-quality sites should be lower.

**4.4** The test data come from an inbred homozygous mouse, therefore any heterozygous genotypes are most likely mapping and alignment artefacts. Can you find out what is the ts/tv of the heterozygous SNPs? Do you expect higher or lower ts/tv? Use the filtering expression `-i 'GT="het"'` to select sites with heterozygous genotypes.

Another useful command is `bcftools filter` which allows to "soft filter" the VCF: instead of removing sites, it can annotate the `FILTER` column to indicate sites which fail. Apply the above filters to produce a final callset, adding also the `--SnpGap` and the `--IndelGap` option to filter variants in close proximity to indels:

```
bcftools filter -s LowQual -i'QUAL>=30 && AD[*:1]>=25' -g8 -G10 out.vcf -o out.flt.vcf
```

### Exercise 5: Variant normalization

The same indel variant can be represented in different ways. For example, consider the following 2bp deletion. Although the resulting sequence does not change, the deletion can be placed at two different positions within the short repeat:

```
      12345
      TTCTC
POS=1  T--TC
POS=3  TTC--
```

In order to be able to compare indels between two datasets, we left-align such variants.

**5.1** Use the `bcftools norm` command to normalize the filtered callset. Note that you will need to provide the `--fasta-ref` option. Check in the output how many indels were realigned.

### Exercise 6: Multi-sample variant calling

In many types of experiments we want to sequence multiple samples and compare their genetic variation. The single-sample variant calling we have done so far has the disadvantage of not providing information about reference genotypes. Because only variant sites are stored, we are not able to distinguish between records missing due to reference genotypes versus records missing due to lack of coverage.

**6.1** Type `ls *bam` to check that there are two BAM files in the directory. Can you modify the calling command from **Exercise 3** to use both BAM files? Write the output to a BCF file called `multi.bcf` and index the file afterwards.

**6.2** Apply the same filters as before and write the output to a BCF file called `multi.filt.bcf`, then index the file.

**6.3** What is the ts/tv of the raw calls and of the filtered set?

## 6.4 What is the ts/tv of the removed sites?

### Exercise 7: Data visualization

It is often useful to visually inspect a SNP or indel of interest in order to assess the quality of the SNP call and interpret the genomic context of the SNP. We can use the **IGV** tool to view some of the SNPs positions from the VCF file. On the command line, type:

```
sh bin/IGV_2.3.32/igv.sh
```

Set the reference genomes to be mouse (**mm10/GRCm38**) by clicking on the **Genomes** then **Load Genomes from Server** buttons and select **mm10**. On the file menu down the left, select **Load from File** and select the BAM file **A\_J.bam**

In the top bar, enter the position **chr19:10,001,874-10,002,017** to view the SNP bases at position **10001946**.

**7.1** How many forward aligned reads support the SNP call? How many forward aligned reads support the SNP? How many reverse reads support the SNP? Note: hover the mouse pointer over the coverage bar at the top to get this information.

**7.2** Was this SNP called also by **bcftools**? Did it pass the filters?

**7.3** In the top bar, enter the position 'chr19:10072443' to view the SNP at position 10072443. Was the SNP also called by **bcftools**? Did it pass the filters? Does this look like a real SNP? Wh?

### Exercise 8: Functional consequences

There are several popular programs available for predicting functional consequences. Here we will use the lightweight **bcftools csq** command. It is haplotype-aware and can correctly predict consequences for frame-restoring and other compound variants such as MNPs. On input it requires a VCF, the fasta reference file and a GFF file with the gene model. Because our data is not phased, we will provide the **--phase** option (which does not actually phase the data, but tells the program to make an assumption about the phase):

```
bcftools view -i 'FILTER="PASS"' multi.filt.bcf | bcftools csq -p m -f  
GRCm38_68.19.fa -g Mus_musculus.part.gff3.gz -Ob -o multi.filt.annot.bcf
```

```
bcftools index multi.filt.annot.bcf
```

**8.1** The **bcftools csq** command annotated the VCF with a new INFO tag **BCSQ**. Use the **bcftools query -f '%BCSQ\n'** command to extract the consequence at position **19:10088937**. What is the functional annotation at this site? What is the amino acid change?

### Answers to exercises:

**1.1** You will find there is a consistent GC bias. However, in our case this is OK because the stats was produced from a small BAM fragment only, with a locally different GC content.

**2.1** 66 reads

**2.2** The reference allele is **A** and the alternate allele is **G**. The upper/lower case letters indicate the forward/reverse orientation of the read.

**3.1** Add the **-v** option to the command:

```
bcftools mpileup -f GRCm38_68.19.fa A_J.bam | bcftools call -mv | less -S
```

**3.2** Look up the REF and ALT columns: the reference base is **A**, the alternate allele is **G**.

**3.3** Look up the tag **DP** in the INFO column: there were 69 raw reads at the position.

**3.4** There are 0 reference and 66 alternate high-quality reads.

**3.5** Five bases TGTGG were inserted after the T at position 10003649

**4.1** The complete command is

```
bcftools query -f'%POS %QUAL [%GT %AD] %REF %ALT\n' -i'QUAL>=30 && type="snp" && AD[1]>=25' out.vcf | head
```

**4.2** The complete command is

```
bcftools stats -i'QUAL>=30 && AD[*:1]>=25' out.vcf | grep TSTV | cut -f5
```

**4.3** The complete command is

```
bcftools stats -e'QUAL>=30 && AD[*:1]>=25' out.vcf | grep TSTV | cut -f5
```

**4.4** The complete command is

```
bcftools stats -i 'GT="het"' out.vcf | grep TSTV | cut -f5
```

**5.1** The complete command is

```
bcftools norm -f GRCm38_68.19.fa out.flt.vcf -o out.flt.norm.vcf
```

**6.1** Use the commands

```
bcftools mpileup -a AD -f GRCm38_68.19.fa *.bam -Ou | bcftools call -mv -Ob -o multi.bcf
bcftools index multi.bcf
```

**6.2** Use the commands

```
bcftools filter -s LowQual -i'QUAL>=30 && AD[*:1]>=25' -g8 -G10 multi.bcf -Ob -o multi.filt.bcf
bcftools index multi.filt.bcf
```

**6.3** Use the commands

```
bcftools stats multi.filt.bcf | grep TSTV | cut -f5
bcftools stats -i 'FILTER="PASS"' multi.filt.bcf | grep TSTV | cut -f5
```

**6.4** Use the command

```
bcftools stats -e 'FILTER="PASS"' multi.filt.bcf | grep TSTV | cut -f5
```

**7.1** 75 reads total, 39 on the forward and 36 on the reverse strand.

**7.2** Use the command

```
bcftools view -H -r 19:10001946 multi.filt.bcf
```

**7.3** It is an alignment artefact, the aligner preferred two SNPs instead of a long deletion. The call was removed by filtering only because we excluded calls in close proximity of indels:

```
bcftools view -H -r 19:10072443 multi.filt.bcf
```

**8.1** The C>T mutation changes the amino acid at position 163 in the protein sequence, from valin to isoleucin.

## Public data archives for NGS data

Jacqui Keane

jm15@sanger.ac.uk



---

---

---

---

---

---

---

---

## Purpose of data archives

- ▶ For archiving and distribution of data generated by NGS experiments
- ▶ Submit your own data that you want to publish
- ▶ Finding data sets that might be relevant to your own research
- ▶ Retrieve data sets from publication
- ▶ Many different data archives for different data types

---

---

---

---

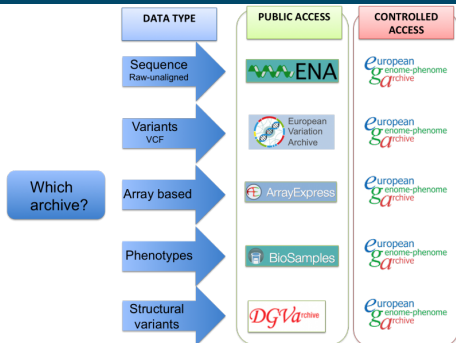
---

---

---

---

## Which data archive?



---

---

---

---

---

---

---

---



### Data sharing across archives

A world map with red arrows indicating data sharing between NCBI (North America), EBI (Europe), and DDBJ (Japan).

Logos for NCBI (National Center for Biotechnology Information), EMBL-EBI, and DDBJ (DNA Data Bank of Japan) are shown below the map. The Sanger logo is in the bottom right corner.

WT NGS Bioinformatics

---

---

---

---

---

---

---

---

### Global data archives

Data Type	DDBJ	EBI	NCBI
Primary Sequence Data	DDBJ Sequence Read Archive (DRA)	European Nucleotide Archive (ENA)	Sequence Read Archive (SRA)
Annotated Sequences	DDBJ		GenBank
Variation	-	European Variation Archive (EVA)	dbSNP
Structural Variation	-	Genomic Variants Archive (DGVa)	dbVar
Expression	DDBJ Omics Archive (DOR)	ArrayExpress	Gene Expression Omnibus (GEO)
Restricted	Japanese Genome-phenome Archive (JGA)	European Genome-phenome Archive (EGA)	dbGAP
Samples	BioSample	BioSample	BioSample
Studies	BioProject	BioProject	BioProject

WT NGS Bioinformatics

Sanger logo

---

---

---

---

---

---

---

---

### European Nucleotide Archive (ENA)

► For data from experiments based on nucleotide sequencing

The diagram shows three components: ENA-Annotation (Feature annotation) with a sequence and feature map; ENA-Assembly (Assembly information) with a contig assembly graph; and ENA-Reads (Sequencing and sampling information) with a sequencing run visualization.

WT NGS Bioinformatics

Sanger logo

---

---

---

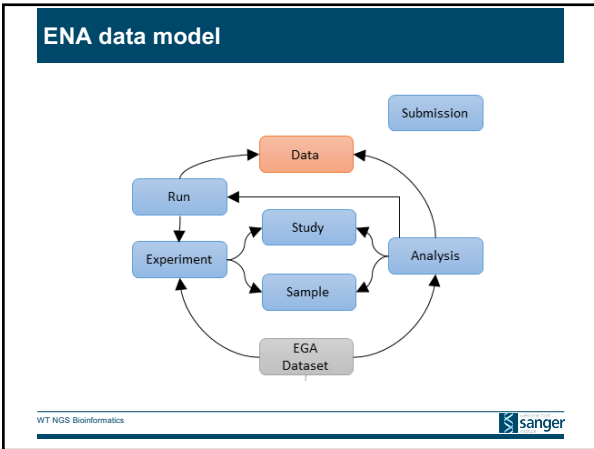
---

---

---

---

---




---

---

---

---

---

---

---

---

### ENA accessions

Type	Accession	Description
Study	ERP/PRJE	Information about the sequencing study
Sample	ERS/SAME	Information about the samples sequenced
Experiment	ERX	Information about sequencing experiment including platform used and library information
Read	ERR	Raw data files containing sequence data (CRAM, BAM, Fastq)
Analysis	ERZ	Secondary analysis results computed from the primary sequencing reads (BAM, EMBL)
Annotated Sequence	LN CWSE	Assembled and annotated sequence, one number for each sequence e.g. CWSE01000001-CWSE01000051

WT NGS Bioinformatics

---

---

---

---

---

---

---

---

### ENA accessions

Type	Accession	Description
Study	ERP/PRJE	Information about the sequencing study
Sample	<b>ERS/SAME</b>	Information about the samples sequenced
Experiment	ERX	Information about sequencing experiment including platform used and library information
Read	<b>ERR</b>	Raw data files containing sequence data (CRAM, BAM, Fastq)
Analysis	ERZ	Secondary analysis results computed from the primary sequencing reads (BAM, EMBL)
Annotated Sequence	<b>LN</b> <b>CWSE</b>	Assembled and annotated sequence, one number for each sequence e.g. CWSE01000001-CWSE01000051

WT NGS Bioinformatics

---

---

---

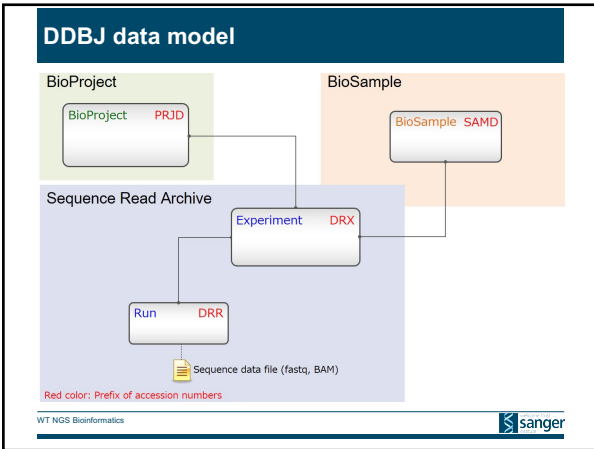
---

---

---

---

---



---

---

---

---

---

---

---

---

### ENA data submission

The screenshot shows the ENA data submission website interface with the following elements:

- Header: ENA European Nucleotide Archive
- Navigation: Home, New Submission, Studies, Sample Groups, Samples, Experiments, Runs, Assemblies, Variants
- Start section: You can use this service to submit sequence reads, genome assemblies and variations and to register studies (projects) and samples. To submit sequences other than reads or assemblies please use [SRA-Short Reads](#). To register variants studies, used to group together sequencing studies, please contact [genbank@ebi.ac.uk](mailto:genbank@ebi.ac.uk). Please select the type of submission you would like to make:
- Submit sequence reads and experiments
- Register study (project)
- Register samples
- Submit assemblies
- Submit variations

WT NGS Bioinformatics

---

---

---

---

---

---

---

---

### Browsing ENA

- ▶ Let's browse at
  - ▶ <http://www.ebi.ac.uk/ena>
  - ▶ PRJEB6352

WT NGS Bioinformatics

---

---

---

---

---

---

---

---

### European Variation Archive (EVA)

- ▶ For genetic variation data from all species
- ▶ Data submission
  - ▶ Same infrastructure as ENA
  - ▶ Consists of VCF file(s) and metadata that describes sample(s), experiment (s), and analysis that produced the variants
  - ▶ Accessions are ERZ
- ▶ NCBI equivalent is dbSNP

WT NGS Bioinformatics



---

---

---

---

---

---

---

---

### Browsing EVA

- ▶ Let's browse at
  - ▶ <http://www.ebi.ac.uk/eva/?Study%20Browser&browserType=sgv>

WT NGS Bioinformatics



---

---

---

---

---

---

---

---

### Array Express

- ▶ For functional genomics data from array and sequencing based experiments (RNA-Seq, CHIP-Seq)
  - ▶ raw e.g. Affymetrix CEL files, fastq files
  - ▶ processed e.g. aligned bam, txt files of read counts
- ▶ Data submission is via 'Annotare' web interface
- ▶ NCBI equivalent is GEO

WT NGS Bioinformatics



---

---

---

---

---

---

---

---

### Browsing ArrayExpress

- ▶ Let's browse at
  - ▶ <https://www.ebi.ac.uk/arrayexpress/browse.html>

WT NGS Bioinformatics



---

---

---

---

---

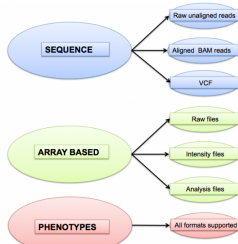
---

---

---

### European Genome-phenome Archive (EGA)

- ▶ For personally identifiable genetic and phenotypic data
- ▶ Individuals whose consent agreements authorise that data is release for specific research use only



WT NGS Bioinformatics



---

---

---

---

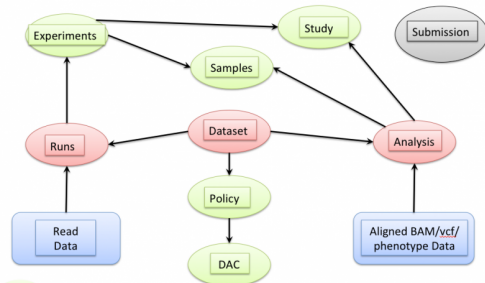
---

---

---

---

### EGA data model



● May register prior to upload  
● Register post upload ONLY

WT NGS Bioinformatics



---

---

---

---

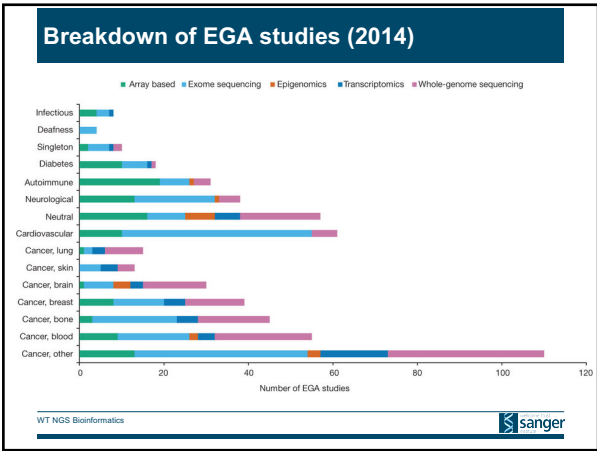
---

---

---

---





---

---

---

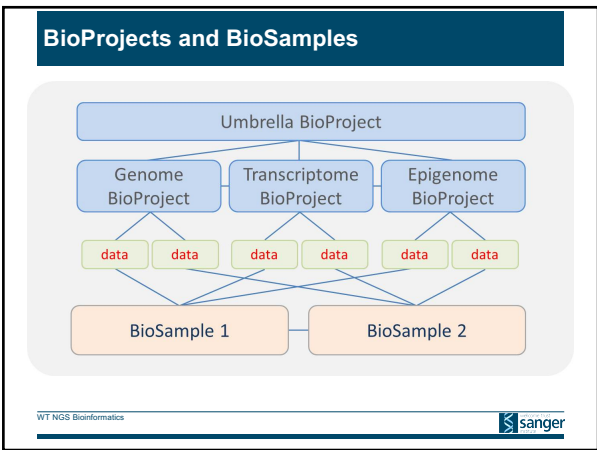
---

---

---

---

---



---

---

---

---

---

---

---

---

- ### BioSample database
- ▶ Stores descriptive information about biological samples used to generate experimental data
    - ▶ e.g. cell line, blood sample, environmental isolate
    - ▶ species, phenotypic information e.g. disease state, clinical info on individual
  - ▶ Can link up data from different archives for same sample
  - ▶ Accessions always begin with SAM
    - ▶ Next is E, N or D, for EBI, NCBI or DDBJ respectively
    - ▶ Next is A or a G, for a sample or a group of samples
    - ▶ Finally is a numeric component
- WT NGS Bioinformatics | sanger

---

---

---

---

---

---

---

---

### BioProject database

- ▶ Organises samples & data produced by projects
  - ▶ Deposited by several research groups
  - ▶ Deposited into several archival databases
- ▶ Can be created for
  - ▶ Genome sequencing and assembly
  - ▶ Transcriptome sequencing and expression
  - ▶ Targeted locus sequencing
  - ▶ Variation detection
- ▶ Accessions always begin with PRJ
  - ▶ Next is E, N or D for EBI, NCBI and DDBJ respectively
  - ▶ Finally is a numeric component

WT NGS Bioinformatics



---

---

---

---

---

---

---

---

### WTSI data sharing policy

- ▶ Aim to provide rapid and open access to data produced
- ▶ Immediate release
  - ▶ Register sequencing studies in BioProject database
  - ▶ Register samples in BioSample database
- ▶ Within 90 days
  - ▶ Primary sequence data (CRAM) in ENA/EGA
- ▶ At publication
  - ▶ Secondary analysis in other archives
    - ▶ VCF, expression data, annotated sequences

WT NGS Bioinformatics



---

---

---

---

---

---

---

---

### Useful resources

- ▶ EBI Training
  - ▶ <https://www.ebi.ac.uk/training/online/course-list>
- ▶ NCBI Handbook
  - ▶ <http://www.ncbi.nlm.nih.gov/books/NBK143764/>
- ▶ DDBJ Training
  - ▶ [http://trace.ddbj.nig.ac.jp/index\\_e.html](http://trace.ddbj.nig.ac.jp/index_e.html)
- ▶ NAR Journal
  - ▶ <http://nar.oxfordjournals.org/>

WT NGS Bioinformatics



---

---

---

---

---

---

---

---



# Structural variation detection and interpretation

Thomas Keane,  
EVA, EGA, ENA archive infrastructure team,  
EMBL-EBI  
@drtkeane  
E: tk2@ebi.ac.uk



ADVANCED COURSES AND SCIENTIFIC CONFERENCES

## SVs and human disease

Table 1 Copy-number variations and neurogenetic disorders (expanded from References 49 and 50)

Syndrome	OMIM	Locus	Rearrangement	Gene(s)	Reference
<i>Neurodevelopmental</i>					
WBS del(7)q11.23	194050	7q11.23	del	CGS incl. <i>ELN</i>	51
dup(7)q11.23	609757		dup		52
AS	105830	15q11-q12	mat del, pat UPD15	<i>UBE3A</i>	53
PWS	178270		pat del, mat UPD15	CGS	54
dup(15)	608636	15q11-q13	dup	CGS	55
idic(15)		idic(15)(q13)	trip		56
MDLS	247200	17p13.3	del	CGS incl. <i>LIS1</i>	57
SMS	182290	17p11.2	del	CGS incl. <i>RAI1</i>	58
PTLS	610883		dup	<i>RAI1</i>	59
NF1	162200	17q11.2	del	CGS incl. <i>NF1</i>	60
del(17)q21.31	610443	17q21.31	del		61-63
DGS/VCFs	188400	22q11.2	del	CGS incl. <i>TBX1, COMT</i>	64
	192430				
dup(22)q11.2	608363		dup	CGS	65
del(22)q13	606232	22q13.3	del	<i>SHANK3/PROSAP2</i>	66
RTT	312750	Xq28	del	<i>MECP2</i>	67
Rett-like syndrome	300260		dup, trip	<i>MECP2</i>	68
PMD	312080	Xq22.2	dup, del	<i>PLP1</i>	69
<i>Neurodegenerative</i>					
PD	188601	4q21	dup, trip	<i>SNCA</i>	70
SMA	253300	5q13	del, gene conv	<i>SMN1, SMN2</i>	71
ADLD	189500	5q23.2	dup	<i>LMNB1</i>	72
CMT1A	118220	17p12	dup	<i>PMP22</i>	73, 74
HNPP	162500		del		75
AD	104300	21q21	dup	<i>APP</i>	76

Abbreviations: AD, Alzheimer disease; ACD, autosomal dominant leukodystrophy; AS, Angelman syndrome; CGS, contiguous gene deletion/duplication syndrome; CMT1A, Charcot-Marie-Tooth type 1 disease; del, deletion; dup(7)q11.23, reciprocal duplication of the WBS region; dup, duplication; gene conv, gene conversion; HNPP, hereditary neuropathy with liability to pressure palsies; MDLS, Miller-Dieker syndrome; NF1, neurofibromatosis type 1; PD, Parkinson disease; PMD, Polizaeus-Merzbacher syndrome; PWS, Prader-Willi syndrome; RTT, Rett syndrome; SMA, spinal muscular atrophy; trip, triplication; UPD, uniparental disomy; WBS, Williams-Beuren syndrome.

Stankiewicz and Lupski (2010) Ann. Rev. Med.



ADVANCED COURSES AND SCIENTIFIC CONFERENCES

## Genomic structural variation

- Any form of rearrangement of chromosome structure
  - Contribute to genetic diversity and evolution, new gene formation, gene function, phenotypic diversity, rare variants of large effect
- Frequent causes of disease
  - Referred to as genomic disorders
  - Mendelian diseases or complex traits such as behaviors
  - E.g. increase in gene dosage due to increase in copy number
- Several type or categories of structural variation
  - Insertions, deletions, copy number changes, inversions, translocations
  - Complex events contain combinations of these in close proximity
- Breakpoint: as a pair of bases that are adjacent in an experimentally sequenced 'sample' genome but not in the reference genome
- Many experimental techniques to detect SVs



ADVANCED COURSES AND SCIENTIFIC CONFERENCES

## Methods for detecting SVs

- **Chromosome banding:** chromosomes are prepared from dividing cells, stained, and viewed with a microscope. Large deletions, duplications, and translocations are detected if the banding pattern or chromosome structure is altered.
- **Fluorescence in situ hybridization (FISH):** fluorescent-labeled DNA probes hybridize to metaphase or interphase cells to visualize a locus on a chromosome and determine copy number. FISH can determine the location of chromosomal segments identified by microarray, NGS, and WGS.
- **Microarray:** array comparative genome hybridisation (array CGH) detects copy-number differences between abnormal and reference genomes. SNP arrays detect changes in copy-number and allelic ratios. CNV location and SV organization are not determined by microarray methods.
- **Whole-genome sequencing (WGS):** Breakpoints of CNV and copy-neutral SV are detectable by paired-end reads that have discordant mappings to the reference genome.
- **Mate-pair sequencing:** whereas standard NGS methods sequence the ends of 300-500 bp DNA fragments, mate-pair or jumping libraries sequence the ends of DNA fragments that are several kb in length. These large-fragment mate-pair libraries increase the likelihood of detecting SV that has breakpoints within interspersed repeats.
- **Third generation sequencing:** sequencing long molecules of DNA (several kbp) and subsequent alignment to a reference genome to detect SVs

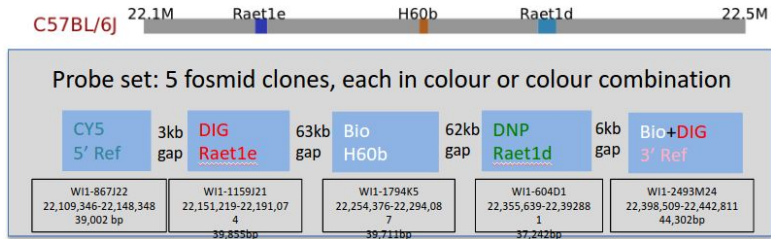
Weckselblatt and Rudd (2015) Trends in Genetics



ADVANCED COURSES AND SCIENTIFIC CONFERENCES

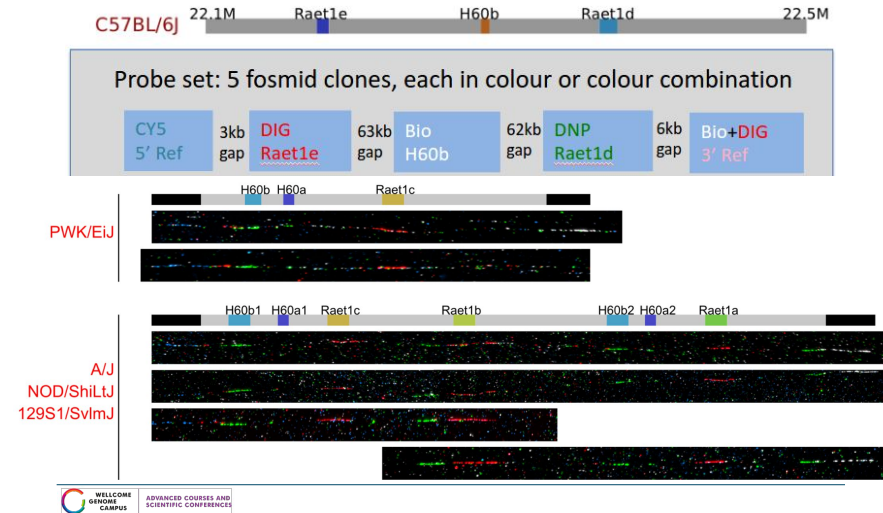
## Fiber FISH

Chr10:21,100,000-21,700,000

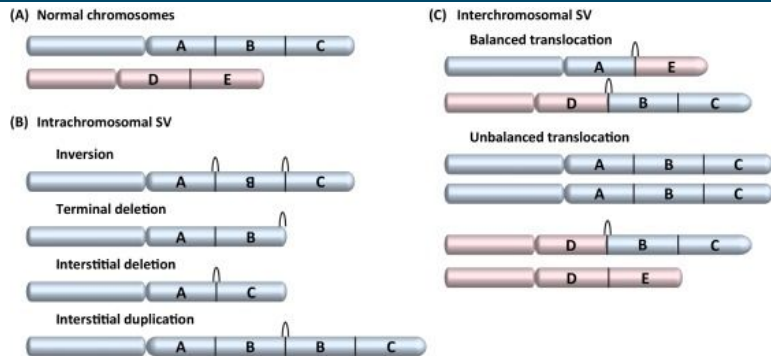


## Fiber FISH

Chr10:21,100,000-21,700,000



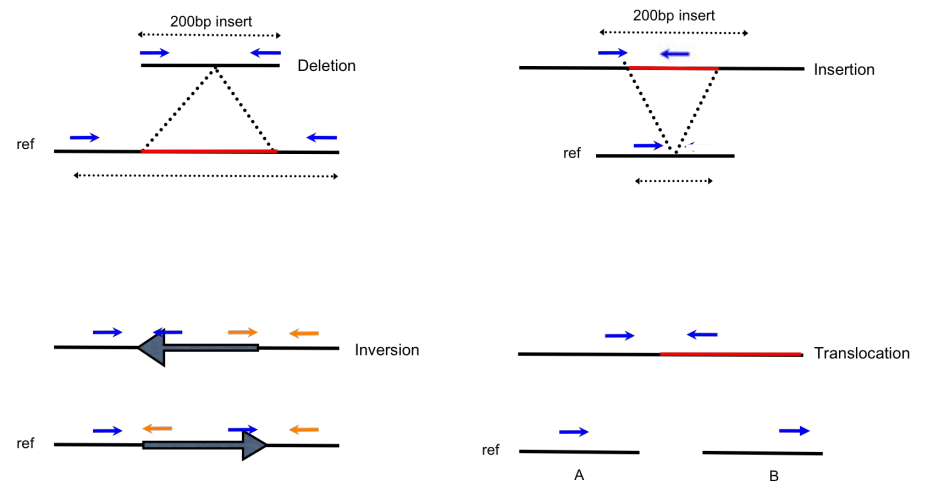
## SV types



(A) Two nonhomologous chromosomes shown in blue and pink. Segments are labeled with letters A-E. Black arches indicate structural variation (SV) breakpoint junctions. (B) Intrachromosomal rearrangements include inversions, interstitial and terminal deletions, and interstitial duplications. (C) Simple translocations between two different chromosome ends. Balanced translocations do not result in copy-number variation (CNV), but unbalanced translocations have partial monosomy (segment E) and partial trisomy (segments B,C).

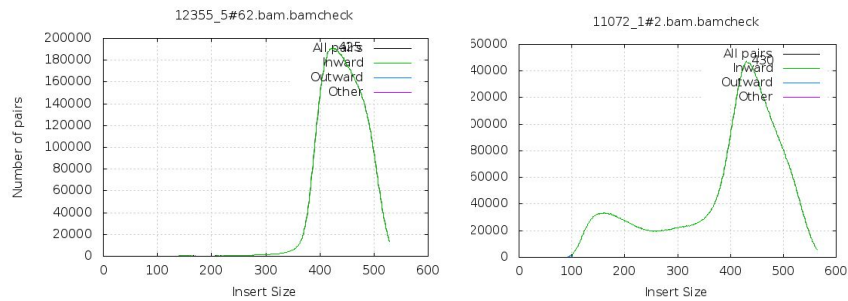
Weckselblatt and Rudd (2015) *Trends in Genetics*

## SV types and NGS paired-end sequencing

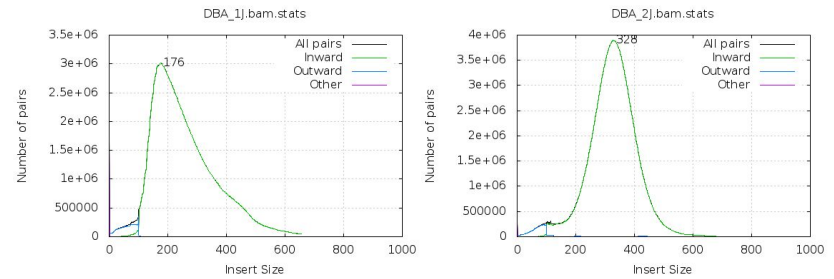




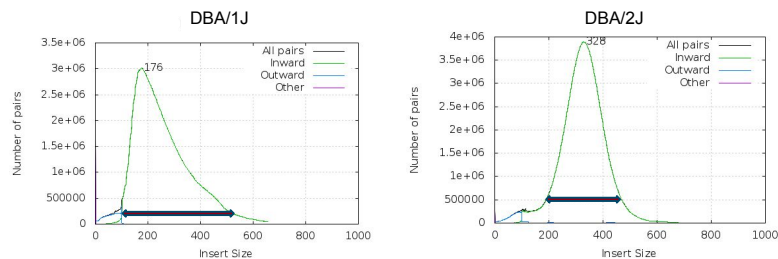
## Fragment Size QC



## Fragment size again



## Fragment size again



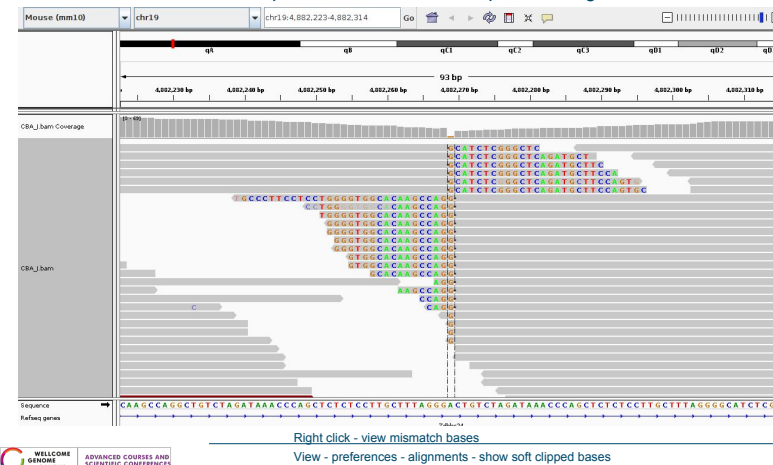
DBA/1J fragment size distribution has larger range (~450bp) vs. DBA/2J (~250bp)

SV caller only considers read pairs discordant if they fall outside of the extremes of the fragment size distribution

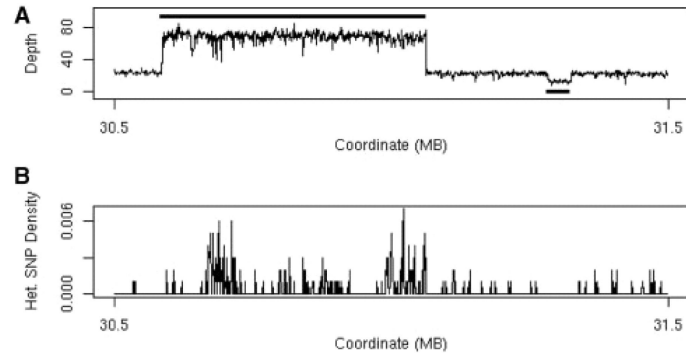
- Observed in DBA/1J that we had lower sensitivity to call SVs in the 300-500bp range compared to DBA/2J

## Sources of evidence 2: Split reads

- A split-read alignment is a single DNA fragment that spans a breakpoint and therefore does not contiguously align to the reference genome
- Errors in the sequencing and alignment processes creates some ambiguity in the exact location of the breakpoint associated with a split-read alignment



## Sources of evidence 3: Read depth

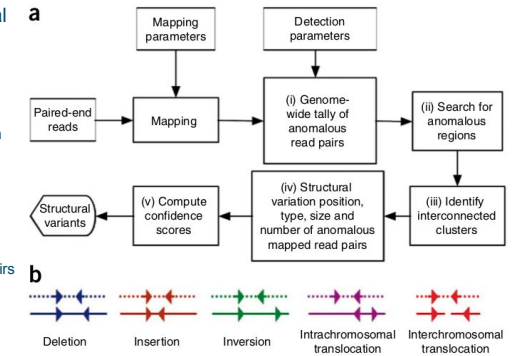


**Fig. 1.** (A) Plot of sequencing depth across a one megabase region of A/J chromosome 17 clearly shows both a region of 3-fold increased copy number (30.6–31.1 Mb) and a region of decreased copy number (at 31.3 Mb). The solid black line above the depth plot indicates the called copy number gain and the solid black line below the plot indicates the called copy number loss. (B) Plot of the heterozygous SNP rate for the same region showing the high number of apparent heterozygous SNPs associated with the copy number gain.

Simpson et al. (2009) *Bioinformatics*

## Read pairs: Breakdancer

- Identifies deletions, insertions, inversions and intrachromosomal and interchromosomal translocations
- Input: BAM file
- Algorithm:
  - Analyse a subset of reads from each sequencing library (determine mean and standard deviation of fragment size)
  - Walk along each chromosome to identify all of the anomalous read pairs



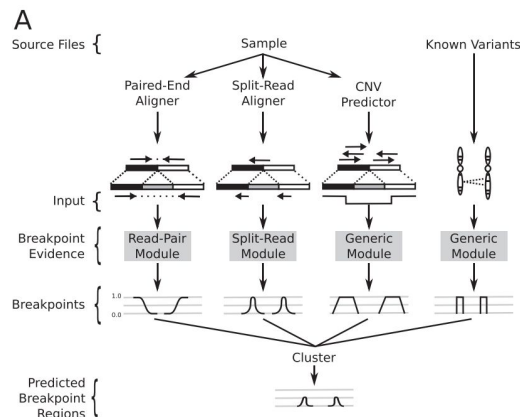
**Figure 1** | Overview of BreakDancer algorithm. (a) The workflow. (b) Anomalous read pairs recognized by BreakDancerMax. A pair of arrows represents the location and the orientation of a read pair. A dotted line represents a chromosome in the analyzed genome. A solid line represents a chromosome in the reference genome.

- Output
  - Text with one SV event per line
  - Filter by: minimum number of reads, quality score, type of SV

Chen et al. (2009) *Nature methods*

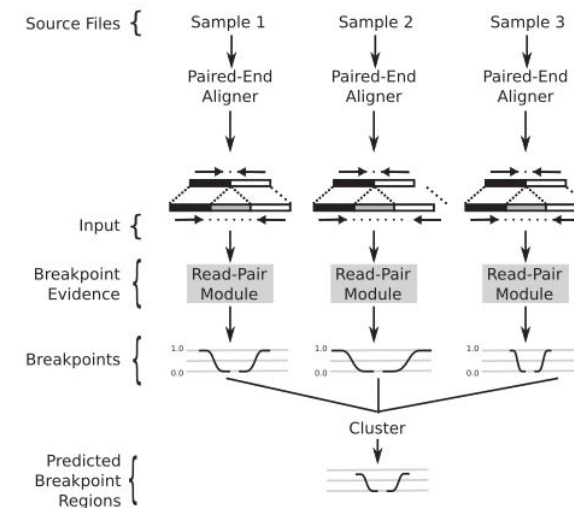
## Read pairs + split reads + depth: Lumpy

- Any number of alignment signals can be integrated into a single discovery process
  - Read pairs, split-reads, depth, user supplied evidence
- Distinct modules that map signals from each alignment evidence type to the common probability interval pair.
- Evidence from the different alignment signals is mapped to breakpoint intervals, overlapping intervals are clustered and the probabilities are integrated



Layer et al. (2014) *Genome Biology*

## Lumpy: multi-signal and multi-sample workflows



Layer et al. (2014) *Genome Biology*

## VCF for SVs

```
##fileformat=VCFv4.1
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/sv/breakpoint_assemblies.fasta
##INFO=

```

## VCF for SVs

```
##fileformat=VCFv4.1
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/sv/breakpoint_assemblies.fasta
##INFO=

```



## VCF for SVs

```
##fileformat=VCFv4.1
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/sv/breakpoint_assemblies.fasta
##INFO=

```

## VCF for SVs

```
##fileformat=VCFv4.1
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/sv/breakpoint_assemblies.fasta
##INFO=

```

- What does the CIEND info tag describe?
- How many different types of insertions can be described from the ALT tags?
- The first and second entries are both deletions, but what is the difference between them?



## VCF for SVs

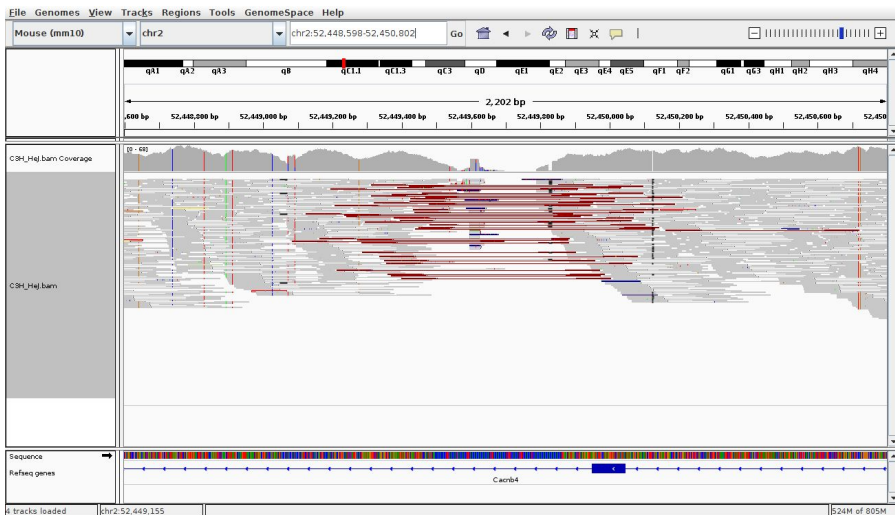
```
##fileformat=VCFv4.1
##fileDate=20100501
##reference=1000GenomesPilot-NCBI36
##assembly=ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/sv/breakpoint_assemblies.fasta
##INFO=<ID=REFID,Number=,Type=String,Description="ID of the assembled alternate allele in the assembly file">
##INFO=<ID=CI,Number=2,Type=Integer,Description="Confidence interval around END for imprecise variants">
##INFO=<ID=CIPPOS,Number=2,Type=Integer,Description="Confidence interval around POS for imprecise variants">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the variant described in this record">
##INFO=<ID=HOMLEN,Number=,Type=Integer,Description="Length of base pair identical micro-homology at event breakpoints">
##INFO=<ID=HOMSEQ,Number=,Type=String,Description="Sequence of base pair identical micro-homology at event breakpoints">
##INFO=<ID=SVLEN,Number=,Type=Integer,Description="Difference in length between REF and ALT alleles">
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
##ALT=<ID=DEL,Description="Deletion">
##ALT=<ID=DEL:ME:ALU,Description="Deletion of ALU element">
##ALT=<ID=DEL:ME:L1,Description="Deletion of L1 element">
##ALT=<ID=DUP,Description="Duplication">
##ALT=<ID=DUP:TANDEN,Description="Tandem Duplication">
##ALT=<ID=INS,Description="Insertion of novel sequence">
##ALT=<ID=INS:ME:ALU,Description="Insertion of ALU element">
##ALT=<ID=INS:ME:L1,Description="Insertion of L1 element">
##ALT=<ID=INV,Description="Inversion">
##ALT=<ID=CNV,Description="Copy number variable region">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype quality">
##FORMAT=<ID=CN,Number=1,Type=Integer,Description="Copy number genotype for imprecise events">
##FORMAT=<ID=CNQ,Number=1,Type=Float,Description="Copy number genotype quality for imprecise events">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT BAQ0001
1 2827694 rs2376870 C<T>GATCCGGGAC C <DEL> 6 PASS SVTYPE=DEL;END=2827708;HOMLEN=1;HOMSEQ=C;SVLEN=-14 GT:GQ 1/1:13.9
2 14477084 . C <DEL:ME:ALU> 12 PASS SVTYPE=DEL;END=14477381;SVLEN=-297;CIPOS=-22,18;CIEND=-12,32 GT:GQ 0/1:12
3 9426916 . A <INS:ME:L1> 23 PASS SVTYPE=INS;END=9426916;SVLEN=6027;CIPOS=-16,22 GT:GQ 1/1:15
4 12665100 . C <DUP> 14 PASS SVTYPE=DUP;END=12665200;SVLEN=21100;CIPOS=-500,500;CIEND=-10,10 GT:GQ:CN:CNQ ./.:0:3:16.2
4 18665128 . T <DUP:TANDEN> 11 PASS SVTYPE=DUP;END=18665204;SVLEN=76;CIPOS=-10,10;CIEND=-10,10 GT:GQ:CN:CNQ ./.:0:5:8.3
```

- Can you write out the VCF entry for a Alu insertion at chromosome 7, position 125467, of length 258bp, and a breakpoint confidence interval of +/-20bp with one sample that is heterozygous for the insertion and has genotype quality of 40?

## SV Visualisation

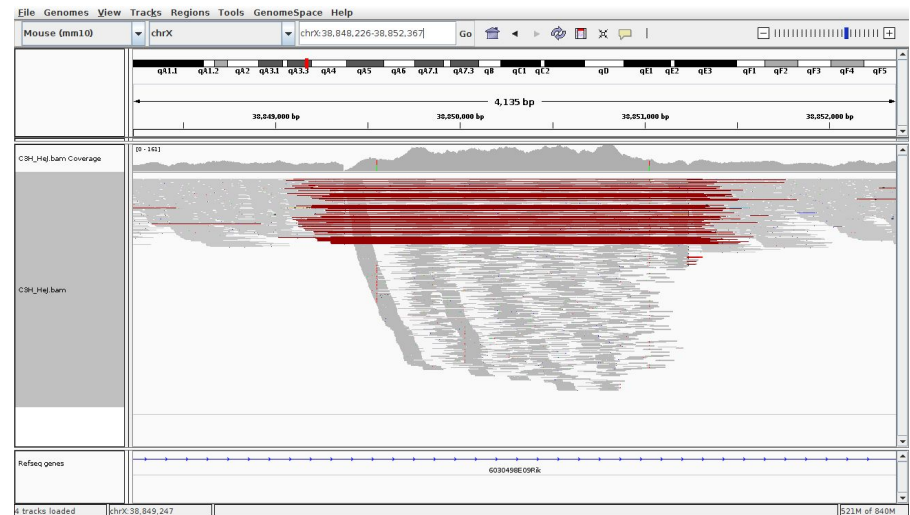
- Structural variation visualisation can be more challenging than SNPs and indels
- Inspect several hundred base pairs or multiple kbp
- Analyse complicated read pair patterns to determine type of SV and sources of error
- Look for soft clipped bases for breakpoint accuracy
- Many NGS visualisation software packages exist
- IGV from Broad institute is a popular and easy to use visualisation software
  - Requires BAM file and fasta file of the reference genome
  - Viewing settings need to be tailored for the type of SV being visualised (see notes below each screenshot)

## IGV - Deletion



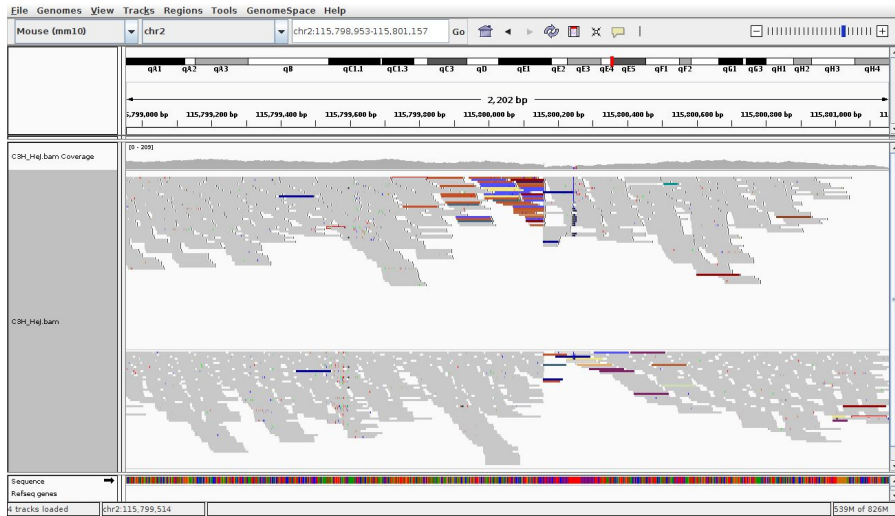
Right click - Squished

## IGV - Repeat element deletion



Right click - Squished

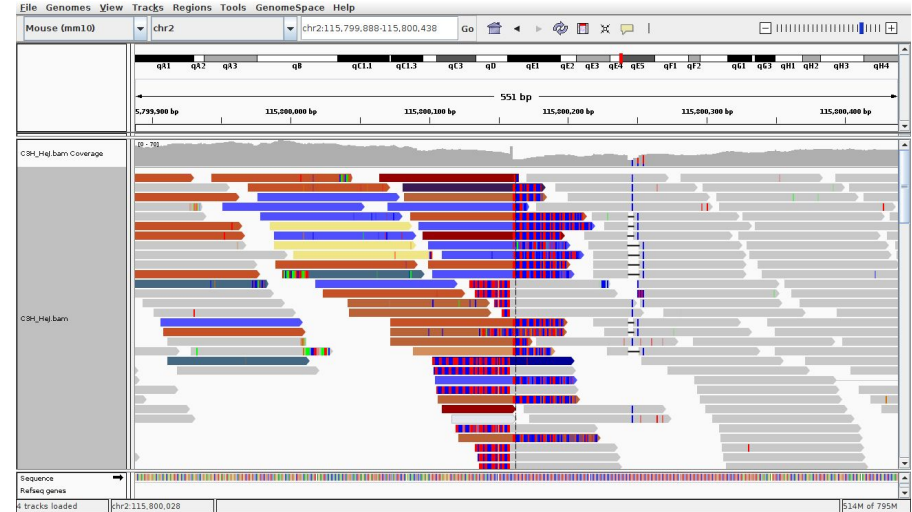
## IGV - Insertion



Right click - group alignments by 'read strand'



## IGV - Insertion (zoomed)

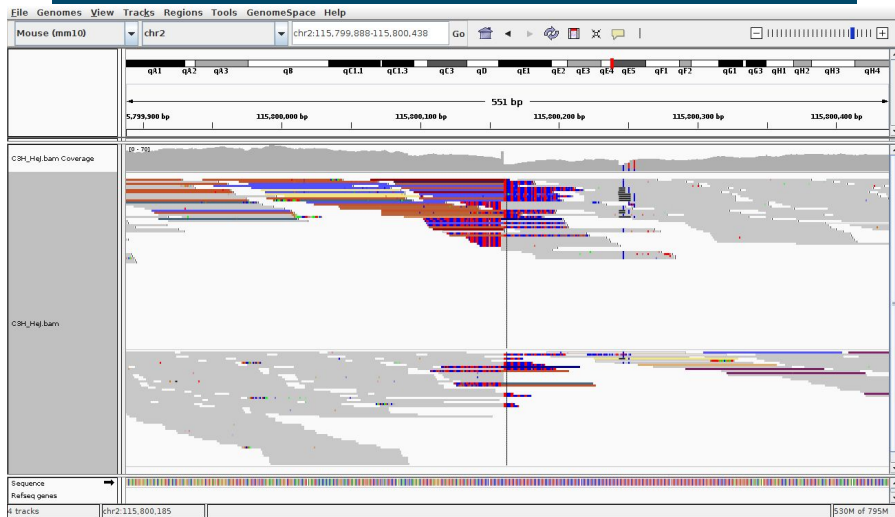


Right click - view mismatch bases

View - preferences - alignments - show soft clipped bases



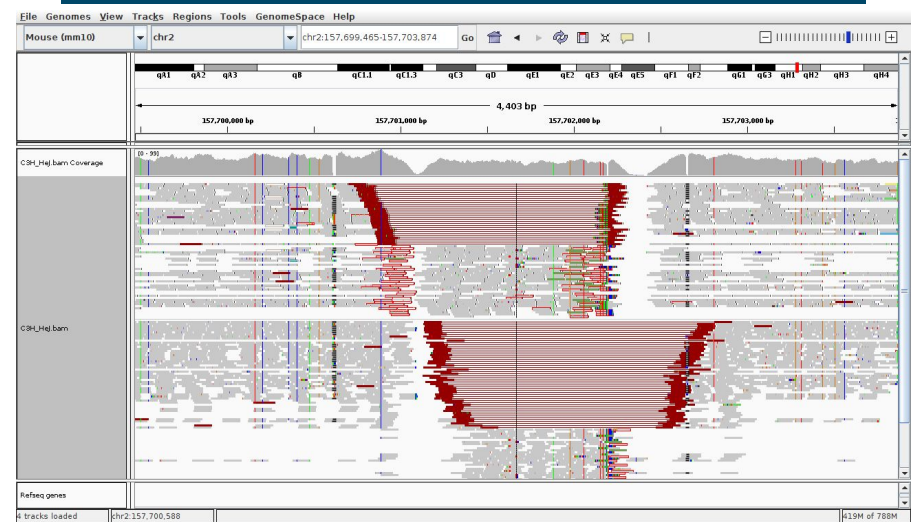
## IGV - Insertion (zoomed)



Right click - Squished



## IGV - what is this?



➤ Right click - group alignments by - read strand

➤ Red lines are reads that are aligned further apart than expected





## IGV - mouse over or click on a red read

```

Left alignment
Read name = HS4_07512:4:2201:4712:44023#7
Sample = C3H_Hej
Read group = 7512_4#7
-----
Location = chr2:157,700,909
Alignment start = 157,700,856 (+)
Cigar = 100M
Mapped = yes
Mapping quality = 46
Secondary = no
Supplementary = no
Duplicate = no
Failed QC = no
-----
Base = T
Base phred quality = 38
-----
Mate is mapped = yes
Mate start = chr2:157702197 (+)
Insert size = 1343
First in pair
Pair orientation = F1F2
-----
MD = 32T67
RG = 7512_4#7
NM = 1
MQ = 56
AS = 95
XS = 63
CT = 1F100M1242T2F23S77M
-----

Right alignment
Read name = HS4_07512:4:2201:4712:44023#7
Sample = C3H_Hej
Read group = 7512_4#7
-----
Location = chr2:157,700,909
Alignment start = 157,702,198 (+)
Cigar = 23S77M
Mapped = yes
Mapping quality = 56
Secondary = no
Supplementary = no
Duplicate = no
Failed QC = no
-----
Mate is mapped = yes
Mate start = chr2:157700855 (+)
Insert size = -1343
Second in pair
Pair orientation = F1F2
-----
MD = 52C24
RG = 7512_4#7
NM = 1
MQ = 46
AS = 72
XS = 41
-----
    
```

## SVs and long read sequencing

- Single molecule sequencing of large DNA fragments
- Platforms: Oxford nanopore and Pacific Biosciences
- Read lengths 10-20Kbp routinely
- Longer than most common transposable element repeats
- What does it mean for SV detection? Span both breakpoints with single read
- Some new challenges
  - Reads are error prone, 5-20% error
  - Challenging to align the reads correctly

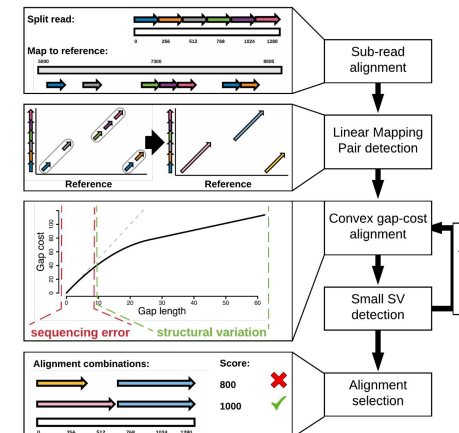


## Alignment challenges

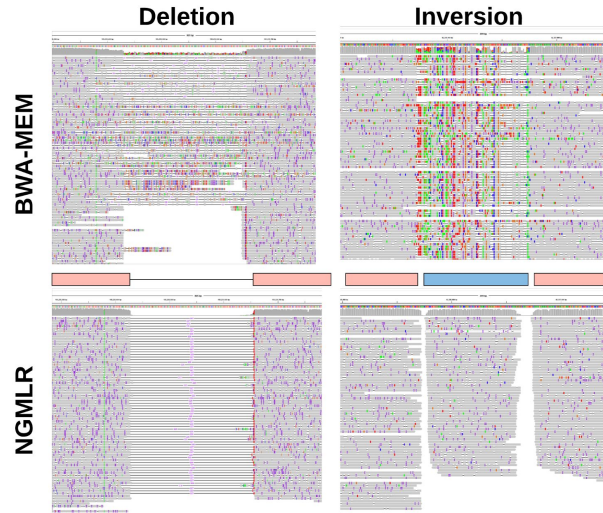


## CoNvex Gap-cost alignMents for Long Reads (NGMLR)

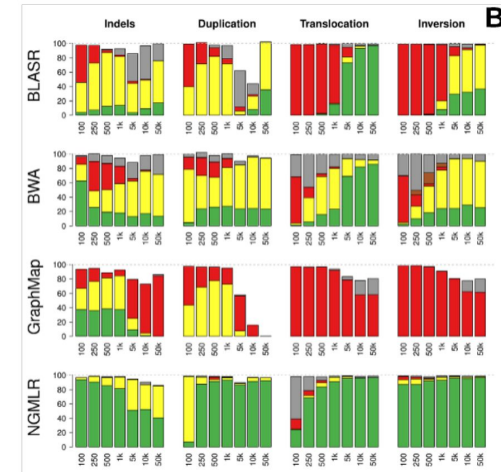
- NGMLR - aligner specifically designed for long reads
- Convex scoring model
  - Extending an indel is penalized proportionally less the longer the indel is



## Alignment challenges



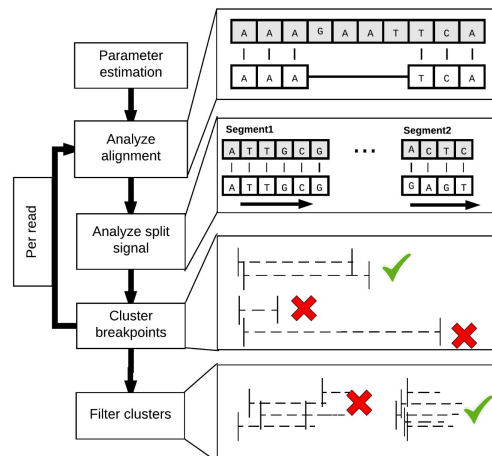
## Comparison of aligners (simulated data)



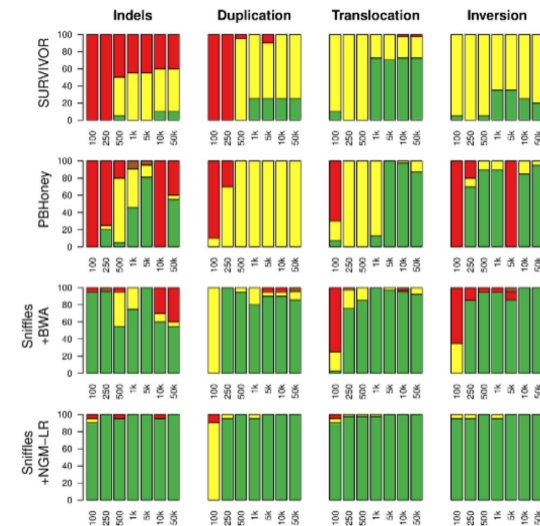
Alignment status: Precise (green), indicated (yellow), forced (red), unaligned reads (white), or trimmed but not aligned through the SV (grey).

## Sniffles

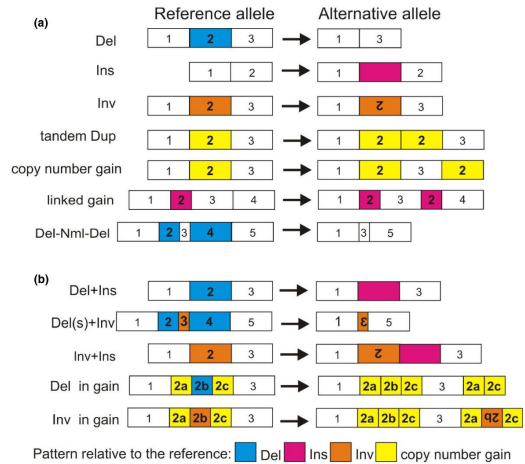
SV detection from long read alignments



## Sniffles performance



## Complex SVs

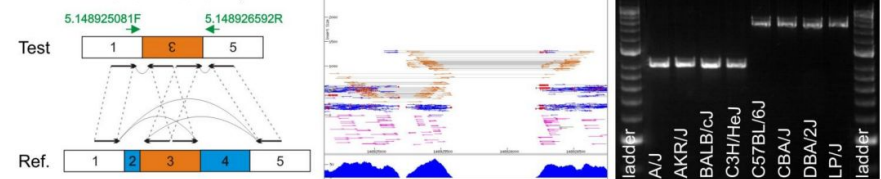


**Figure 3.** Architecture of structural variants. (a) Simple SVs: deletion (Del), insertion (Ins), inversion (Inv), tandem duplication (tandem Dup) and other types of copy number gains. Linked gain is a small copy number gain at close proximity to its copy. Inverted linked gain (not drawn) is similar to a linked gain but the copy is inverted. Del+Nml+Del is two deletions separated by a normal copy of small size. (b) Complex SVs: deletion co-occurring with insertion (Del+Ins), inversion with flanking deletions (Del(s)+Inv), inversion with insertion (Inv+Ins), deletion within a copy number gain (Del in gain) and inversion within a copy number gain (Inv in gain). Yalcin *et al.* (2012) Genome Biology

## Complex SV Examples

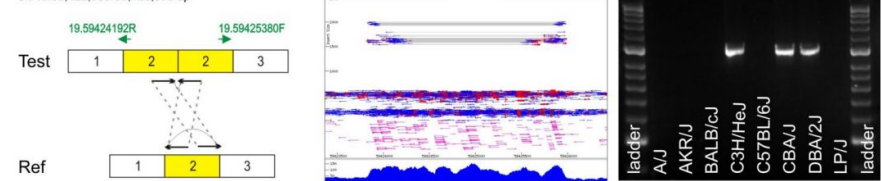
### H5

del-71bp\_inv-325\_del-645 [11110000]  
 1st del - chr5:148,925,178-148,925,248 bp  
 inv - chr5:148,925,249-148,925,573 bp  
 2nd del - chr5:148,925,574-148,926,218 bp



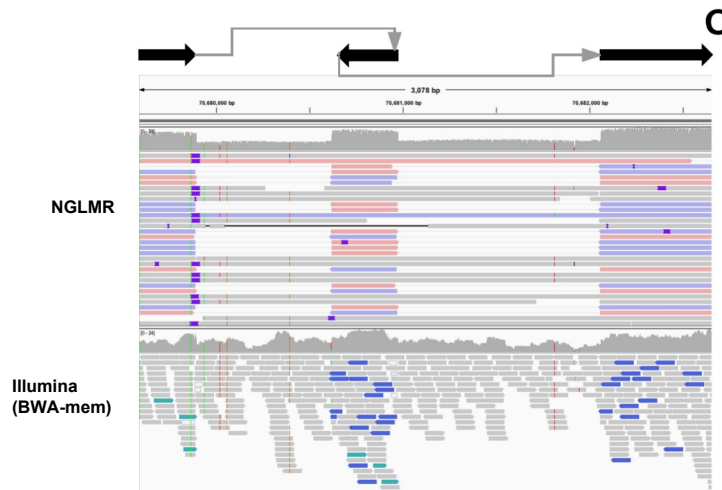
### H8

Tandem duplication of 2181 bp [00010110]  
 chr19:59,423,833-59,425,976 bp



Yalcin *et al.* (2012) Genome Biology

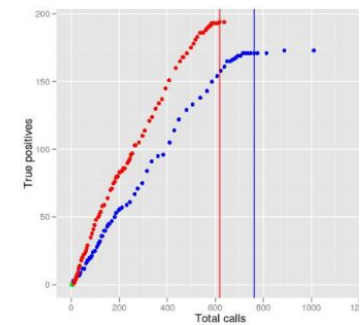
## Complex SVs - Long reads



3kb region: two deletions flanking an inverted sequence

## Evaluating SV calls

- Specificity vs sensitivity
- False positives vs. false negatives
- Desirable to have high sensitivity and specificity
- How to determine sensitivity?
  - External sources of true/known SVs
- Specificity
  - Validate a random selection of SVs by another technology
  - e.g. PCR products
- Receiver operator curves to investigate effects of varying parameters



## Computer exercises

1. Trivia questions about a VCF output file from the Lumpy SV caller.
  - a. <http://www.genomebiology.com/2014/15/6/R84>
2. Use the Breakdancer software package to call structural variants on a yeast sample that was paired-end sequenced on the illumina Hiseq.
3. Use the Lumpy software package to call structural variants on a yeast sample that was paired-end sequenced on the illumina Hiseq.
4. Call SVs using the Sniffles caller on a yeast sample that was sequenced on the Pacbio platform.
5. Introduction to BEDtools for doing regional comparisons over genomic co-ordinates.

# WTAC NGS Bioinformatics Course

## Module 5: Structural variant calling

Wednesday 10th October, 2018

Open a terminal and go to the Module 5 directory.

### Exercise 1: VCF for SVs

On the terminal, cd to the 'exercise1' directory.

There is a VCF file called ERR1015121.vcf that was produced using the Lumpy SV calling software. You can read the VCF file by using the less command:

```
less ERR1015121.vcf
```

- What does the CIPOS format tag indicate?
- What does the PE tag indicate?
- What tag is used to describe an inversion event?
- What tag is used to describe a duplication event?
- How many deletions were called in total? (Hint: DEL is the info field for a deletion. The -c option of the grep command can be used to return a count of matches.)
- What type of event is predicted at IV:437148? What is the length of the SV? How many paired-end reads and split-reads support this SV variant call?
- What is the total number of SV calls predicted on the IV chromosome?

### Exercise 2: Breakdancer

On the terminal, cd to the 'exercise2' directory.

In this exercise, we will use the Breakdancer software package to call structural variants on a yeast sample that was paired-end sequenced on the illumina HiSeq 2000.

**2.1** Breakdancer first needs to examine the BAM file to get information on the fragment size distribution for each sequencing library contained in the BAM file.

The `breakdancer.config` file has information about the sequencing library fragment size distribution. Use the 'cat' command to print the contents of the 'breakdancer.config' file.

- What is the mean and standard deviation of the fragment size?

**2.2** Now we will run the breakdancer SV caller. Run the command:

```
breakdancer_max breakdancer.config > ERR1015121.breakdancer.out
```

The output from Breakdancer is a simple text format with one line per SV event (NOT VCF format). Breakdancer calls four different types of structural variants: deletions (DEL), insertions (INS), inversions (INV), intra chromosomal translocations (ITX), and inter chromosomal translocations (CTX).

- What type of SV event is predicted at position III:83065? What is the size of this SV? What is the score of this SV?
- What type of SV event is predicted at position II:258766?

**2.3** Next you will convert the output of breakdancer into a standard format call BED, that is accepted by many other tools and genome browsers. The BED format is explained here: <https://genome.ucsc.edu/FAQ/FAQformat.html#format1>

Can you create a BED file for the **deletions** that were predicted by breakdancer in 2.2 using standard unix commands. Here are some hints:

1. Extract all the deletions from the breakdancer.out file (Hint: use grep)
2. We want to print columns: 1, 2, 5, 7, and 9 to correspond to create a BED file with columns: chromosome, start, end, name, and score. (Hint: use awk to do this, e.g. awk '{print \$1"\t"\$2}')
3. Print the resulting bed output into a file called: breakdancer.dels.bed

**2.4** Can you open the IGV genome browser and inspect the SV event at II:258766. To do this, type:

```
igv.sh &
```

First, you need to open the genome (click on the leftmost dropdown box at the top, and find 'S. cerevisiae EF3 r62' and select it). Then use the File menu to open your BAM file (File - Load from file, and select ERR1015121.bam). Next load the BED file for the deletion calls that you created in 2.3 (File - Load from file, and select 'breakdancer.dels.bed').

In the location text box, type: II:257766-259766:

- Can you see the structural variant? (Hint: you may need to zoom out a little to see the full structural variant).
- Can you see any evidence to supports this SV call?
- Can you estimate the size of the SV?

The VCF in the exercise 1 directory was produced by another structural variant caller on the same sample as this exercise. Can you load this VCF into IGV also (File - Load from file, and select ERR1015121.vcf in the exercise 1 directory) and answer the following questions:

- Was the deletion at II:258766 also called by the other structural variant software?
- Can you navigate to II:508,064-511,840? Is there a SV deletion called in this region by either SV caller? Is there any read support for a SV deletion in this region? If so, how many read pairs could support the deletion call (Hint: change the IGV view to 'squished' and 'View as pairs' to see any inconsistently aligned read pairs).

### Exercise 3: Lumpy structural variant caller

In this exercise, we will use the Lumpy software package to call structural variants on a yeast sample that was paired-end sequenced on the Illumina Hiseq 2000.

Lumpy is designed to take BAM files that have been aligned with BWA-mem.

**3.1** On the terminal, cd to the 'exercise3' folder and check that there is a BAM file called ERR1015069.bam and index file in the directory (hint: ls -l).

**3.2** The first step for running Lumpy is to extract the read pairs that are discordantly mapped (i.e. pairs that are not mapped within the expected fragment size distribution). We will use Samtools to extract these reads:

```
samtools view -bh -F 1294 ERR1015069.bam | samtools sort -O bam -T
ERR1015069.temp -o ERR1015069.discordants.bam
```

- Can you index the bam file? (Hint: use samtools index)
- What does the -F option in 'samtools view' do?
- Which BAM flags does 1294 indicate? (Hint: in your web browser, visit <https://broadinstitute.github.io/picard/explain-flags.html> and enter 1294 to find out)

**3.3** Next we will use Lumpy to extract the reads that are only split mapped (i.e. split read alignments). This is all one single command:

```
samtools view -h ERR1015069.bam | extractSplitReads_BwaMem -i stdin | samtools
view -b - | samtools sort -O bam -T ERR1015069.temp -o ERR1015069.splitters.bam
```

- Can you index the bam file? (Hint: use samtools index)

**3.4** We will now do the structural variant calling with lumpy, providing it with the original BAM file and the two BAM files we prepared in 3.2 and 3.3.

```
lumpyexpress -B ERR1015069.bam -S ERR1015069.splitters.bam -D
ERR1015069.discordants.bam -o ERR1015069.vcf
```

- What type of SV event occurs at position IV:383993? What is the length of the SV event?

- What type of SV event occurs at position XV:43018? What is the length of the SV event?

#### Exercise 4: Sniffles - long read SV caller

Sniffles is a SV caller that is designed for long reads (Pacbio or Oxford Nanopore). It is very important that the reads are first aligned with an aligner suitable for long reads. NGMLR is a long-read mapper designed to align PacBio or Oxford Nanopore (standard and ultra-long) to a reference genome with a focus on reads that span structural variations.

We will use data from a *Saccharomyces cerevisiae* strain (YPS128) that was sequenced at the Wellcome Trust Sanger Institute and deposited in the ENA (Project: PRJEB7245, sample: SAMEA2757770, analysis: ERZ448241).

**4.1** On the terminal, cd to the 'exercise3' directory.

The sequencing reads are contained in a fastq file:

```
YPS128.filtered_subreads.10x.fastq.gz
```

The reference genome is in the ref directory in a fasta file:

```
Saccharomyces_cerevisiae.R64-1-1.dna.toplevel.fa
```

- Can you align the reads with NGMLR and send the output to a SAM file called `YPS128.10x.filtered_subreads.sam`? You can find the usage of `ngmlr` by typing:  
`ngmlr`

**Note:** the `-t` parameter to use multiple threads in parallel (this will increase the speed of the alignment by using more than one CPU core - for these machines, I suggest using 6).

- Can you convert the output to BAM format (`samtools view -b`)
- Sort the BAM file (`samtools sort`) and produce a sorted BAM file called:  
`YPS128.10x.filtered_subreads.sorted.bam`
- Finally, use `samtools` to index the sorted BAM file (`samtools index`).

**4.2** We will now use the BAM file to do structural variant calling with Sniffles.

Sniffles takes the BAM file as input and outputs VCF. Using the default parameters, can you call SVs with Sniffles and output the results into a VCF file called `YPS128.10x.vcf`. To find the usage for Sniffles, type:

```
sniffles
```



You don't need to change any of the default parameters, but you will need to work out how to provide the input BAM file and specify the output VCF file. The documentation on sniffles is here: <https://github.com/fritzsedlazeck/Sniffles/wiki/Parameter>

### 4.3 IGV inspection

- What sort of SV was called at on chromosome 'Mito' at position 29295?
- What is the length of the SV?
- How many reads are supporting the SV?
- From a visual inspection of the SV in IGV, can you determine how accurate is the breakpoint of the called SV compared to what you see in IGV?

### Optional exercise 5: Bedtools

On the terminal, cd to the 'exercise5' directory.

Bedtools is an extremely useful tool for doing regional comparisons over genomic co-ordinates. It has many commands for doing region based comparisons with BAM, VCF, GFF, BED file formats. To see the list of commands available, on the command line type:

```
bedtools
```

In this directory, there are two VCF files and the yeast genome annotation in GFF3 format (Saccharomyces\_cerevisiae.R64-1-1.82.genes.gff3).

**5.1** For instance, we can quickly find out how many of the SVs intersect with annotated regions of the genome by using the 'bedtools intersect' command. For the intersect command, the -a and -b parameters are used to denote the input files.

- Using the 'bedtools intersect', can you determine how many SVs in ERR1015069.dels.vcf overlap with an annotated region of the yeast genome? (hint: use the unix command wc to count the number of lines in the output, and note the -u parameter in bedtools intersect).
- How many SVs do not overlap with a gene? (hint: note the -v parameter to bedtools intersect)

**5.2** The default is to report overlaps between features in A and B so long as at least one base pair of overlap exists. However, the -f option allows you to specify what fraction of each feature in A should be overlapped by a feature in B before it is reported.

- Can you be more strict and require 50% of overlap between the SV and the genes?
- How many SVs overlap with this more strict definition?

**5.3** You can use bedtools to find the closest feature to a SV using the 'bedtools closest' command.

- What is the closest gene to the structural variant at IV:384220?

**5.4** We can use the 'bedtools intersect' command to determine how many SVs overlap between two VCF files.

- Can you determine how many SVs have a 50% reciprocal overlap between the two files: ERR1015069.dels.vcf ERR1015121.dels.vcf (Hint: first find the option for reciprocal overlap by typing: bedtools intersect -h)

### References:

LUMPY: a probabilistic framework for structural variant discovery  
<http://www.genomebiology.com/2014/15/6/R84>

BreakDancer: an algorithm for high-resolution mapping of genomic structural variation  
<http://www.nature.com/nmeth/journal/v6/n9/abs/nmeth.1363.html>

NGMLR: <https://github.com/philres/ngmlr>

Sniffles: <https://github.com/fritzsedlazeck/Sniffles/wiki>

More information can be found in this paper: Accurate detection of complex structural variations using single molecule sequencing

<https://www.biorxiv.org/content/early/2017/07/28/169557>

BEDTools: a flexible suite of utilities for comparing genomic features  
<http://bioinformatics.oxfordjournals.org/content/26/6/841.long>

### URLs

Lumpy-SV: <https://github.com/arq5x/lumpy-sv>

Breakdancer: <http://gmt.genome.wustl.edu/packages/breakdancer/>

Bedtools: <https://github.com/arq5x/bedtools>

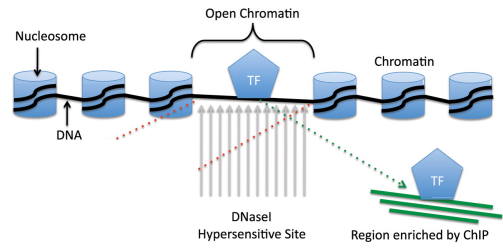
### Documentation

Calling SVs with Lumpy: <https://github.com/arq5x/lumpy-sv#lumpy-traditional-usage>

# Intro to ChIP-seq

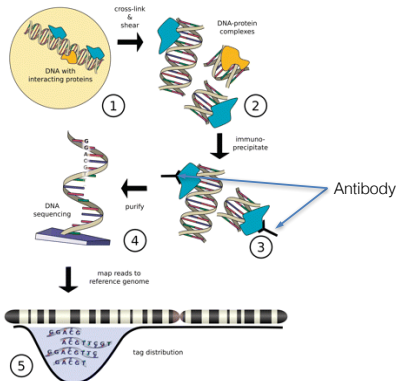
Victoria Offord  
Based on material by  
Daniel Gaffney

## Epigenetics/ChIP in one slide

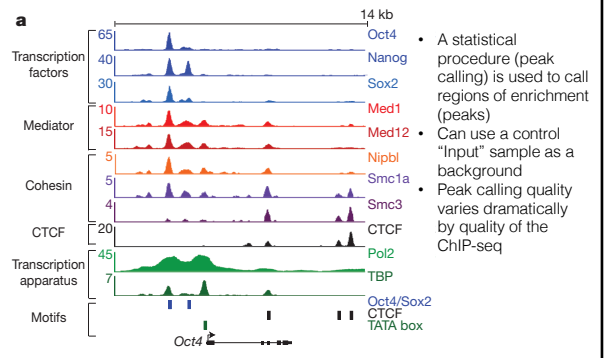


- Regulation of transcription involves interaction of protein and DNA

## How does ChIP-seq work?



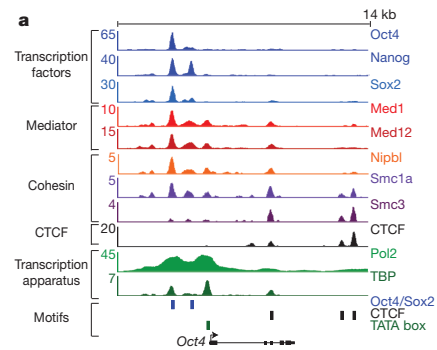
## What does ChIP-seq look like?



### Applications of ChIP-seq

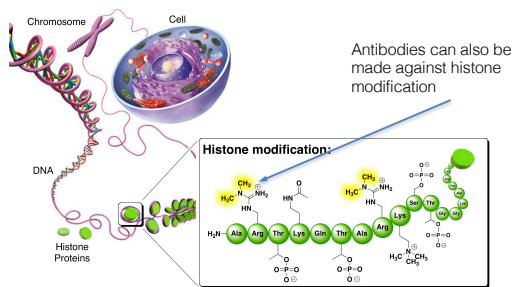
- ChIP-seq is one of the most commonly used approaches for identifying gene regulatory regions
- Two common types:
  1. Transcription factors
  2. Histone modifications

### ChIP-seq for transcription factors



- Each of these TFs requires a high quality, ChIP-grade anti-body
- Most antibodies (~60%) are not good enough for ChIP-seq

### Histone modifications



### Histone mark cheat sheet

Histone mark	Candidate State	Interpretation
H3K9me2,3	-	Silenced genes
H3K27me3	Inactive/poised promoter, polycomb repressed	Downregulation of nearby genes
H3K36me3	Transcriptional transition	Actively transcribed gene bodies.
H4K20me1	Transcriptional transition	Transcriptional activation
H3K4me1,2,3	Strong enhancer	Promoter of active genes
H3K27ac	Active promoter/strong enhancer	Active transcription
H3K9ac	Active promoter	Switch from transcription initiation to elongation.

EPIGENETIC JARGON CHEAT - SHEET

Regulatory Element	Meaning
Promoter	DNA Sequence (100-1kb), initial secure binding site for: RNA Pol complex Transfacs Adjacent regulated gene, defined relative to TSS. Poised: simultaneous activation/repressive histone mods.
Enhancer/Silencer	DNA Seq (50-1.5kb), bound by transfacs ( <i>activator / repressor</i> ) Can act on gene up to 1Mb away: DNA folding brings it close to promoter. Enhancer: Bound by activator, which interacts with complex initiating transcription. Silencer: bound by repressor, which interferes with GTF assembly.
Insulator	DNA, 300-2kb, Block enhancers from acting on promoters: positioned between enhancer and promoter, form chromatin-loop domains.
Polycomb-repressed	Polycomb – group proteins actively remodel chromatin to silence genes.

### The histone code

Then: go back and ask what fraction of classified regions contain peaks of a given type.

Ernst et al 2011

State	CTCF	H3K27me3	H3K9me3	H4K20me1	H3K4me1	H3K4me2	H3K4me3	H3K27ac	H3K9ac	WCE
1	16	2	2	6	17	93	99	96	98	2
2	12	2	6	9	53	94	95	14	44	1
3	15	79	0	9	48	75	49	1	10	1
4	11	1	15	11	96	99	75	67	98	4
5	5	0	10	3	88	57	5	84	25	1
6	7	1	1	3	58	76	8	6	5	1
7	2	1	2	1	66	3	0	6	2	1
8	92	2	1	3	6	3	0	0	1	1
9	5	0	43	43	37	11	2	9	4	1
10	1	0	47	3	0	0	0	0	0	1
11	0	0	3	2	0	0	0	0	0	0
12	1	27	0	2	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
14	22	28	19	41	5	5	28	5	13	37
15	85	85	91	88	76	77	91	73	85	78

Chromatin mark observation frequency (%)

Candidate state annotation

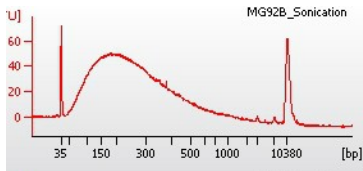
- Active promoter
- Weak promoter
- Weak/poised promoter
- Strong enhancer
- Strong enhancer
- Weak/poised enhancer
- Weak/poised enhancer
- Insulator
- Transcriptional transition
- Transcriptional elongation
- Weak transcribed
- Polycomb repressed
- Heterochrom; low signal
- Repetitive/CNV

### Histone mark cheat sheet

Histone mark	Candidate State	Interpretation
H3K9me2,3	-	Silenced genes
H3K27me3	Inactive/poised promoter, polycomb repressed	Downregulation of nearby genes
H3K36me3	Transcriptional transition	Actively transcribed gene bodies.
H4K20me1	Transcriptional transition	Transcriptional activation
H3K4me1,2,3	Strong enhancer	Promoter of active genes
H3K27ac	Active promoter/strong enhancer	Active transcription
H3K9ac	Active promoter	Switch from transcription initiation to elongation.

- ### ChIP-seq experimental considerations
- Antibody quality: 60% of antibodies not high enough quality
  - Numbers of cells: 2-3M recommended, more for TFs (5-10M)
  - Crosslinking time: ~10 mins
  - Shearing

### Shearing

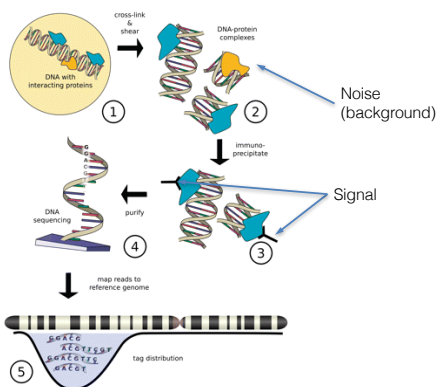


- Aim for fragments in 150-400bp range
- Efficiency varies by cell type
- Optimise by varying number of shearing cycles
- Run input samples on Bioanalyser to check efficiency

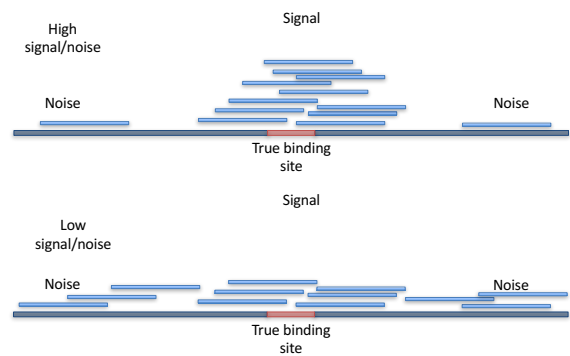
### ChIP-seq technical issues

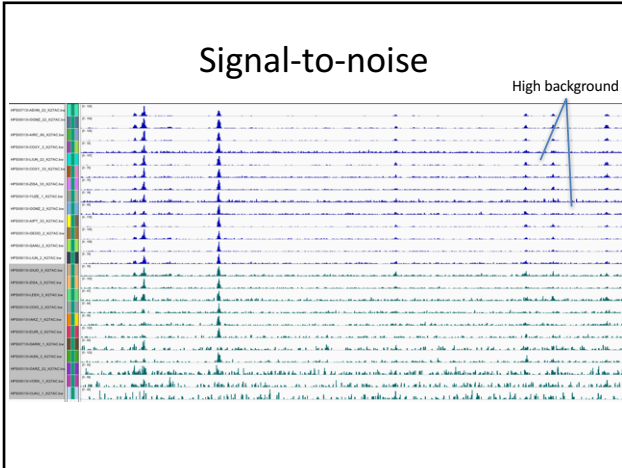
1. Signal / noise: Does my antibody work?
2. Library complexity: Did I have enough starting material?

### Signal / noise



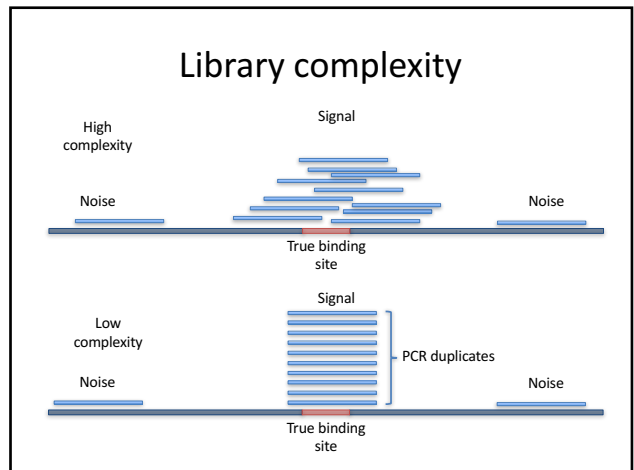
### Signal / noise



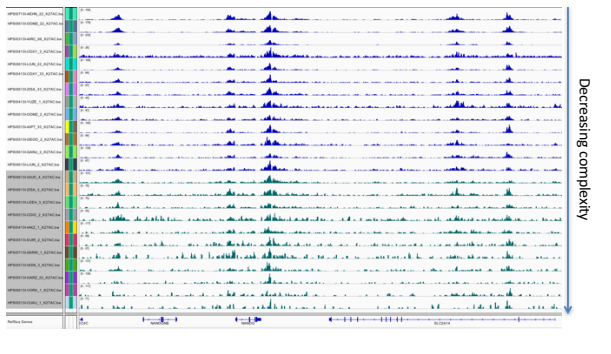


- ### FRIP
- Fragments In Peaks
  - # Fragments found in peaks / Total # fragments
  - >1%

- ### Library complexity
- Problem: Not enough starting material
    - Not enough cells
    - Antibody efficiency
  - More PCR required



### Library complexity



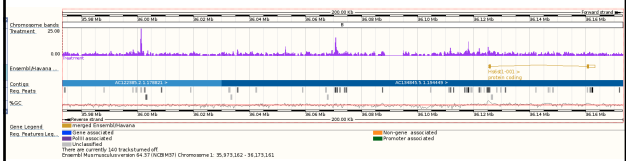
### Nonredundant fraction

- # unique fragments positions / total # fragments
- >0.8

### Basic analysis of CHIP-seq

1. Read alignment
2. Visualisation
3. Peak calling
  - Peak annotation (mapping peaks to genes etc)
  - Motif analysis
4. Differential binding
  - Case / control
  - Naïve / stimulated

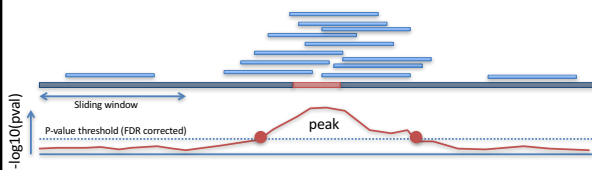
### Visualisation in a genome browser



- Convert mapped reads to “signal” – e.g. read depth at each bp or in windows
- BAM files to e.g. wig, bedgraph
- IGV, ensembl, UCSC



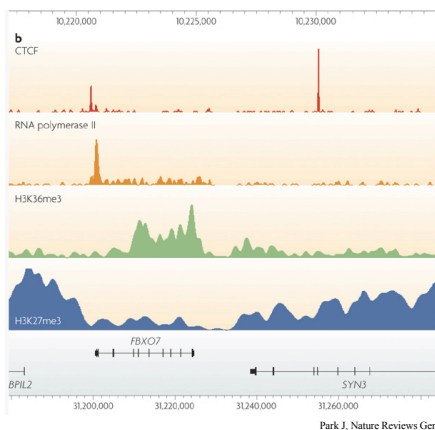
## Peak calling



- Observed counts
- Expected counts
- Poisson test: p-value – prob(observing frag count at least as extreme under null)

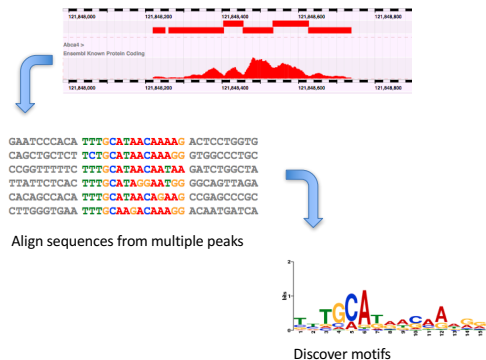
## Peak calling challenges

- What's expected?
  - Treatment sample (with antibody)
  - Input sample (no antibody)
- Replicates
  - Yes! (min 2, more = better)
- Peak sizes
  - These are variable: small for TFs, large for some Histone mods, and for Pol2 etc.



Park J, Nature Reviews Genetics, 2009

## Motif analysis



## 1 Introduction

ChIP-Seq is the combination of chromatin immunoprecipitation (ChIP) assays with high-throughput sequencing (Seq) and can be used to identify DNA binding sites for transcription factors and other proteins. The goal of this hands-on session is to perform the basic steps of the analysis of ChIP-Seq data, as well as some downstream analysis. Throughout this practical we will try to identify potential transcription factor binding sites of PAX5 in human lymphoblastoid cells.

### 1.1 Learning outcomes

By the end of this tutorial you can expect to be able to:

- generate an unspliced alignment by aligning raw sequencing data to the human genome using **Bowtie2**
- manipulate the SAM output in order to visualise the alignment in **IGV**
- based on the aligned reads, find immuno-enriched areas using the peak caller **MACS2**
- perform functional annotation and motif analysis on the predicted binding regions

### 1.2 Tutorial sections

This tutorial comprises the following sections:

1. [Introducing the tutorial dataset](#)
2. [Aligning the PAX5 sample to the genome](#)
3. [Manipulating SAM output](#)
4. [Visualising alignments in IGV](#)
5. [Aligning the control sample to the genome](#)
6. [Identifying enriched areas using MACS](#)
7. [File formats](#)
8. [Inspecting genomic regions using bedtools](#)
9. [Motif analysis](#)

### 1.3 Authors

This tutorial was converted into a Jupyter notebook by [Victoria Offord](#) based on materials developed by Angela Goncalves, Myrto Kostadima, Steven Wilder and Maria Xenophontos.

## 1.4 Running the commands from this tutorial

You can run the commands in this tutorial either directly from the Jupyter notebook (if using Jupyter), or by typing the commands in your terminal window.



```
pwd
```



```
ls -l
```

### 1.4.1 Running commands in the terminal

You can also follow this tutorial by typing all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, select the cell below with the mouse and then either press control and enter or choose Cell -> Run in the menu at the top of the page.



```
echo cd $PWD
```

Open a new terminal on your computer and type the command that was output by the previous cell followed by the enter key. The command will look similar to this:



```
cd /home/manager/pathogen-informatics-training/Notebooks/ChIP-Seq/
```

Now you can follow the instructions in the tutorial from here.

## 1.5 Prerequisites

This tutorial assumes that you have the following software or packages and their dependencies installed on your computer. The software or packages used in this tutorial may be updated from time to time so, we have also given you the version which was used when writing the tutorial.

Package	Link for download/installation instructions	Version tested
bedtools	<a href="http://bedtools.readthedocs.io/en/latest/content/installation.html">http://bedtools.readthedocs.io/en/latest/content/installation.html</a>	2.25.0
Bowtie2	<a href="http://bowtie-bio.sourceforge.net/bowtie2">http://bowtie-bio.sourceforge.net/bowtie2</a>	2.2.6
IGV	<a href="http://software.broadinstitute.org/software/igv">http://software.broadinstitute.org/software/igv</a>	2.3.8
MACS2	<a href="https://github.com/taoliu/MACS">https://github.com/taoliu/MACS</a>	2.1.0.20150420
meme	<a href="http://meme-suite.org/tools/meme">http://meme-suite.org/tools/meme</a>	4.10.0
samtools	<a href="https://github.com/samtools/samtools">https://github.com/samtools/samtools</a>	1.6
tomtom	<a href="http://web.mit.edu/meme_v4.11.4/share/doc/tomtom.html">http://web.mit.edu/meme_v4.11.4/share/doc/tomtom.html</a>	4.10.0
UCSC tools	<a href="http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64">http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64</a>	NA

## 1.6 Let's get started!

To get started with the tutorial, head to the first section: [introducing the tutorial dataset](#).

The answers to all questions in the tutorial can be found in [answers.ipynb](#).

## 2 Introducing the tutorial dataset

The data we will use for this practical comes from the [ENCODE \(Encyclopedia of DNA Elements\) Consortium](#), a big international collaboration aimed at building a comprehensive catalogue of functional elements in the human genome. As part of this project, many human tissues and cell lines were studied using high-throughput sequencing technologies.

In this tutorial, we will work on datasets from, [GM12878](#), a lymphoblastoid cell line produced from the blood of a female donor of European ancestry. Specifically, we will look at binding data for the transcription factor **PAX5**. PAX5 is a known regulator of B-cell differentiation. Aberrant expression of PAX5 is linked to lymphoblastoid leukaemia. If there is time, we will also look at ChIP-seq data for **Polymerase II** and the histone modification **H3K36me3**.

The .fastq file that we will align is called **PAX5.fastq**. This file is based on PAX5 ChIP-Seq data produced by the Myers lab in the context of the ENCODE project. We will align these reads to the human genome.

The tutorial files can be found in the data directory. Let's go there now!

**Move into the directory containing the tutorial data files.**



```
cd data
```

**Check to see if the tutorial files are there.**



```
ls *.fastq
```

**If the previous ls command didn't return anything, download and uncompress the tutorial files.**



```
wget ftp://ftp.sanger.ac.uk/pub/project/pathogens/workshops/chipseq_data.tar.gz
tar -xf chipseq_data.tar.gz
mv chipseq_data/* .
```

**Take a look at one of the FASTQ files.**



```
head PAX5.fastq
```

---

### 2.1 What's next?

For a quick recap of what the tutorial covers and the software you will need, head back to the [introduction](#).

Otherwise, let's get started with [aligning the PAX5 sample to the genome](#).

## 3 Aligning the PAX5 sample to the genome

There are a number of competing tools for short read alignment, each with its own set of strengths, weaknesses, and caveats. Here we will use **Bowtie2**, a widely used ultrafast, memory efficient short read aligner.

Bowtie2 has a number of parameters in order to perform the alignment. To view them all type:



```
bowtie2 -help
```

Bowtie2 uses indexed genome for the alignment in order to keep its memory footprint small. Because of time constraints we will build the index only for one chromosome of the human genome. For this we need the chromosome sequence in fasta format. This is stored in a file named `HS19.fa`, under the subdirectory `genome`.

**If you are not in there already, change into the data directory.**



```
cd data
```

We will be storing our indexed genome in a folder called `bowtie_index`.

**Check if the `bowtie_index` folder already exists.**



```
ls bowtie_index
```

**If it doesn't exist already, create the folder `bowtie_index`.**



```
mkdir bowtie_index
```

**Then, index the chromosome using the command:**



```
bowtie2-build genome/HS19.fa bowtie_index/hs19
```

Be patient, building the index may take 5-10 minutes!

This command will output 6 files that constitute the index. These files that have the prefix **hs19** and are stored in the `bowtie_index` directory.

**To check the files have been successfully created type:**



```
ls -l bowtie_index
```

Now that the genome is indexed we can move on to the actual alignment. In the following command the first argument (`-k`) instructs Bowtie2 to report only uniquely mapped reads. The following argument (`-x`) specifies the basename of the index for the genome to be searched; in our case is **hs19**. Then there is the name of the FASTQ file and the last argument (`-S`) that ensures that the output is in SAM format.

### Align the PAX5 reads using Bowtie2:



```
bowtie2 -k 1 -x bowtie_index/hs19 PAX5.fastq -S PAX5.sam
```

The above command outputs the alignments in **SAM** format and stores them in the file `PAX5.sam`.

In general before you run Bowtie2, you have to know which FASTQ format you have. The available FASTQ formats in Bowtie2 are:

```
--phred33 input quals are Phred+33 (default)
--phred64 input quals are Phred+64
--int-quals input quals are specified as space-delimited integers
```

See [http://en.wikipedia.org/wiki/FASTQ\\_format](http://en.wikipedia.org/wiki/FASTQ_format) to find more detailed information about the different quality encodings.

The `PAX5.fastq` file we are working on uses encoding **Phred+33** (the default). Bowtie2 will take 2-3 minutes to align the file. This is fast compared to other aligners that sacrifice some speed to obtain higher sensitivity.

### Look at the file in the SAM format by typing:



```
head -n 10 PAX5.sam
```

You can find more information on the SAM format by looking at <https://samtools.github.io/hts-specs/SAMv1.pdf>.

---

## 3.1 Questions

**Q1. How can you distinguish between the header of the SAM format and the actual alignments?**

*Hint: look at section 1.3 in the documentation (<https://samtools.github.io/hts-specs/SAMv1.pdf>).*

**Q2. What information does the header provide you with?**

*Hint: use the documentation to work out what the header tags mean*

**Q3. Which chromosome are the reads mapped to?**

---

## 3.2 What's next?

For a quick recap of what the tutorial covers head back to the [introduction](#).

If you want a reintroduction to the tutorial dataset, head back to [introducing the tutorial dataset](#).

Otherwise, let's continue on to [manipulating SAM output](#).

## 4 Manipulating SAM output

SAM files are rather big and when dealing with a high volume of HTS data, storage space can become an issue. Using [samtools](#) we can convert SAM files to BAM files (their binary equivalent files that are not human readable) that occupy much less space.

To convert your SAM file to a BAM file, you have to instruct `samtools` that the input is in SAM format (`-S`), the output should be in BAM format (`-b`) and that you want the output to be stored in the file specified by the `-o` option.

**If you are not in there already, change into the data directory.**



```
cd data
```

**Convert SAM to BAM using `samtools` and store the output in the file `PAX5.bam`:**



```
samtools view -bSo PAX5.bam PAX5.sam
```

---

### 4.1 What's next?

For a quick recap of what the tutorial covers head back to the [introduction](#).

If you want a reintroduction to the tutorial dataset, head back to [aligning the PAX5 sample to the genome](#).

Otherwise, let's continue on to [visualising alignments in IGV](#).



## 5 Visualising alignments in IGV

It is often instructive to look at your data in a genome browser. Here, we use [IGV](#), a stand-alone browser, which has the advantage of being installed locally and providing fast access. Please check their website (<http://www.broadinstitute.org/igv>) for all the formats that IGV can display.

Web-based genome browsers, like [Ensembl](#) or the [UCSC browser](#), are slower, but provide more functionality. They do not only allow for more polished and flexible visualisation, but also provide easy access to a wealth of annotations and external data sources. This makes it straightforward to relate your data with information about repeat regions, known genes, epigenetic features or areas of cross-species conservation, to name just a few. As such, they are useful tools for exploratory analysis.

Visualisation will allow you to get a "feel" for the data, as well as detecting abnormalities and problems. Also, exploring the data in such a way may give you ideas for further analyses. For our visualization purposes we will use the BAM and bigWig formats.

When uploading a BAM file into the genome browser, the browser will look for the **index** of the BAM file in the same folder where the BAM files is. The index file should have the same name as the BAM file and the suffix `.bai`. Finally, to create the index of a BAM file you need to make sure that the file is **sorted** according to chromosomal coordinates.

**If you are not in there already, change into the data directory.**



```
cd data
```

**Sort alignments according to chromosome position and store the result in the file with the prefix PAX5.sorted:**



```
samtools sort -T PAX5.temp.bam -o PAX5.sorted.bam PAX5.bam
```

**Index the sorted file.**



```
samtools index PAX5.sorted.bam
```

The indexing will create a file called `PAX5.sorted.bam.bai`. Note that you don't have to specify the name of the index file when running `samtools index`.

Another way to visualise the alignments is to convert the BAM file into a **bigWig** file. The bigWig format is for display of dense, continuous data and the data will be displayed as a graph. The resulting bigWig files are in an indexed binary format.

The BAM to bigWig conversion takes place in two steps. First, we convert the BAM file into a bedgraph, called `PAX5.bedgraph`, using the tool `genomeCoverageBed` from [bedtools](#).

**To find the structure of the command and the mandatory arguments type:**



```
genomeCoverageBed
```

Apart from the BAM file, we also need to provide the size of the chromosomes for the organism of interest in order to generate the bedgraph file. These have to be stored in a tab-delimited file. When using the UCSC Genome Browser, Ensembl, or Galaxy, you typically indicate which species or genome build you are working with. The way you do this for bedtools is to create a "genome" file, which simply lists the names of the chromosomes (or scaffolds, etc.) and their size (in basepairs).

**To obtain chromosome lengths for the human genome, type:**



```
fetchChromSizes hg19 > genome/hg19.all.chrom.sizes
```

We next want to remove any chromosome length information for the patched chromosomes, which are accessioned scaffold sequences that represent assembly updates. That way we will only keep the information of the current assembly.

**Remove this information using awk:**



```
awk '$1 !~ /[_.]/' genome/hg19.all.chrom.sizes > genome/hg19.chrom.sizes
```

**Now generate the bedgraph file, called PAX5.bedgraph, by typing:**



```
genomeCoverageBed -bg -ibam PAX5.sorted.bam \  
-g genome/hg19.chrom.sizes > PAX5.bedgraph
```

We then need to convert the bedgraph into a binary graph, called PAX5.bw, using the tool bedGraphToBigWig from the UCSC tools.

**To convert the bedgraph type:**



```
bedGraphToBigWig PAX5.bedgraph genome/hg19.chrom.sizes PAX5.bw
```

Now we will load the data into the IGV browser for visualisation.

**To launch IGV :**



```
igv.sh &
```

**On the top left of your screen choose "Human hg19" from the drop down menu. Then in order to load the desired files go to "File -> Load from File".**

**On the pop up window navigate to the tutorial folder and select the file PAX5.sorted.bam.**

**Repeat these steps in order to load PAX5.bw as well.**

**Select "chr1" from the drop down menu on the top left.**

**Right click on the name of PAX5.bw and choose "Maximum" under the "Windowing Function".**

**Right click again and select "Autoscale".**

## 5.1 Questions

**Q1. Look for gene NASP in the search box. Can you see a PAX5 binding site near the NASP gene?**

*Hint: use the "+" button on the top right zoom in more to see the details of the alignment*

**Q2. What is the main difference between the visualisation of BAM and bigWig files?**

---

## 5.2 What's next?

You can head back to [manipulating SAM output](#) or continue on to [aligning the control sample to the genome](#).

## 6 Aligning the control sample to the genome

In the ChIP-Seq folder you will find another `.fastq` file called `Control.fastq`.

If you are not in there already, change into the data directory.



```
cd data
```

Use the `head` command to look at this file:



```
head Control.fastq
```

Use the information on the FASTQ Wikipedia page ([http://en.wikipedia.org/wiki/FASTQ\\_format](http://en.wikipedia.org/wiki/FASTQ_format)) to determine the quality encoding this FASTQ file is using. Then, adapting your commands to the quality encoding where needed, follow the steps you used to align the PAX5 sample to the genome and manipulate the SAM file in order to align the control reads to the human genome.

---

### 6.1 What's next?

You can head back to [visualising alignments in IGV](#) or continue on to [identifying enriched areas using MACS](#).

## 7 Finding enriched areas using MACS

**MACS2** stands for **m**odel-based **a**nalysis of **C**hIP-**S**eq. It was designed for identifying transcription factor binding sites. MACS2 captures the influence of genome complexity to evaluate the significance of enriched ChIP regions, and improves the spatial resolution of binding sites through combining the information of both sequencing tag position and orientation. MACS2 can be easily used for ChIP-Seq data alone, or with a control sample to increase specificity.

**If you are not in there already, change into the data directory.**



```
cd data
```

**Consult the MACS2 help file to see the options and parameters:**



```
macs2 -help
```



```
macs2 callpeak -help
```

The input for MACS2 can be in ELAND, BED, SAM, BAM or BOWTIE formats (you just have to set the `--format` flag).

Options that you will have to use include:

`-t` to indicate the input ChIP file

`-c` to indicate the name of the control file

`--format` the tag file format

(if this option is not set MACS automatically detects which format the file is)

`--name` to set the name of the output files

`--gsize` to set the mappable genome size

(with the read length we have, 70% of the genome is a fair estimation)

`--call-summits` to detect all subpeaks in each enriched region and return their summits

`--pvalue` the P-value cutoff for peak detection.

Now run macs using the following command:



```
macs2 callpeak -t PAX5.sorted.bam -c Control.sorted.bam \  
-format BAM -name PAX5 -gsize 138000000 -pvalue 1e-3 \  
-call-summits
```

MACS2 generates its peak files in a file format called `.narrowPeak` file. This is a **BED** format describing genomic locations. Many types of genomic data can be represented as (sets of) genomic regions. In the following section we will look into the BED format in more detail, and we will perform simple operations on genomic interval data.

---

## 7.1 What's next?

You can head back to [aligning the control sample to the genome](#) or continue on to [file formats](#).

## 8 File Formats

### 8.1 BED files

Over the years a set of commonly used file formats for genomic intervals have emerged. Most of these file formats are tabular where each row consists of an interval and columns have a pre-defined meaning, describing chromosomes, locations, scores, etc. The UCSC web browser has an informative list of these at <http://genome.ucsc.edu/FAQ/FAQformat.html>.

The **BED** format is the simplest file format of these. A minimal bed file has at least three columns denoting **chromosome**, **start** and **end** of an interval. The following example denotes three intervals, two on chromosome chr1 and one on chr2.

chromosome	start	end
chr1	50	100
chr1	500	1000
chr2	600	800

BED files follow the UCSC Genome Browser's convention of making the start position **0-based** and the end position **1-based**. In other words, you should interpret the "start" column as being 1 base pair higher than what is represented in the file. For example, the following BED feature represents a single base on chromosome 1; namely, the 1st base.

chromosome	start	end	description
chr1	0	1	I-am-the-first-position-on-chrom-1

Using the bed format documentation found at <http://genome.ucsc.edu/FAQ/FAQformat.html#format1> answer the following questions.

#### 8.1.1 Questions

**Q1. The simplest bed file contains just three columns (chromosome, start, end) and is often called BED3 format. What extra columns does BED6 contain?**

*Hint: look for information about columns 4 to 6 in the documentation <http://genome.ucsc.edu/FAQ/FAQformat.html#format1>*

**Q2. In the above examples, what are the lengths of the intervals?**

**Q3. Can you output a BED6 format with a transcript called "loc1", transcribed on the forward strand and having three exons of length 100 starting at positions 1000, 2000 and 3000?**

*Hint: you will need one line per exon*

## 8.2 narrowPeak files

The narrowPeak format is a BED6+4 format used to describe and visualise called peaks. Previously, we have used MACS2 to call peaks on the PAX5 ChIP-seq data set.

If you are not in there already, change into the data directory.



```
cd data
```

View the first 10 lines in PAX5\_peaks.narrowPeak using the head command:



```
head -10 PAX5_peaks.narrowPeak
```

NarrowPeak files can also be uploaded to IGV or other genome browsers.

Try uploading the peak file generated by MACS2 to IGV.

### 8.2.1 Questions

**Q4.** What additional information is given in the narrowPeak file, beside the location of the peaks?

*Hint: See <http://genome.ucsc.edu/FAQ/FAQformat.html#format12> for details*

**Q5.** Does the first peak that was called look convincing to you?

## 8.3 GTF files

A second popular format is the GTF format. Each row in a GTF formatted file denotes a genomic interval. The GTF format documentation can be found at <http://mblab.wustl.edu/GTF2.html>.

The three intervals from above might be:

seqid	source	type	start	stop	score	strand	phase	attributes
chr1	gene	exon	51	100	.	+	0	gene_id "001";transcript_id "001.1";
chr1	gene	exon	501	1000	.	+	2	gene_id "001";transcript_id "001.1";
chr2	repeat	exon	601	800	.	+	.	

The 9th column permits intervals to be grouped and linked in a hierarchical fashion. This format is thus popular to describe gene models. Note how the first two intervals are linked through a common transcript\_id and gene\_id.

The aim of the [GENCODE project](#) is to annotate all evidence-based genes and gene features in the entire human genome at a high accuracy. Annotation of the GENCODE gene set is carried out using a mix of manual annotation, experimental analysis and computational biology methods.



The GENCODE v18 geneset is available in the genome folder.

**Look at the first 10 lines of the GENCODE annotation file:**



```
head -n 10 genome/gencode.v18.annotation.gtf
```

---

### 8.3.1 Questions

**Q6.** In the small example table above, why have the coordinates changed from the BED description?

---

## 8.4 What's next?

You can head back to [identifying enriched areas using MACS](#) or continue on to [inspecting genomic regions using bedtools](#).

## 9 Inspecting genomic regions using bedtools

In this section we perform simple functions, such as overlaps, on the most common file type used for describing genomic regions, the **BED** file. We will examine the results of the ChIP-Seq peak calling you have performed on the transcription factor PAX5 and perform simple operations on these files, using the **bedtools** suite of programs. You will then annotate the MACS2 peaks with respect to genomic annotations. Finally, we will select the most significantly enriched peaks, and extract the genomic sequence flanking their summits, the point of highest enrichment.

**If you are not in there already, change into the data directory.**



```
cd data
```

The **bedtools** package permits complex, interval-based manipulation of BED and GTF files. They are also very fast. The general invocation of **bedtools** is `bedtools <COMMAND>`.

**To get an overview of the available commands, simply call `bedtools` without any command or options in the terminal window.**



```
bedtools
```

To get help for a command, type `bedtools <COMMAND>`. Extensive documentation and examples are available at <https://bedtools.readthedocs.org/en/latest/>. We will now use **bedtools** to calculate simple coverage statistics of the peak calls over the genome (keep in mind that only peaks on Chromosome 1 are in the file).

**To bring up the help page for the `bedtools genomecov` command, type:**



```
bedtools genomecov
```

**Calculate the genome coverage of the PAX5 peaks:**



```
bedtools genomecov -i PAX5_peaks.narrowPeak -g genome/hg19.chrom.sizes
```

In order to biologically interpret the results of ChIP-Seq experiments, it is useful to look at the genes and other annotated elements that are located in proximity to the identified enriched regions. We will now use **bedtools** to identify how many PAX5 peaks overlap GENCODE genes.

**First we use `awk` to filter out only the genes from the GTF file:**



```
awk '$3=="gene"' genome/gencode.v18.annotation.gtf \  
> genome/gencode.v18.annotation.genes.gtf
```

**Next, count the total number of PAX5 peaks:**



```
wc -l PAX5_peaks.narrowPeak
```

Then use bedtools to find the number overlapping GENCODE genes:



```
bedtools intersect -a PAX5_peaks.narrowPeak \
  -b genome/gencode.v18.annotation.genes.gtf | wc -l
```

You can use the bedtools closest command to find the closest gene to each peak.



```
bedtools closest -a PAX5_peaks.narrowPeak \
  -b genome/gencode.v18.annotation.genes.gtf | head
```

Transcription factor binding near to the **transcript start sites (TSS)** of genes is known to drive gene expression or repression, so it is of interest to know which TSS regions are bound by PAX5. To determine this, we will first create a BED file of the GENCODE TSS using the GTF.

You can use this awk command to create the TSS BED file:



```
awk 'BEGIN {FS=OFS="\t"} { if($7=="+") {tss=$4-1} else { tss = $5 } } \
  print $1,tss, tss+1, ".", ".", $7, $9}' \
  genome/gencode.v18.annotation.genes.gtf > genome/gencode.tss.bed
```

Now use the bedtools closest command again to find the closest TSS to each peak:



```
sortBed -i genome/gencode.tss.bed > genome/gencode.tss.sorted.bed
bedtools closest -a PAX5_peaks.narrowPeak \
  -b genome/gencode.tss.sorted.bed > PAX5_closestTSS.txt
```



Use head to inspect the results:



```
head PAX5_closestTSS.txt
```

You have now matched up all the PAX5 transcription factor peaks to their nearest gene transcription start site.

## 9.1 Questions

Q1. Looking at the output of the bedtools genomecov we ran, what percentage of chromosome 1 do the peaks of PAX5 cover?

Q2. Looking at the output from bedtools intersect, what proportion of PAX5 peaks overlap genes?

Q3. Looking at PAX5\_closestTSS.txt, which gene was found to be closest to MACS peak 2?

## 9.2 What's next?

You can head back to [file formats](#) or continue on to [motif analysis](#).

## 10 Motif analysis

It is often interesting to find out whether we can associate the identified binding sites with a sequence pattern or motif. To do so, we will identify the summit regions of the strongest PAX5 binding sites, retrieve the sequences associated with these regions, and use **MEME** for motif analysis.

Since many peak-finding tools merge overlapping areas of enrichment, the resulting peaks tend to be much wider than the actual binding sites. The summit and its vicinity are the best estimate for the true protein binding site, and so it is here where we look for repeated sequence patterns, called motifs, to which the transcription factor may preferentially bind.

Sub-dividing the enriched areas by accurately partitioning enriched loci into a finer-resolution set of individual binding sites, and fetching sequences from the summit region where binding motifs are most likely to appear enhances the quality of the motif analysis. Sub-peak summit sequences have already been called by MACS2 with the `--call-summits` option.

*De novo* motif finding programs take as input a set of sequences in which to search for repeated short sequences. Since motif discovery is computationally heavy, we will restrict our search for the Oct4 motif to the genome regions around the summits of the 300 most significant PAX5 subpeaks on Chromosome 1.

**If you are not in there already, change into the data directory.**



```
cd data
```

**Sort the PAX5 peaks by the height of the summit (the maximum number of overlapping reads).**



```
sort -k5 -nr PAX5_summits.bed > PAX5_summits.sorted.bed
```

**Using the sorted file, select the top 300 peaks and create a BED file for the regions of 60 base pairs centred around the peak summit.**



```
awk 'BEGIN{FS=OFS="\t"}; NR < 301 { print $1, $2-30, $3+29 }' \
PAX5_summits.sorted.bed > PAX5_top300_summits.bed
```

The human genome sequence is available in FASTA format in the `bowtie_index` directory.

**Use `bedtools` to extract the sequences around the PAX5 peak summits in FASTA format, which we save in a file named `PAX5_top300_summits.fa`.**



```
bedtools getfasta -fi genome/HS19.fa \
-bed PAX5_top300_summits.bed -fo PAX5_top300_summits.fa
```

We are now ready to perform de novo motif discovery, for which we will use the tool **MEME**.

Open a web browser, go to the MEME website at <http://meme-suite.org/>, and choose the "MEME" tool.

Fill in the necessary details, such as:

- the sub-peaks fasta file PAX5\_top300\_summits.fa (will need uploading), or just paste in the sequences.
- the number of motifs we expect to find (1 per sequence)
- the width of the desired motif (between 6 to 20) in the "Advanced" options
- the maximum number of motifs to find (3 by default).

For PAX5 one classical motif is known.

### Start Search.

Your MEME analysis will now be queued and will run on a server in the US. The results page will refresh automatically and once the tool has finished running there will be a link to the results. Depending on how busy the servers are your analysis may take a longer or shorter time to run.

You can check the load of the server here:

<http://meme-suite.org/opal2/dashboard?command=statistics>

## 10.1 Analyse the results from MEME

We would like to know if this motif is similar to any other known motif. We will use the results from **TOMTOM** for this.

On either the results from the web MEME run or the local run please follow the link "MEME html output". Scroll down until you see the first motif logo.

Click under the option Submit/Download and choose the TOMTOM button to compare to known motifs in motif databases, and on the new page choose to compare your motif to those in the JASPAR CORE and UniPROBE Mouse database.

## 10.2 Running MEME locally

If you want to speed things up you may want to run MEME on your own machine. You can try to do this as well if you wish, or skip the following bonus exercise and go to the next section.

To bring up the help page for the local installation of MEME, type:



```
meme
```

Run MEME locally, setting the output directory with the option `-o` (e.g. `-o meme_out`).



```
meme PAX5_top300_summits.fa -o meme_out -dna -nmotifs 1 -minw 6 -maxw 20
```

Once MEME has finished running look in this directory for the file `meme.html` and open it in a web browser. You can do this by either copying the path to the file to the address bar in Firefox or double click on the `.html` file.

Alternatively, you can run the following command to automatically open the HTML file in Firefox:



```
firefox meme_out/meme.html
```

Scroll down until you see the first motif logo.

We would like to know if this motif is similar to any other known motif. We will use **TOMTOM** and a set of known motif databases stored in `motif_databases` for this.

To compare your newly found motifs to the motif databases JASPAR CORE and UniPROBE Mouse you can run:



```
tomtom -o tomtom_out meme_out/meme.html \  
motif_databases/JASPAR/JASPAR_CORE_2016_vertbrates.meme \  
motif_databases/MOUSE/uniprobe_mouse.meme
```

Once again, once TOMTOM has finished running look in `tomtom_out` for the file `tomtom.html`.

Open `tomtom.html` in a web browser.



```
firefox tomtom_out/tomtom.html
```

---

## 10.3 Questions

Q1. Which motif was found to be the most similar to your motif?

---

## 10.4 Congratulations, you have reached the end of this tutorial!

We hope you've enjoyed our ChIP-Seq tutorial. You can find the answers to all of the questions in this tutorial in [answers.ipynb](#). You can revisit [inspecting genomic regions using bedtools](#) or, go back to the [beginning](#).

# Differential Expression using RNA-Seq

Victoria Offord  
WTC NGS Bioinformatics  
Johannesburg 2019

## Overview

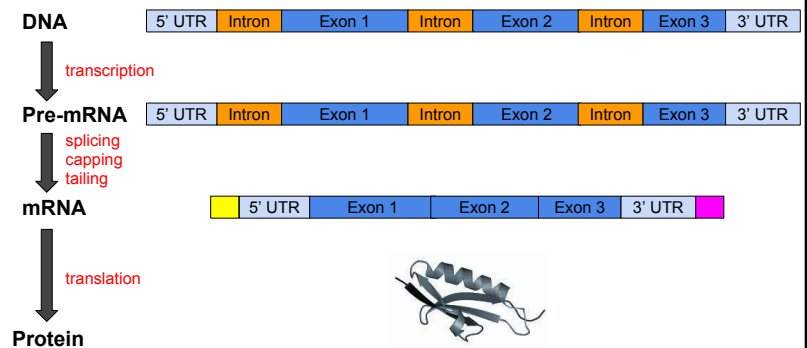
- RNA-seq background
- Mapping to the genome (HISAT2 and IGV)
- Mapping to the transcriptome and counting reads (Kallisto)
- Read count normalisation
- Differential expression and QC (Sleuth)
- What to do with a gene list
- The exercise

What is the transcriptome?

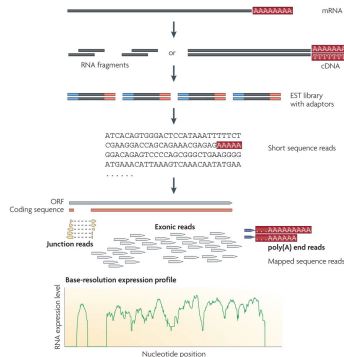
*“The complete set of transcripts in a cell  
and their quantity  
for a specific developmental stage or condition”*

Wang *et al.* (2009)  
Nature Reviews Genetics  
(PubMed: 19015660)

## Central dogma



## RNA Sequencing



Wang *et al.* (2009)  
Nature Reviews Genetics  
(PubMed: 19015660)

## Experimental design

- Successful RNA-Seq studies start with a good study design
- Considerations for generating data to answer your biological question include:
  - library type
  - sequencing depth
  - number of replicates
  - avoiding biases

## Experimental design - library preparation

- Total RNA = mRNA + rRNA + tRNA + regulatory RNAs...
- Ribosomal RNA can represent > 90% total RNA
- Can enrich for the 1-2% mRNA or deplete rRNA
  - enrichment typically needs good RIN and high RNA proportion
  - some samples (e.g. tissue biopsies) may not be suitable
  - bacterial mRNA not polyadenylated -> ribosomal depletion
- Be aware of protocol being used (e.g. some will remove small RNAs)

## Experimental design - library type

- **Stranded vs unstranded**
  - strand-specific protocols better for detangling antisense or overlapping transcripts
- **Single or paired end**
  - paired end better for *de novo* transcript discovery or isoform expression analysis
  - < 55% reads will span 2 or more exons



## Experimental design - replicates

### Biological replicates

- biologically distinct samples
- same type of organism treated or grown in the same condition
- understand biological variation (e.g. variation between individuals)
- relevant biological replicates are required

### Technical replicates

- repeated measurements of the same sample
- understand the variation in equipment or protocols
- technical replicates are not generally required, but try to arrange samples on plates to minimise potential problems

## Experimental design - sequencing depth / replicates

**BIOINFORMATICS** **DISCOVERY NOTE** Vol. 30 no. 3 2014, pages 301–304  
doi:10.1093/bioinformatics/btt688

*Gene expression*

Advance Access publication December 6, 2013

### RNA-seq differential expression studies: more sequence or more replication?

Yuwen Liu<sup>1,2</sup>, Jie Zhou<sup>1,3</sup> and Kevin P. White<sup>1,2,3,\*</sup>

<sup>1</sup>Institute of Genomics and Systems Biology, <sup>2</sup>Committee on Development, Regeneration, and Stem Cell Biology and <sup>3</sup>Department of Human Genetics, University of Chicago, Chicago, IL 60637, USA

Associate Editor: Janet Kelso

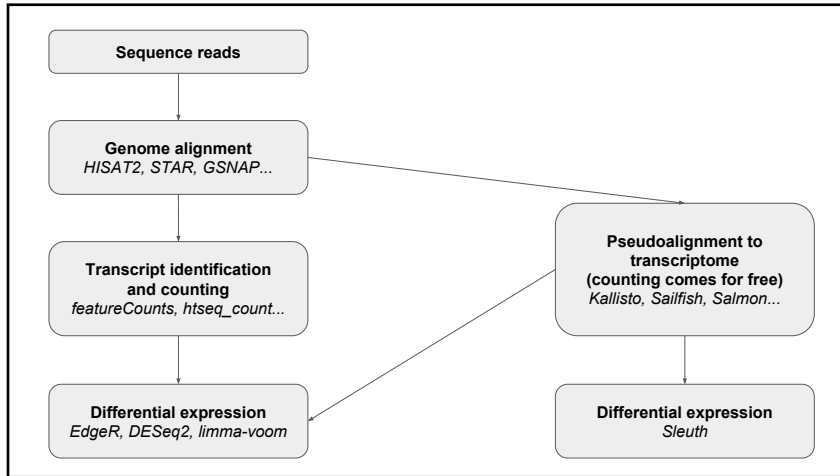
- reduces the coefficient of variation

## What do we need to know?

1. Which genes/transcripts do our reads belong to? **mapping**
2. How many reads align to a specific gene/transcript? **quantification**
3. Do different sample groups express genes/transcripts differently? **DGE analysis**

No universal pipeline to cover every analysis!!!

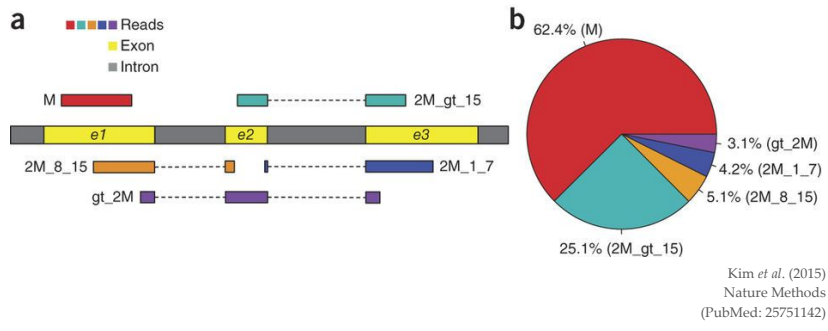




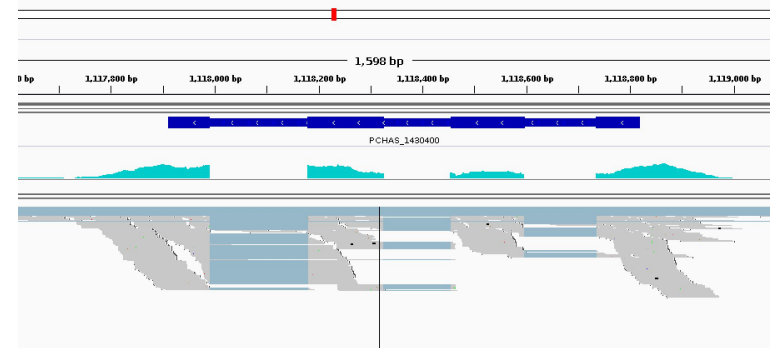
## Mapping RNA-seq reads to the genome (HISAT2)

- Mapping to the genome is great for determining whether your RNA-seq data is of high quality and exploring the structure of genes of interest
- Eukaryotic genes have introns, which are not present in mature mRNA so special mapping algorithms are required (splice-aware)
- **HISAT2** is only one such algorithm, but is accurate, fast and easy to use

## Splice aware alignment



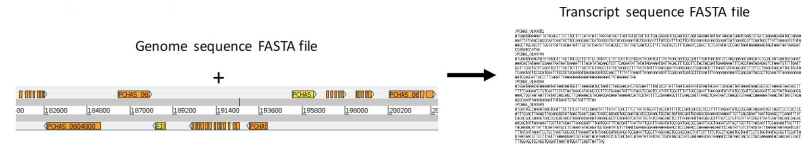
## Visualisation: Integrative Genomics Viewer (IGV)



## Mapping to the transcriptome and counting reads (Kallisto)

- Multiple splice forms per gene introduce ambiguity into the mapping
- Mapping to the spliced transcript sequences allows this ambiguity to be taken into account and allows transcript-specific read counts
- It is also faster because there is less target sequence
- Recent improvements in algorithms (pseudalignment) make this even faster
  - doesn't care where in each transcript reads map to, just which of the transcripts they map to
- Counting comes for free

## Mapping to the transcriptome and counting reads (Kallisto)

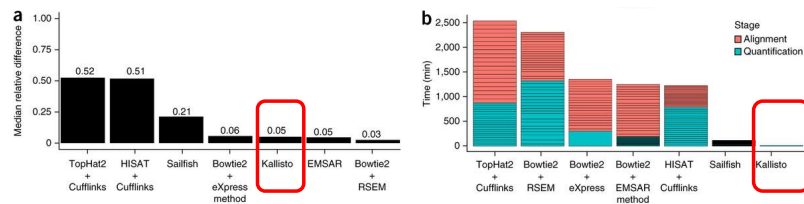


Kallisto has two steps:

1. Building an index from the spliced transcript sequences
2. Quantify reads against the index

Kallisto cannot be used to identify novel transcripts

## Mapping to the transcriptome and counting reads (Kallisto)



Bray *et al.* (2016)  
Nature Biotechnology  
(PubMed: 27043002)

## Normalisation

- Runs with more depth will have more reads mapping to each gene (**sequencing depth bias**)
- Longer genes will have more reads mapping to them (**gene length bias**)
- Most methods will normalise for sequencing depth and gene length

## Normalisation methods

- **RPKM** - reads per kilobase per million
- **FPKM** - fragments per kilobase per million
- **TPM** - transcripts per million

Some of these methods have problems with highly expressed genes, so it's better to use more complicated normalisation procedures (**DESeq2 rlog**, **Sleuth**)

Adapted from StatQuest (<http://statquest.org>)

## RPKM

B E F O R E	Gene	Replicate 1 Counts	Replicate 2 Counts	Replicate 3 Counts
	A (2,000 bases)	10	12	30
	B (4,000 bases)	20	25	60
	C (1,000 bases)	5	8	15
	D (10,000 bases)	0	0	1

A F T E R	Gene (bases)	Replicate 1 RPKM	Replicate 2 RPKM	Replicate 3 RPKM
	A (2,000 bases)	1.43	1.33	1.42
	B (4,000 bases)	1.43	1.39	1.42
	C (1,000 bases)	1.43	1.78	1.42
	D (10,000 bases)	0	0	0.009

## FPKM (fragments per kilobase million)

- RPKM for paired reads
- takes into account that two reads can map to one fragment (and so it doesn't count this fragment twice)



## RPKM vs TPM

RPKM				TPM			
Gene	R1	R2	R3	Gene	R1	R2	R3
A	1.43	1.33	1.42	A	3.33	2.96	3.326
B	1.43	1.39	1.42	B	3.33	3.09	3.326
C	1.43	1.78	1.42	C	3.33	3.95	3.326
D	0	0	0.009	D	0	0	0.02
<b>Total</b>	<b>4.29</b>	<b>4.5</b>	<b>4.25</b>	<b>Total</b>	<b>10</b>	<b>10</b>	<b>10</b>

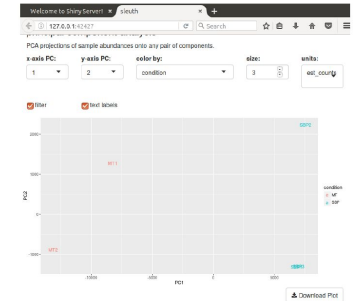
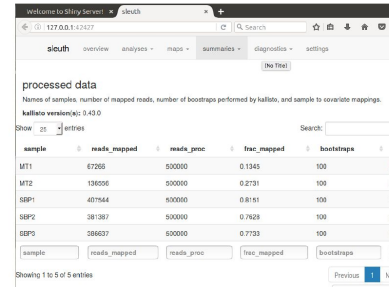
Easier to see the proportion of each gene within a sample as sum of TPMs same across samples

Adapted from StatQuest (<http://statquest.org>)

## Determining differential expression (Sleuth)

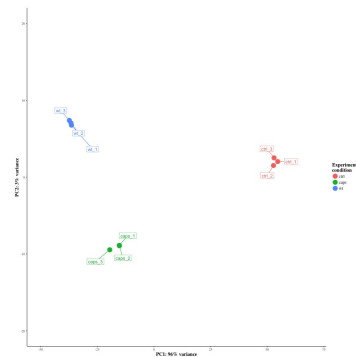
- We don't normally have enough replicates to do traditional tests of significance for RNA-seq data
- Most methods look for outliers in the relationship between average abundance and fold change
- Assume most genes are not differentially expressed

## QC with Sleuth

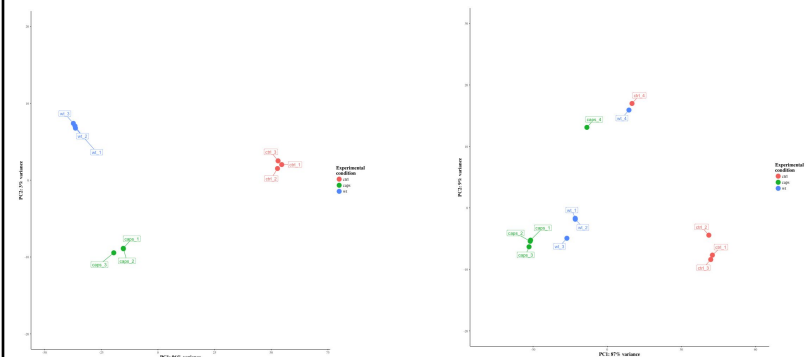


## Principal component analysis (PCA)

- Use to look at variation and strong patterns within data
- Identifies uncorrelated variables or principal components (PC)
- Tries to explain the maximum amount of variance with the smallest number of principal components



## Why QC our data?





# 1 RNA-Seq expression analysis

## 1.1 Introduction

RNA sequencing (**RNA-Seq**) is a high-throughput method used to profile the **transcriptome**, quantify gene expression and discover novel RNA molecules. This tutorial uses RNA sequencing of **malaria parasites** to walk you through transcriptome visualisation, performing simple quality control checks and will show you how to profile transcriptomic differences by identifying differentially expressed genes.

For an introduction to RNA-Seq principles and best practices see:

### **A survey of best practices for RNA-Seq data analysis**

Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Micha Wojciech Szczyniak, Daniel J. Gaffney, Laura L. Elo, Xuegong Zhang and Ali Mortazavi  
*Genome Biol.* 2016 Jan 26;17:13 doi:[10.1186/s13059-016-0881-8](https://doi.org/10.1186/s13059-016-0881-8)

## 1.2 Learning outcomes

By the end of this tutorial you can expect to be able to:

- Align RNA-Seq reads to a reference genome and a transcriptome
- Visualise transcription data using standard tools
- Perform QC of NGS transcriptomic data
- Quantify the expression values of your transcripts using standard tools

## 1.3 Tutorial sections

This tutorial comprises the following sections:

1. [Introducing the tutorial dataset](#)
2. [Mapping RNA-Seq reads to the genome with HISAT2](#)
3. [Visualising transcriptomes with IGV](#)
4. [Transcript quantification with Kallisto](#)
5. [Identifying differentially expressed genes with Sleuth](#)
6. [Interpreting the results](#)
7. [Key aspects of differential expression analysis](#)

## 1.4 Authors

This tutorial was written by [Victoria Offord](#) based on materials from [Adam Reid](#).

## 1.5 Running the commands from this tutorial

You can run the commands in this tutorial either directly from the Jupyter notebook (if using Jupyter), or by typing the commands in your terminal window.

### 1.5.1 Running commands on Jupyter

If you are using Jupyter, command cells (like the one below) can be run by selecting the cell and clicking *Cell -> Run* from the menu above or using *Ctrl Enter* to run the command. Let's give this a try by printing our working directory using the `pwd` command and listing the files within it. Run the commands in the two cells below.



```
pwd
```



```
ls -l
```

### 1.5.2 Running commands in the terminal

You can also follow this tutorial by typing all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, select the cell below with the mouse and then either press control and enter or choose *Cell -> Run* in the menu at the top of the page.



```
echo cd $PWD
```

Open a new terminal on your computer and type the command that was output by the previous cell followed by the enter key. The command will look similar to this:



```
cd /home/manager/pathogen-informatics-training/Notebooks/RNA-Seq/
```

Now you can follow the instructions in the tutorial from [here](#).



## 1.6 Prerequisites

This tutorial assumes that you have the following software or packages and their dependencies installed on your computer. The software or packages used in this tutorial may be updated from time to time so, we have also given you the version which was used when writing the tutorial.

---

Package	Link for download/installation instructions	Version tested
HISAT2	<a href="https://ccb.jhu.edu/software/hisat2/index.shtml">https://ccb.jhu.edu/software/hisat2/index.shtml</a>	2.0.4
samtools	<a href="https://github.com/samtools/samtools">https://github.com/samtools/samtools</a>	1.9
IGV	<a href="https://software.broadinstitute.org/software/igv/">https://software.broadinstitute.org/software/igv/</a>	2.3.81
kallisto	<a href="https://pachterlab.github.io/kallisto/download">https://pachterlab.github.io/kallisto/download</a>	0.43.0
R	<a href="https://www.r-project.org/">https://www.r-project.org/</a>	3.2.2
sleuth	<a href="https://pachterlab.github.io/sleuth/download">https://pachterlab.github.io/sleuth/download</a>	0.30.0
bedtools	<a href="http://bedtools.readthedocs.io/en/latest/content/installation.html">http://bedtools.readthedocs.io/en/latest/content/installation.html</a>	2.25.0

---

## 1.7 Let's get started!

To get started with the tutorial, head to the first section: [introducing the tutorial dataset](#).

The answers to all questions in the tutorial can be found in [answers.ipynb](#).

## 2 Introducing the tutorial dataset

Working through this tutorial, you will investigate the effect of vector transmission on gene expression of the malaria parasite. The dataset you will be using for this tutorial and *Figure 1* have been taken from the following publication:

### Vector transmission regulates immune control of *Plasmodium* virulence

Philip J. Spence, William Jarra, Prisca Lévy, Adam J. Reid, Lia Chappell, Thibaut Brugat, Mandy Sanders, Matthew Berriman and Jean Langhorne  
*Nature*. 2013 Jun 13; 498(7453): 228–231 doi:10.1038/nature12231

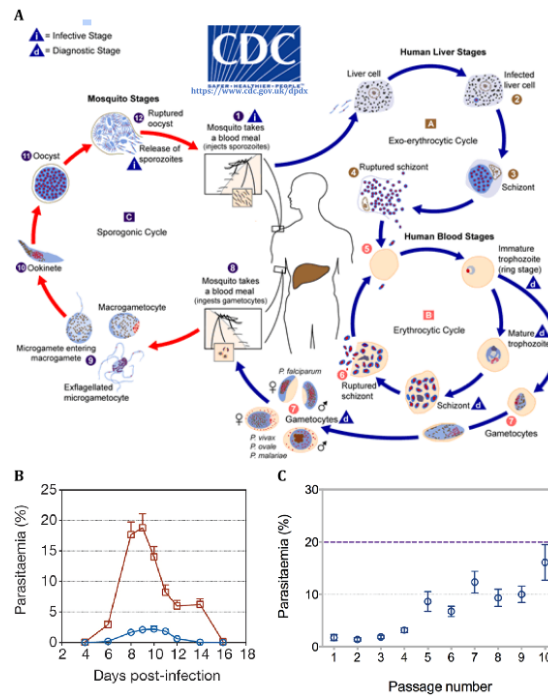


Figure 1. Serial blood passage increases virulence of malaria parasites.

### 2.1 Is the transcriptome of a mosquito-transmitted parasite different from one which has not passed through a mosquito?

The key reason for asking this question is that parasites which are transmitted by mosquito (MT) are less virulent (severe/harmful) than those which are serially blood passed (SBP) in the laboratory. *Figure 1A* shows the malaria life cycle, the red part highlighting the mosquito stage. *Figure 1B* shows the difference in virulence, measured by blood parasitemia (presence of parasites in the blood), between mosquito-transmitted and serially blood passed parasites.

*Figure 1C* shows that increasing numbers of blood passage post mosquito transmission results in increasing virulence, back to around 20% parasitemia. Subsequent mosquito transmission of high virulence parasites render them low virulence again.

We hypothesise that parasites which have been through the mosquito are somehow better able to control the mosquito immune system than those which have not. This control of the immune

system would result in lower parasitemia because this is advantageous for the parasite. Too high a parasitemia is bad for the mouse and therefore bad for the parasite.

## 2.2 Exercise 1

In this tutorial, you will be analysing **five RNA samples**, each of which has been sequenced on an Illumina HiSeq sequencing machine. There are **two conditions**: serially blood-passaged parasites (**SBP**) and mosquito transmitted parasites (**MT**). One with **three biological replicates** (SBP), one with **two biological replicates** (MT).

Sample name	Experimental condition	Replicate number
MT1	mosquito transmitted parasites	1
MT2	mosquito transmitted parasites	2
SBP1	serially blood-passaged parasites	1
SBP2	serially blood-passaged parasites	2
SBP3	serially blood-passaged parasites	3

The tutorial files can be found in the data directory. Let's go there now!

**Move into the directory containing the tutorial data files.**



```
cd data
```

**Check to see if the tutorial FASTQ files are there.**



```
ls *.fastq
```

**If the previous `ls` command didn't return anything, download and uncompress the tutorial FASTQ files.**



```
wget ftp://ftp.sanger.ac.uk/pub/project/pathogens/workshops/RNASeq_fq.tar.gz
tar -xf RNASeq_fq.tar.gz
mv RNASeq_tutorial_fastqs/* .
gunzip *.fastq.gz
```

The FASTQ files contain the raw sequence reads for each sample. There will typically be four lines per read:

1. Header
2. Sequence
3. Separator (usually a '+')
4. Encoded quality value

Take a look at one of the FASTQ files.



```
head MT1_1.fastq
```

You can find out more about the FASTQ format at [https://en.wikipedia.org/wiki/FASTQ\\_format](https://en.wikipedia.org/wiki/FASTQ_format).

---

## 2.3 Questions

### 2.3.1 Q1: Why is there more than one FASTQ file per sample?

*Hint: think about why there is a MT1\_1.fastq and a MT1\_2.fastq*

### 2.3.2 Q2: How many reads were generated for the MT1 sample?

*Hint: we want the total number of reads from both files (MT1\_1.fastq and MT1\_2.fastq) so perhaps think about the FASTQ format and the number of lines for each read or whether there's anything you can use in the FASTQ header to search and count...*

---

## 2.4 What's next?

For a quick recap of what the tutorial covers and the software you will need, head back to the [Introduction](#).

Otherwise, let's get started with [mapping RNA-Seq reads to the genome using HISAT2](#).

## 3 Mapping RNA-Seq reads to the genome using HISAT2

### 3.1 Introduction

For this exercise, we have reduced the number of reads in each sample to around 2.5 million to reduce the mapping time. However, this will be sufficient to detect most differentially expressed genes.

The objectives of this part of the tutorial are:

- use HISAT2 to build an index from the reference genome
- use HISAT2 to map RNA-Seq reads to the reference genome

#### 3.1.1 Mapping RNA-Seq reads to a genome

By this stage, you should have already performed a standard NGS quality control check on your reads to see whether there were any issues with the sample preparation or sequencing. For more information, see our [NGS Data formats and QC tutorial](#).

Next, we map our RNA-Seq reads to a reference genome to get context. This allows you to visually inspect your RNA-Seq data, identify contamination, novel exons and splice sites as well as giving you an overall feel for your transcriptome.

**HISAT2** To map the RNA-Seq reads from our five samples to the reference genome, we will be using **HISAT2**, a fast and sensitive splice-aware aligner. HISAT2 compresses the genome using an indexing scheme based on the [Burrows-Wheeler transform \(BWT\)](#) and [Ferragina-Manzini \(FM\) index](#) to reduce the amount of space needed to store the genome. This also makes the genome quick to search, using a whole-genome FM index to anchor each alignment and then tens of thousands local FM indexes for very rapid extensions of these alignments.

For more information, and to find the original version of *Figure 2*, please see the HISAT paper:

**HISAT: a fast spliced aligner with low memory requirements**

Daehwan Kim, Ben Langmead and Steven L Salzberg

*Nat Methods*. 2015 Apr;12(4):357-60. doi:10.1038/nmeth.3317

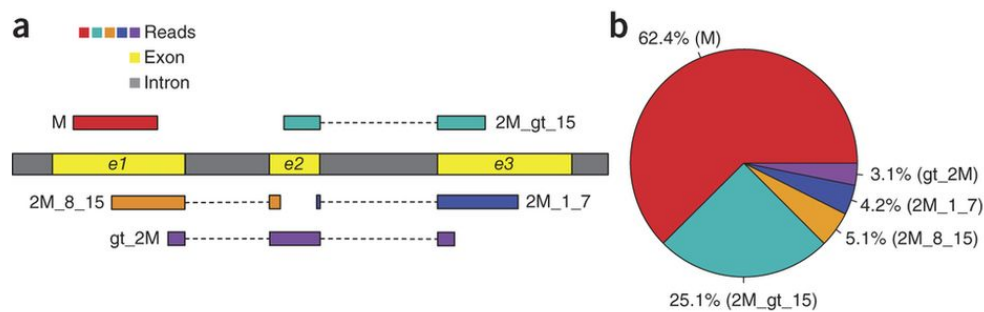
HISAT2 is a splice-aware aligner which means it takes into account that when a read is mapped it may be split across multiple exons with (sometimes large) intronic gaps between aligned regions. As you can see in *Figure 2*, HISAT2 splits read alignments into five classes based on the number of exons the read alignment is split across and the length of the anchor (longest continuously mapped portion of a split read):

- Aligns to a single exon (*M*)
- Alignment split across 2 exons with long anchors over 15bp (*2M\_gt\_15*)
- Alignment split across 2 exons with intermediate anchors between 8bp and 15bp (*2M\_8\_15*)
- Alignment split across 2 exons with short anchors less than 7bp (*2M\_1\_7*)
- Alignment split across more than 2 exons (*gt\_2M*)

HISAT2 used the global index to place the longest continuously mapped portion of a read (*anchor*). This information is then used to identify the relevant local index. In most cases, HISAT2 will only

need to use a single local index to place the remaining portion of the read without having to search the rest of the genome.

For the human genome, HISAT2 will build a single global index and 48,000 local FM indexes. Each of the local indexes represents a 64kb genomic region. The majority of human introns are significantly shorter than 64kb, so >90% of human introns fall into a single local index. Moreover, each of the local indexes overlaps its neighbour by ~1kb which means that it also has the ability to detect reads spanning multiple indexes.



**Figure 2. Read types and their relative proportions from 20 million simulated 100-bp reads**

There are five HISAT2 RNA-seq read mapping categories: (i) M, exonic read; (ii) 2M\_gt\_15, junction reads with long, >15-bp anchors in both exons; (iii) 2M\_8\_15, junction reads with intermediate, 8- to 15-bp anchors; (iv) 2M\_1\_7, junction reads with short, 1- to 7-bp, anchors; and (v) gt\_2M, junction reads spanning more than two exons (Figure 2A). Exonic reads span only a single exon and represent over 60% of the read mappings in the 20 million 100-bp simulated read dataset.

## 3.2 Exercise 2

Be patient, each of the following steps will take a couple of minutes!

**Make sure you are in the data directory with the tutorial files.**



```
cd data
```

**Look at the usage instructions for hisat2-build.**



```
hisat2-build -h
```

This not only tells us the version of HISAT2 we're using (essential for publication methods):

```
HISAT2 version 2.0.4 by Daehwan Kim
```

But, that we also need to give hisat2-build two pieces of information:

```
Usage: hisat2-build [options]* <reference_in> <ht2_index_base>
```

These are:

- `<reference_in>`  
location of our reference sequence file (PccAS\_v3\_genome.fa)
- `<ht2_index_base>`  
what we want to call our HISAT2 index files (PccAS\_v3\_hisat2.idx)

**Build a HISAT2 index for our *Plasmodium chabaudi chabaudi* AS (*P. chabaudi*) reference genome using `hisat2-build`.**



```
hisat2-build PccAS_v3_genome.fa PccAS_v3_hisat2.idx
```

You can see the generated index files using:



```
ls PccAS_v3_hisat2.idx*
```

**Look at the usage for `hisat2`.**



```
hisat2 -h
```

Here we can see that `hisat2` needs several bits of information so that it can do the mapping:

```
hisat2 [options]* -x <ht2-idx> {-1 <m1> -2 <m2> | -U <r>} [-S <sam>]
```

- `-x <ht2-idx>`  
the prefix that we chose for our index files with `hisat2-build` (PccAS\_v3\_hisat2.idx)
- `{-1 <m1> -2 <m2> | -U <r>}`  
the left (-1) and right (-2) read files for the sample (MT1\_1.fastq and MT1\_2.fastq respectively)
- `[-S <sam>]`  
the name of the file we want to write the output alignment to (MT1.sam) as, by default, `hisat2` will print the results to the terminal (stdout)

We will also be adding one more piece of information, the maximum intron length (default 500,000 bases). For this analysis, we want to set the maximum intron length to 10,000. We can do this by adding the option `--max-intronlen 10000`.

**Map the reads for the MT1 sample using HISAT2.**



```
hisat2 --max-intronlen 10000 -x PccAS_v3_hisat2.idx \  
-1 MT1_1.fastq -2 MT1_2.fastq -S MT1.sam
```

HISAT2 has written the alignment in SAM format. This is a format which allows humans to look at our alignments. However, we need to convert the SAM file to its binary version, a BAM file. We do this for several reasons. Mainly we do it because most downstream programs require our alignments to be in BAM format and not SAM format. However, we also do it because the BAM file is smaller and so takes up less (very precious!) storage space. For more information, see the format guide: <http://samtools.github.io/hts-specs/SAMv1.pdf>.

### Convert the SAM file to a BAM file.



```
samtools view -b -o MT1.bam MT1.sam
```

We now need to sort the BAM file ready for indexing. When we aligned our reads with HISAT2, the alignments were produced in the same order as the sequences in our FASTQ files. To index the BAM file we need the alignments to be ordered by their respective positions in the reference genome. We can do this using `samtools sort` which will sort the alignments by their co-ordinates for each chromosome.

### Sort the BAM file.



```
samtools sort -o MT1_sorted.bam MT1.bam
```

Next, we need to index our BAM file. This makes searching the alignments much more efficient. It allows programs like IGV (which we will be using to visualise the alignment) to quickly get the alignments that overlap the genomic regions you're looking at. We can do this with `samtools index` which will generate an index file with the extension `.bai`.

### Index the BAM file so that it can be read efficiently by IGV.



```
samtools index MT1_sorted.bam
```

Now repeat this process of mapping, converting (SAM to BAM), sorting and indexing with the reads from the MT2 sample.

Hopefully, the sorted and indexed BAM files have already been generated for you. Let's check.



```
ls SBP*bam*
```

If this doesn't return `.bam` and `.bai` files for your three SBP samples, run these commands.



```
chmod +x map_SBP_samples.sh  
./map_SBP_samples.sh
```

These commands run a bash script which will do the mapping, converting, sorting and indexing for all of the SBP samples. There's a great introduction to bash scripting and loops as part of our [Unix tutorial](#).

If you have time at the end of the tutorial, feel free to take a look at the script and a breakdown of what it does in [Running commands on multiple samples](#). Bash scripts and loops are a useful way of automating an analysis and running the same commands for multiple samples. Imagine if you had 50 samples and not 5!

---

## 3.3 Questions

### 3.3.1 Q1: How many index files were generated when you ran `hisat2-build`?

*Hint: look for the files with the `.ht2` extension*



### 3.3.2 Q2: What was the *overall alignment rate* for each of the MT samples (MT1 and MT2) to the reference genome?

*Hint: look at the the output from the `hisat2` commands*

### 3.3.3 Q3: How many MT1 and MT2 reads were not aligned to the reference genome?

*Hint: look at the the output from the `hisat2` commands, you're looking for reads (not read pairs) which have aligned 0 times (remember that one read from a pair may map even if the other doesn't)*

---

## 3.4 What's next?

For a quick recap of what the tutorial covers head back to the [introduction](#).

If you want a reintroduction to the tutorial dataset, head back to [introducing the tutorial dataset](#).

Otherwise, let's continue on to [visualising transcriptomes with IGV](#).

## 4 Visualising transcriptomes with IGV

### 4.1 Introduction

**Integrative Genome Viewer (IGV)** allows us to visualise genomic datasets. We have a quick start guide [here](#) which contains the information you need to complete Section 4.2. The IGV user guide contains more information on all of the IGV features and functions: <http://software.broadinstitute.org/software/igv/UserGuide>.

The objectives of this part of the tutorial are:

- load a reference genome into IGV and navigate the genome
  - load an annotation file into IGV and explore gene structure
  - load read alignments into IGV and inspect read alignments
- 

### 4.2 Exercise 3

First, we will use `samtools` to create an index for the *P. chabaudi* reference genome, which IGV will use to traverse the genome. This index file will have the extension `.fai` and should always be in the same directory as the reference genome.

**Make sure you are in the data directory with the tutorial files.**



```
cd data
```

**Index the genome fasta file (required by IGV).**



```
samtools faidx PccAS_v3_genome.fa
```

**Start IGV.**



```
igv.sh
```

This will open the IGV main window. Now, we need to tell IGV which genome we want to use. IGV has many pre-loaded genomes available, but *P. chabaudi* is not one of them. This means we will need to load our genome from a file.

**Load your reference genome into IGV. Go to "Genomes -> Load Genome from File...". Select "PccAS\_v3\_genome.fa" and click "Open". For more information, see [Loading a reference genome](#) in our quick start guide.**

We not only want to see where our reads have mapped, but what genes they have mapped to. For this, we have an annotation file in [GFF3 format](#). This contains a list of features, their co-ordinates and orientations which correspond to our reference genome.

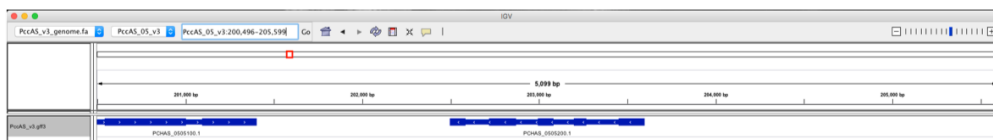
```
##gff-version 3
##sequence-region PccAS_01_v3 1745 580742
##sequence-region PccAS_02_v3 825 578413
...
PccAS_01_v3  DEFAULT  gene    1745   3349   .      +      .      ID=PCHAS_0100100
PccAS_01_v3  DEFAULT  mRNA   1745   3349   .      +      .      ID=PCHAS_0100100.1;Parent=PCHAS_0100100
PccAS_01_v3  DEFAULT  CDS    1745   1845   .      +      0      ID=PCHAS_0100100.1:exon:1;...
PccAS_01_v3  DEFAULT  CDS    2047   2317   .      +      1      ID=PCHAS_0100100.1:exon:2;...
...
```

### Example from PccAS\_v3 GFF3

Load your annotation file into IGV. Go to "File -> Load from File...". Select "PccAS\_v3.gff3" and click "Open". For more information, see [Loading gene annotations](#) in our quick start guide.

This will load a new track called "PccAS\_v3.gff3". The track is currently shown as a density plot. You will need to zoom in to see individual genes.

Search for the gene PCHAS\_0505200 by typing "PCHAS\_0505200" in the search box to zoom in and centre the view on PCHAS\_0505200.



IGV - PCHAS\_0505200

To get a clearer view of the gene structure, right click on the annotation track and click "Expanded".

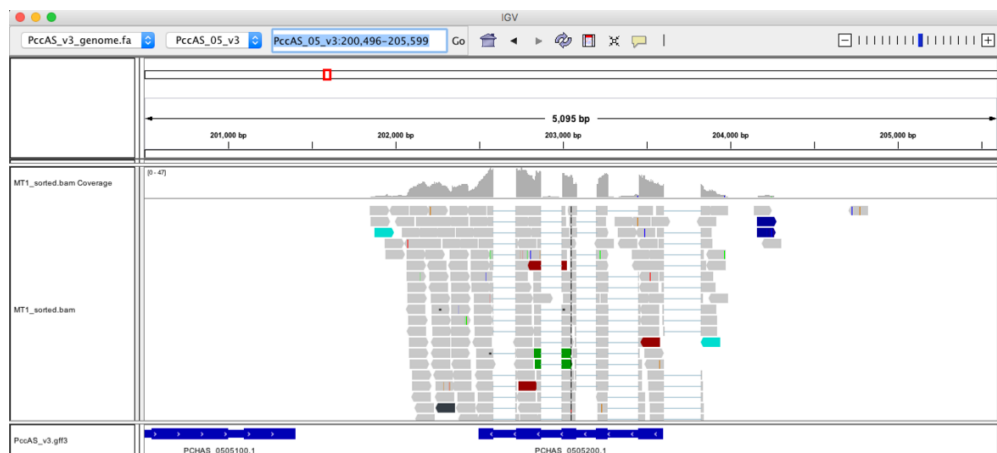


IGV - PCHAS\_0505200 expanded

In the annotation track, genes are presented as blue boxes and lines. These boxes represent exons, while the lines represent intronic regions. Arrows indicate the direction (or strand) of transcription for each of the genes. Now we have our genome and its annotated features, we just need the read alignments for our five samples.

Load your alignment file for the MT1 sample into IGV. Go to "File -> Load from File...". Select "MT1\_sorted.bam" and click "Open". For more information, see [Loading alignment files](#) in our quick start guide.

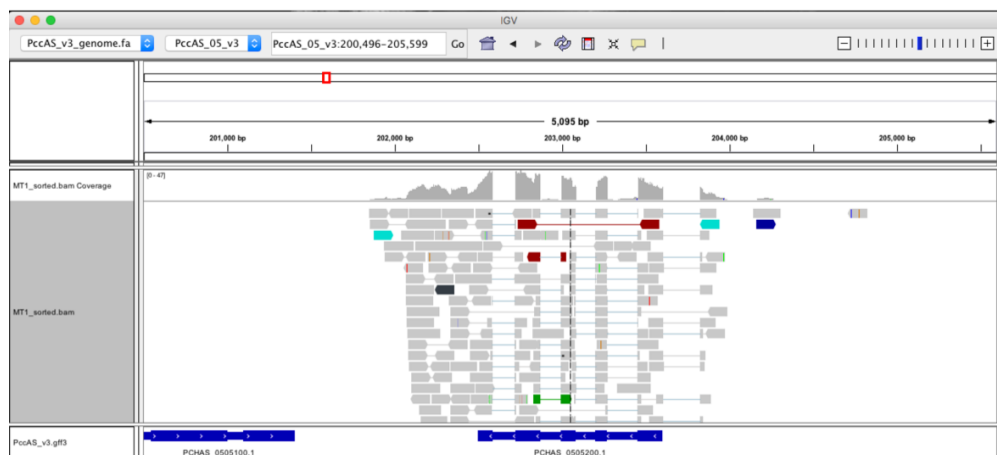
*Note: BAM files and their corresponding index files must be in the same directory for IGV to load them properly.*



IGV - MT1 read alignment

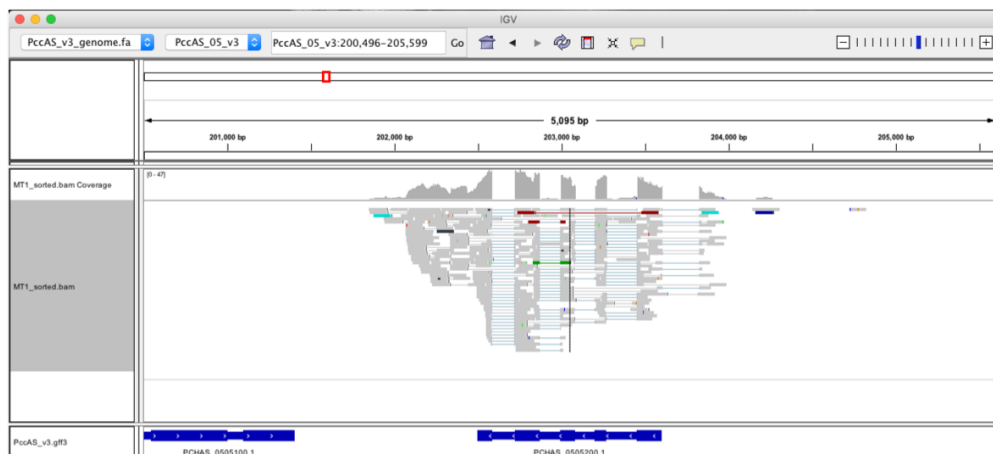
This will load a new track called "MT1\_sorted.bam" which contains the read alignments for the MT1 sample. We can change how we visualise our data by altering the view options. By default, IGV will display reads individually so they are compactly arranged. If you were to hover over a read in the default view, you will only get the details for that read. However, if we change our view so that the reads are visualised as pairs, the read pairs will be joined together by line and when we hover over either of the reads, we will get information about both of the reads in that pair.

To view our reads as pairs, right click on the MT1\_sorted.bam alignment track and click "View as pairs".



IGV - paired view

To condense the alignment, right click on the MT1\_sorted.bam alignment track and click "Squished".

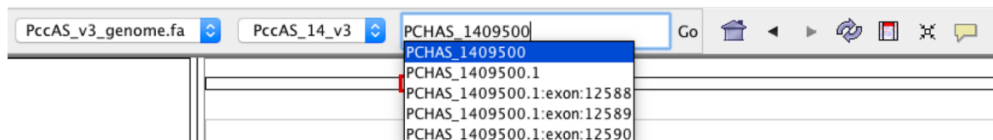


IGV - squished view

For more information on sorting, grouping and visualising read alignments, see the [IGV user guide](#).

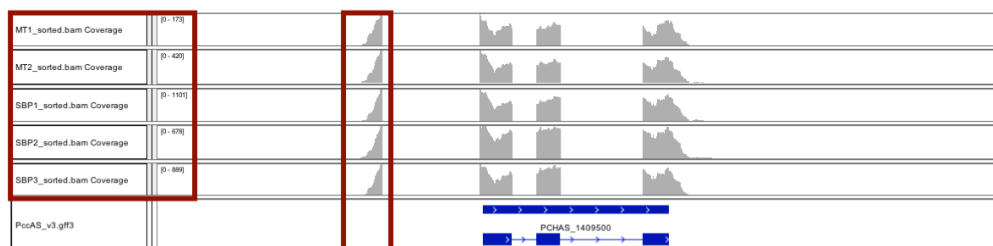
Load the remaining sorted BAM files for the MT2 sample and the three SBP samples.

Using the search box in the toolbar, go to PCHAS\_1409500. For more information, see [Jump to gene or locus](#) in our quick start guide.



IGV - search PCHAS\_1409500

The first thing to look at is the coverage range for this viewing window on the left-hand side. The three SBP samples have 2-3 times more reads mapping to this gene than the two MT samples. While at first glance it may seem like this gene may be differentially expressed between the two conditions, remember that some samples may have been sequenced to a greater depth than others. So, if a sample has been sequenced to a greater depth we would expect more reads to map in general.



IGV - coverage PCHAS\_1409500

From the gene annotation at the bottom we can also see that there are three annotated exon/CDS features for this gene. However, the coverage plot suggests there may be a fourth unannotated exon which, given the direction of the gene, could suggest a 5' untranslated region (UTR). Note the clean drop off of the coverage at around position 377,070.

---

## 4.3 Questions

### 4.3.1 Q1: How many CDS features are there in "PCHAS\_1402500"?

*Hint: Look at Jump to gene or locus in our quick start guide.*

### 4.3.2 Q2: Does the RNA-seq mapping agree with the gene model in blue?

*Hint: Look at the coverage track and split read alignments.*

### 4.3.3 Q3: Do you think this gene is differentially expressed and is looking at the coverage plots alone a reliable way to assess differential expression?

*Hint: Look at the coverage similarities/differences between the MT and SBP samples.*

---

## 4.4 What's next?

You can head back to [mapping RNA-Seq reads to the genome using HISAT2](#) or continue on to [transcript quantification with Kallisto](#).

## 5 Transcript quantification with Kallisto

### 5.1 Introduction

After visually inspecting the genome alignment, the next step in a typical RNA-Seq analysis is to estimate transcript abundance. To do this, reads are assigned to the transcripts they came from. These assignments are then used to quantify gene or transcript abundance (expression level).

For this tutorial, we are using [Kallisto](#) to assign reads to a set of transcript sequences and quantify transcript abundance. Kallisto does not assemble transcripts and cannot identify novel isoforms. So, when a reference transcriptome isn't available, the transcripts will need to be assembled *de novo* from the reads. However, for this tutorial, we already have a reference transcriptome available.

The objectives of this part of the tutorial are:

- use Kallisto to index a transcriptome
- use Kallisto to estimate transcript abundance

#### 5.1.1 Quantifying transcripts with Kallisto

Many of the existing methods used for estimating transcript abundance are **alignment-based**. This means they rely on mapping reads onto the reference genome. The gene expression levels are then calculated by counting the number of reads overlapping the transcripts. However, read alignment is a computationally and time intensive process. So, in this tutorial, we will be running [Kallisto](#) which uses a fast, **alignment-free** method for transcript quantification.

#### Near-optimal probabilistic RNA-seq quantification

Nicolas L Bray, Harold Pimentel, Páll Melsted and Lior Pachter

*Nat Biotechnol.* 2016 May;34(5):525-7. doi: [10.1038/nbt.3519](https://doi.org/10.1038/nbt.3519)

Kallisto uses a process called **pseudoalignment** to make it efficient. Rather than looking at where the reads map, Kallisto uses the *compatibility* between the reads and transcripts to estimate transcript abundance. Thus, most transcript quantification with Kallisto can be done on a simple laptop (Figure 3).

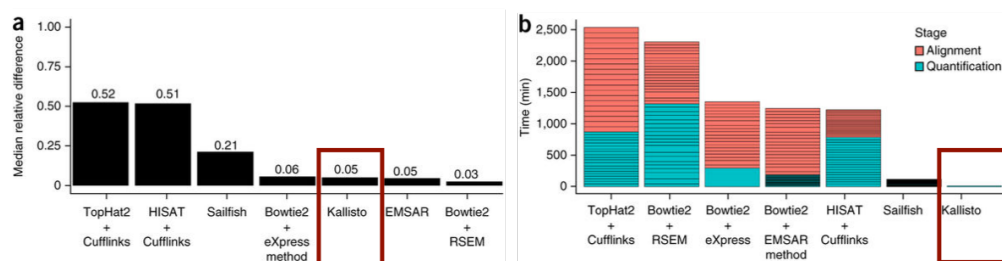


Figure 3. Performance of kallisto and other methods

#### Figure 3. Performance of kallisto and other methods

(a) Accuracy of kallisto, Cufflinks, Sailfish, EMSAR, eXpress and RSEM on 20 RSEM simulations of 30 million 75-bp paired-end reads. (b) Total running time in minutes for processing the 20 simulated data sets of 30 million paired-end reads described in a. Please see the [Kallisto publication](#) for original figure and more information.

**Step 1: building a Kallisto index** As with alignment-based methods, Kallisto needs an index. To generate the index, Kallisto first builds a transcriptome de Bruijn Graph (T-BDG) from all of the  $k$ -mers (short sequences of  $k$  nucleotides) that it finds in the transcriptome. Each node in the graph corresponds to a  $k$ -mer and each transcript is represented by its path through the graph. Using these paths, each  $k$ -mer is assigned a  $k$ -compatibility class. Some  $k$ -mers will be redundant i.e. shared by the same transcripts. These are skipped to make the index compact and quicker to search. A great worked example of this process can be found [here](#).

The command `kallisto index` can be used to build a Kallisto index from transcript sequences.



```
kallisto index
```

Here we can see the version of Kallisto that we're using (useful for publication methods) and the information that we'll need to give `kallisto index`. The only information we need to give `kallisto index` is the location of our transcript sequences (`PccAS_v3_transcripts.fa`). However, it's useful to have a meaningful filename for the resulting index. We can add this by using the option `-i` which expects a value, our index prefix (`PccAS_v3_kallisto`).

**Step 2: estimating transcript abundance** With this Kallisto index, you can use `kallisto quant` to estimate transcript abundances. You will need to run this command separately for each sample.



```
kallisto quant
```

We can see that `kallisto quant` needs us to tell it where our sample reads are. Although we don't have to, it's usually a good idea to keep the results of each quantification in a different directory. This is because the output filename are always the same (e.g. `abundances.tsv`). If we ran a second analysis, these could get overwritten. To use a different output directory, we can use the `-o` option. We will also be using the `-b` option for bootstrapping.

**Bootstrapping** Not all reads will be assigned unambiguously i.e. to a single transcript. This means that there will be "noise" in our abundance estimates where reads can be assigned to multiple transcripts. Because Kallisto is so quick, it has time to quantify the uncertainty in its abundance estimates using random resampling and replacement. This process is called **bootstrapping** and indicates how reliable the expression estimates are from the observed pseudoalignment. The bootstrap values can be used downstream to distinguish the technical variability from the biological variability in your experiment.

---

## 5.2 Exercise 4

Make sure you are in the data directory with the tutorial files.



```
cd data
```



Build an index called *PccAS\_v3\_kallisto* from transcript sequences in *PccAS\_v3\_transcripts.fa*.



```
kallisto index -i PccAS_v3_kallisto PccAS_v3_transcripts.fa
```

Quantify the transcript expression levels for the MT1 sample with 100 bootstrap samples and calling the output directory *MT1*.



```
kallisto quant -i PccAS_v3_kallisto -o MT1 -b 100 MT1_1.fastq MT1_2.fastq
```

You'll find your Kallisto results in a new output directory which we called **MT1**. Let's take a look.



```
ls MT1
```

Running `kallisto quant` generated three output files in our MT1 folder:

- **abundance.h5**  
HDF5 binary file containing run info, abundance estimates, bootstrap estimates, and transcript length information.
- **abundance.tsv**  
Plain text file containing abundance estimates (doesn't contain bootstrap estimates).
- **run\_info.json**  
JSON file containing information about the run.

*Note: when the number of bootstrap values (-b) is very high, Kallisto will generate a large amount of data. To help, it outputs bootstrap results in HDF5 format (abundance.h5). This file can be read directly by [sleuth](#).*

In the **MT1/abundance.tsv** file we have the abundance estimates for each gene for the MT1 sample. Let's take a quick look.



```
head MT1/abundance.tsv
```

In **MT1/abundance.tsv** there are five columns which give us information about the transcript abundances for our MT1 sample.

- **target\_id**  
Unique transcript identifier.
- **length**  
Number of bases found in exons.
- **eff\_length**  
*Effective length*. Uses fragment length distribution to determine the effective number of positions that can be sampled on each transcript.
- **est\_counts**  
Estimated counts\*. This may not always be an integer as reads which map to multiple transcripts are fractionally assigned to each of the corresponding transcripts.
- **tpm**  
*Transcripts per million*. Normalised value accounting for length and sequence depth bias.

In the last column we have our normalised abundance value for each gene. These are our transcripts per million or TPM. If you have time at the end of this tutorial, see our [normalisation guide](#) which covers common normalisation methods and has a bonus exercise.

To get the result for a specific gene, we can use `grep`.



```
grep PCHAS_0100100 MT1/abundance.tsv
```

If we wanted to get the TPM value for a particular gene, we can use `awk`.



```
awk -F"\t" '$1=="PCHAS_0100100" {print $5}' MT1/abundance.tsv
```

Use `kallisto quant` four more times, for the MT2 sample and the three SBP samples.

---

## 5.3 Questions

### 5.3.1 Q1: What *k*-mer length was used to build the Kallisto index?

*Hint: look at the terminal output from `kallisto index`*

### 5.3.2 Q2: How many transcript sequences are there in *PccAS\_v3\_transcripts.fa*?

*Hint: you can use `grep` or look at the terminal output from `kallisto quant` or in the `run_info.json` files*

### 5.3.3 Q3: What is the transcripts per million (TPM) value for PCHAS\_1402500 in each of the samples?

*Hint: use `grep` to look at the `abundance.tsv` files*

### 5.3.4 Q4: Do you think PCHAS\_1402500 is differentially expressed?

---

## 5.4 What's next?

You can head back to [visualising transcriptomes with IGV](#) or continue on to [identifying differentially expressed genes with sleuth](#).

## 6 Identifying differentially expressed genes with Sleuth

### 6.1 Introduction

In the previous sections we have quantified our transcript abundance and looked at why counts are normalised. In this section you will be using `sleuth` to do some simple quality checks and get a first look at the results.

The objectives of this part of the tutorial are:

- use `sleuth` to perform quality control checks
- use `sleuth` to identify differentially expressed (DE) transcripts
- use `sleuth` to investigate DE transcripts

#### 6.1.1 Differential expression analysis (DEA)

Differential expression analysis tries to identify genes whose expression levels differ between experimental conditions. We don't normally have enough replicates to do traditional tests of significance for RNA-Seq data. So, most methods look for outliers in the relationship between average abundance and fold change and assume most genes are not differentially expressed.

Rather than just using a fold change threshold to determine which genes are differentially expressed, DEAs use a variety of statistical tests for significance. These tests give us a **p-value** which is an estimate of how often your observations would occur by chance.

However, we perform these comparisons for each one of the thousands of genes/transcripts in our dataset. A p-value of 0.01 estimates a probability of 1% for seeing our observation just by chance. In an experiment like ours with 5,000 genes we would expect 5 genes to be significantly differentially expressed by chance (i.e. even if there were no difference between our conditions). Instead of using a p-value we can use a **q-value** which accounts for the multiple testing and adjusts the p-value accordingly.

#### 6.1.2 `sleuth`

`sleuth` is a companion tool for `Kallisto`. Unlike most other tools, `sleuth` can utilize the technical variation information generated by `Kallisto` so that you can look at both the technical and biological variation in your dataset.

For the DEA, `sleuth` essentially tests two models, one which assumes that the abundances are equal between the two conditions (reduced) and one that does not (full). To identify DE transcripts it identifies those with a significantly better fit to the "full" model. For more information on `sleuth` and how it works, see Lior Pachter's blog post [A sleuth for RNA-Seq](#).

`sleuth` is written in the `R` statistical programming language, as is almost all RNA-Seq analysis software. Helpfully, it produces a web page that allows interactive graphical analysis of the data. However, we strongly recommend learning `R` for anyone doing a significant amount of RNA-seq analysis. It is nowhere near as hard to get started with as full-blown programming languages such as Perl or Python!

---

## 6.2 Exercise 5

For this tutorial, we've provided a series of R commands as an R script that will get sleuth running. **Make sure you are in the data directory with the tutorial files.**



```
cd data
```

### 6.2.1 Running sleuth

The commands we need to run sleuth are in the file `sleuth.R`. There's a great overview of the commands and what they do by the developers of sleuth [here](#). Using R is not as hard as it seems, most of this script was copied from the manual!

**Open `sleuth.R` and have a quick look at the commands.**



```
cat sleuth.R
```

You may also want to have a look at `hiseq_info.txt` which is where we define which condition each sample is associated with.



```
cat hiseq_info.txt
```

**You can run scripts containing R commands using `Rscript` followed by the script name. Run `sleuth.R`.**



```
Rscript sleuth.R
```

You won't see any output from this script in the notebook, just a `*` next to the command input (`[*]`) to let you know it's running.

If you were to run the script directly on the command line, sleuth will return a link which you can follow (<http://127.0.0.1:42427>). This will take you to a web page where you can navigate and explore the sleuth results.

**\*\* Click the link below or type the URL your a web browser (e.g. chrome or firefox) to open the sleuth results.\*\***

<http://127.0.0.1:42427>

You should now see a page with the heading "sleuth live". If not, just give the script a little longer and then refresh the page.

## 6.2.2 Using sleuth to quality check (QC) transcript quantification

Quality control checks are absolutely vital at every step of the experimental process. We can use sleuth to perform simple quality checks (QC) on our dataset.

At the top of the page, sleuth provides several tabs which we can use to determine whether the data is of good quality and whether we should trust the results we get.

First, let's take a look at a summary of our dataset.

**In the web page that has been launched, click on "Summaries -> processed data".**

Notice that the number of reads mapping differs quite a bit between MT and SBP samples? This is why we QC our data. In the MT samples >95% of the reads mapped to the genome, but only 15-30% are assigned to the transcriptome compared to >75% for the SBP samples. This suggests that there may be some residual ribosomal RNA left over from the RNA preparation. It's not a problem as we have enough reads and replicates for our analysis.

processed data

Names of samples, number of mapped reads, number of bootstraps performed by kallisto, and sample to covariate mappings.

kallisto version(s): 0.43.0

Show 25 entries

sample	reads_mapped	reads_proc	frac_mapped	bootstraps_present	bootstraps_used	condition
MT1	168298	1250000	0.1346	100	100	MT
MT2	341149	1250000	0.2729	100	100	MT
SBP1	1017818	1250000	0.8143	100	100	SBP
SBP2	951731	1250000	0.7614	100	100	SBP
SBP3	966663	1250000	0.7733	100	100	SBP

Showing 1 to 5 of 5 entries

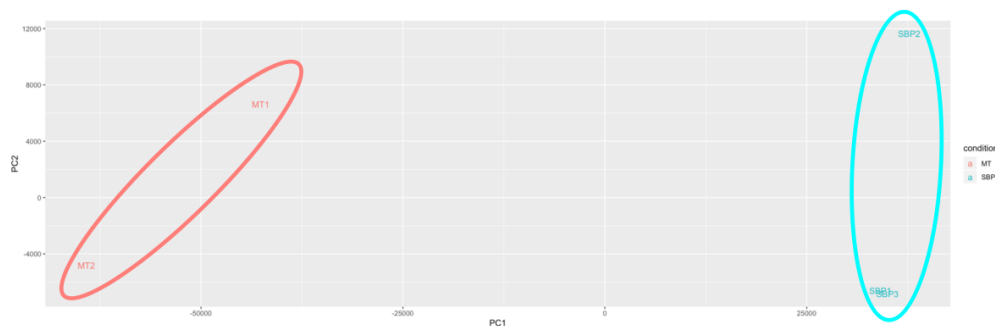
Previous 1 Next

Download Table

sleuth - processed data table

In some cases, we can identify samples which don't agree with other replicates (**outliers**) and samples which are related by experimental bias (**batch effects**). If we don't have many replicates, it's hard to detect outliers and batch effects meaning our power to detect DE genes is reduced.

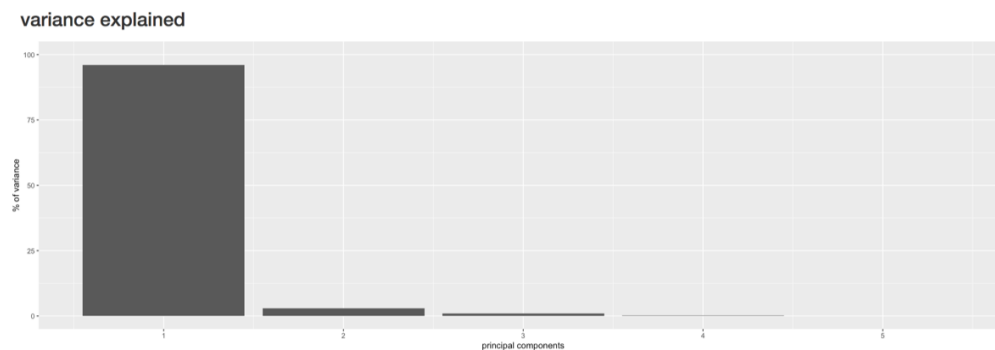
**Principal component analysis (PCA)** plots can be used to look at variation and strong patterns within the dataset. Batch effects and outliers often stand out quite clearly in the PCA plot and mean that you can account for them in any downstream analysis.



sleuth - PCA plot

Our samples form two condition-related clusters with the two MT samples (red) on the left and the three SBP samples on the right (blue). If we look at the variance bar plot, we can see that the

first principal component (PC1) accounts for >90% of the variation in our dataset. As the samples are clearly clustered on the x-axis (PC1) this suggests that most of the variation in the dataset is related to our experimental condition (Mt vs SBP).



sleuth - variance bar plot

### 6.2.3 Using sleuth to look at DE transcripts

We used the output from Kallisto to identify DE transcripts using sleuth. Let's take a look and see if we found any.

To see the results of the sleuth DEA, go to "*analyses -> test table*".

test table

Table of transcript names, gene names (if supplied), sleuth parameter estimates, tests, and summary statistics. What do the column names mean?

fit:  beta:  table type:

Show  entries Search:

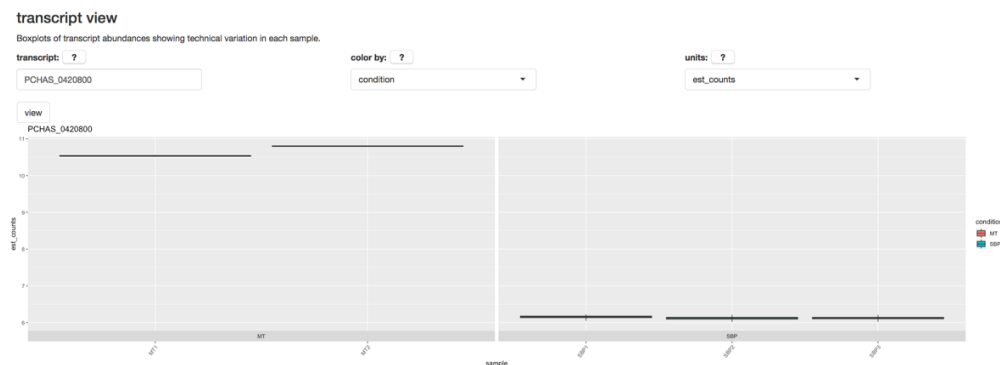
target_id	description	pval	qval	b	se_b	mean_obs	var_obs	tech_var	sigma_sq	smooth_sigma_sq	final_sigma_sq
PCHAS_0420800	hypothetical protein, pseudogene	0.000000e+00	0.000000e+00	-4.533267	0.1049194	7.947168	6.1741517	0.0007167526	0.011281626	0.01249295	0.01249295
PCHAS_1100300	CIR protein	4.181559e-84	8.724822e-81	4.173495	0.2147801	5.990678	5.2354177	0.0355216965	-0.022189493	0.01983491	0.01983491
PCHAS_0902400	histone H4, putative	1.055959e-74	1.468839e-71	-2.247036	0.1228781	6.659965	1.5283374	0.0015008966	0.016613673	0.01661793	0.01661793
PCHAS_0902500	histone H2B, putative	1.703942e-60	1.777638e-57	-2.065962	0.1259195	6.467845	1.2869171	0.0015795677	0.007029692	0.01744730	0.01744730
PCHAS_0702100	fam-a protein	1.125871e-36	0.396522e-34	-3.308070	0.2615178	4.014704	3.3445509	0.0242426628	0.057827230	0.03147513	0.05782723
PCHAS_0625000	lysophospholipase, putative	2.640683e-36	1.836595e-33	-3.286645	0.2612108	3.474759	3.2927774	0.0433405370	0.026214665	0.03853678	0.03853678
PCHAS_0400300	CIR protein	4.151719e-24	2.475017e-21	-4.591849	0.4533829	2.673322	6.4136774	0.1447773921	-0.027237379	0.10188982	0.10188982
PCHAS_0302100	CIR protein	2.765332e-22	1.442467e-19	-3.750792	0.3863285	2.832702	4.2285815	0.0960407750	-0.087309364	0.08105891	0.08105891
PCHAS_0401000	conserved Plasmodium chabaudi protein, unknown function	6.327522e-22	2.933861e-19	-1.685394	0.1751228	4.690290	0.8650022	0.0101870055	0.006927637	0.02661458	0.02661458
PCHAS_0109400	histone H3, putative	7.378663e-22	3.079116e-19	-2.268827	0.2361122	6.121534	1.5941743	0.0022506657	0.064648087	0.01912937	0.06464809

target\_id description pval qval b se\_b mean\_obs var\_obs tech\_var sigma\_sq smooth\_sigma\_sq final\_sigma\_sq

sleuth - transcript table

The important columns here are the **q-value** and the **beta value** (analogous to fold change). By default, the table is sorted by the q-value. We can see that our top transcript is PCHAS\_0420800, a hypothetical protein/pseudogene. Now let's take a closer look at that transcript.

Go to "*analyses -> transcript view*". Enter "PCHAS\_0420800" into the "*transcript*" search box. Click "*view*".



sleuth - transcript view

On the left you have the abundances for the MT replicates and on the right, the SBP replicates. We can see that this transcript is more highly expressed in the MT samples than in the SBP samples. This is also reflected by the fold change in the test table ( $b = -4.5$ ). The  $b$  value is negative as it represents the fold change in SBP samples relative to those in the MT samples.

Finally, let's take a look at the gene level.

To see the results of the sleuth DEA, go to "*analyses -> test\_table*". Under "*table type*" select "*gene table*". Click on the column header "*qval*" in the table to sort the rows by ascending  $q$ -value.

test table

Table of transcript names, gene names (if supplied), sleuth parameter estimates, tests, and summary statistics. [What do the column names mean?](#)

fit:     beta:     table type:     group by:

Show  entries    Search:

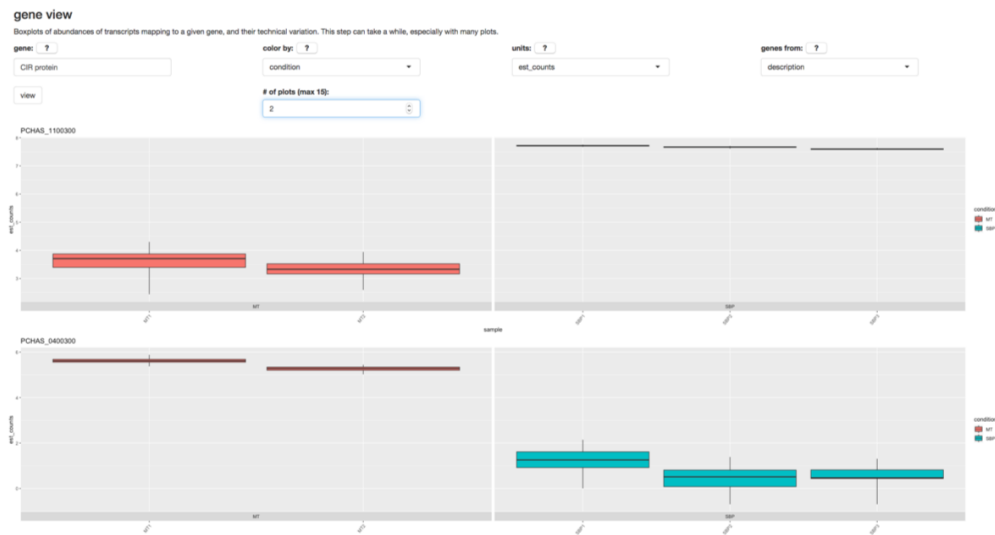
description	target_id	pval	qval	num_transcripts	all_target_ids
hypothetical protein, pseudogene	PCHAS_0420800	0.00000e+00	0.00000e+00	3	PCHAS_0420800
CIR protein	PCHAS_1100300	4.161559e-84	8.724822e-81	180	PCHAS_1100300
histone H4, putative	PCHAS_0902400	1.055959e-74	1.468839e-71	1	PCHAS_0902400
histone H2B, putative	PCHAS_0902500	1.703942e-60	1.777638e-57	1	PCHAS_0902500
fam-a protein	PCHAS_0702100	1.125871e-38	9.396522e-34	142	PCHAS_0702100
lysophospholipase, putative	PCHAS_0625000	2.640983e-36	1.836595e-33	28	PCHAS_0625000
conserved Plasmodium chabaudi protein, unknown function	PCHAS_0401000	6.327522e-22	2.933861e-19	8	PCHAS_0401000
histone H3, putative	PCHAS_0109400	7.378663e-22	3.079116e-19	2	PCHAS_0109400
histone H2A, putative	PCHAS_1116500	2.349455e-21	8.909184e-19	1	PCHAS_1116500
haloacid dehalogenase-like hydrolase, putative	PCHAS_1041800	2.369986e-18	5.817619e-16	10	PCHAS_1041800

description    target\_id    pval    qval    num\_transcripts    all\_target\_ids

sleuth - gene table

The transcripts have now been grouped by their descriptions. Let's take a closer look at the CIR proteins.

Go to "*analyses -> gene view*". In the "*gene*" search box enter "CIR protein" (without the quotes).



sleuth - gene view

Here we can see the individual CIR protein transcript abundances. We can see that PCHAS\_1100300 is more highly expressed in the SBP samples and PCHAS\_0302100 is more highly expressed in the MT samples.

## 6.3 Questions

6.3.1 Q1: Is our gene from earlier, PCHAS\_1402500, significantly differentially expressed?

## 6.4 What's next?

You can head back to [transcript quantification with Kallisto](#) or continue on to [interpreting the results](#).



## 7 Interpreting the results

### 7.1 Introduction

The main objective of this part of the tutorial is to use simple Unix commands to get a list of significantly differentially expressed genes. Using this gene list and the quantitative information from our analysis we can then start to make biological inferences about our dataset.

Using the R script (`sleuth.R`), we printed out a file of results describing the differentially expressed genes in our dataset. This file is called `kallisto.results`.

The file contains several columns, of which the most important are:

- **Column 1:** `target_id` (gene id)
- **Column 2:** description (some more useful description of the gene than its id)
- **Column 3:** `pval` (p value)
- **Column 4:** `qval` (p value corrected for multiple hypothesis testing)
- **Column 5:** `b` (fold change)

With a little Linux magic we can get the list of differentially expressed genes with only the columns of interest as above.

---

### 7.2 Exercise 6

Make sure you are in the data directory with the tutorial files.



```
cd data
```

To get the genes which are most highly expressed in our SBP samples, we must first filter our results. There are two columns we want to filter our data on: `b` (column 5) and `qval` (column 4). These columns represent whether the gene is differentially expressed and whether that change is significant.

The following command will get those genes which have an adjusted p value (`qval`) less than 0.01 and a positive fold change. These genes are more highly expressed in the SBP samples.



```
awk -F "\t" '$4 < 0.01 && $5 > 0' kallisto.results | cut -f1,2,3,4,5 | head
```

We used `awk` to filter the gene list and print only the lines which met our search criteria (`qval > 0.01, b > 0`). The option `-F` tells `awk` what delimiter is used to separate the columns. In this case, it was a tab or its regular expression `"\t"`. We then use `cut` to only print out columns 1-5. You can also do that within the `awk` command. Finally, we use `head` to get the first 10 lines of the output.

---

Alternatively, we can look for the genes which are more highly expressed in the MT samples.



```
awk -F "\t" '$4 < 0.01 && $5 < 0' kallisto.results | cut -f1,2,3,4,5 | head
```

It can be useful to have a quick look and compare gene lists. For example, whether a certain gene product is seen more often in the genes most highly expressed in one condition or another. A quick and dirty method would be to use the gene descriptions (or gene products).

You could extract the gene products (column 2) for genes which are more highly expressed in the SBP samples using `sort` and then `uniq`.



```
awk -F "\t" '$4 < 0.01 && $5 < 0 {print $2}' kallisto.results | sort | uniq
```

We can count each time these unique gene products occur in the list using `uniq -c`.



```
awk -F "\t" '$4 < 0.01 && $5 < 0 {print $2}' kallisto.results | \
  sort | uniq -c
```

And, if we wanted to make it a bit easier to see commonly found gene products we can sort this again by the frequency count we got from the `uniq` command. The `sort` command will put these in ascending numerical (`-n`) order.



```
awk -F "\t" '$4 < 0.01 && $5 < 0 {print $2}' kallisto.results | \
  sort | uniq -c | sort -n
```

If you wanted to look for the frequency of a particular gene product you could also use `grep`.



```
awk -F "\t" '$4 < 0.01 && $5 < 0 {print $2}' kallisto.results | grep -c CIR
```

Or building on the earlier command:



```
awk -F "\t" '$4 < 0.01 && $5 < 0 {print $2}' kallisto.results | \
  sort | uniq -c | grep CIR
```

If you want to read more about this work related to this data it is published:

**Vector transmission regulates immune control of *Plasmodium* virulence**

Philip J. Spence, William Jarra, Prisca Lévy, Adam J. Reid, Lia Chappell, Thibaut Brugat, Mandy Sanders, Matthew Berriman and Jean Langhorne

*Nature*. 2013 Jun 13; 498(7453): 228–231 doi:[10.1038/nature12231](https://doi.org/10.1038/nature12231)

## 7.3 Questions

### 7.3.1 Q1: How many genes are more highly expressed in the SBP samples?

*Hint: try replacing `head` in the earlier command with another unix command to count the number of lines*

### 7.3.2 Q2: How many genes are more highly expressed in the MT samples?

*Hint: try replacing `head` in the earlier command with another unix command to count the number of lines*

### 7.3.3 Q3: Do you notice any particular genes that came up in the analysis?

*Hint: look for gene products that are seen more often in genes more highly expressed in the SBP samples than those more highly expressed in the MT samples*

---

## 7.4 What's next?

You can head back to **identifying differentially expressed genes with Sleuth** or continue on to **key aspects of differential expression analysis**.

## 8 Key aspects of differential expression analysis

### 8.1 Replicates and power

In order to accurately ascertain which genes are differentially expressed and by how much it is necessary to use replicated data. As with all biological experiments doing it once is simply not enough. There is no simple way to decide how many replicates to do, it is usually a compromise of statistical power and cost. By determining how much variability there is in the sample preparation and sequencing reactions, we can better assess how highly genes are really expressed and more accurately determine any differences. The key to this is performing biological rather than technical replicates. This means, for instance, growing up three batches of parasites, treating them all identically, extracting RNA from each and sequencing the three samples separately. Technical replicates, whereby the same sample is sequenced three times do not account for the variability that really exists in biological systems or the experimental error between batches of parasites and RNA extractions.

*Note: more replicates will help improve power for genes that are already detected at high levels, while deeper sequencing will improve power to detect differential expression for genes which are expressed at low levels.*

### 8.2 p-values vs. q-values

When asking whether a gene is differentially expressed we use statistical tests to assign a **p-value**. If a gene has a p-value of 0.05, we say that there is only a 5% chance that it is not really differentially expressed. However, if we are asking this question for every gene in the genome (~5500 genes for *Plasmodium*), then we would expect to see p-values less than 0.05 for many genes even though they are not really differentially expressed. Due to this statistical problem, we must correct the p-values so that we are not tricked into accepting a large number of erroneous results. **Q-values** are p-values which have been corrected for what is known as multiple hypothesis testing. Therefore, it is a q-value of less than 0.05 that we should be looking for when asking whether a gene is differentially expressed.

### 8.3 Alternative software

If you have a good quality genome and genome annotation such as for model organisms e.g. human, mouse, *Plasmodium*; map to the transcriptome to determine transcript abundance. This is even more relevant if you have variant transcripts per gene as you need a tool which will do its best to determine which transcript is really expressed. As well as [Kallisto](#) (Bray et al. 2016; PMID: 27043002), there is [eXpress](#) (Roberts & Pachter, 2012; PMID: 23160280) which will do this.

Alternatively, you can map to the genome and then call abundance of genes, essentially ignoring variant transcripts. This is more appropriate where you are less confident about the genome annotation and/or you don't have variant transcripts because your organism rarely makes them or they are simply not annotated. [Tophat2](#) (Kim et al., 2013; PMID: 23618408), [HISAT2](#) (Pertea et al. 2016; PMID: 27560171), [STAR](#) (Dobinet al., 2013; PMID: 23104886) and [GSNAP](#) (Wu & Nacu, 2010; PMID: 20147302) are all splice-aware RNA-seq read mappers appropriate for this task. You then need to use a tool which counts the reads overlapping each gene model. [HTSeq](#) (Anders et al.,

2015; PMID: [25260700](#)) is a popular tool for this purpose. [Cufflinks](#) (Trapnell et al. 2012; PMID: [22383036](#)) will count reads and determine differentially expressed genes.

There are a variety of programs for detecting differentially expressed genes from tables of RNA-seq read counts. [DESeq2](#) (Love et al., 2014; PMID: [25516281](#)), [EdgeR](#) (Robinson et al., 2010; PMID: [19910308](#)) and [BaySeq](#) (Hardcastle & Kelly, 2010; PMID: [20698981](#)) are good examples.

#### 8.4 What do I do with a gene list?

Differential expression analysis results are a list of genes which show differences between two conditions. It can be daunting trying to determine what the results mean. On one hand, you may find that there are no real differences in your experiment. Is this due to biological reality or noisy data? On the other hand, you may find several thousands of genes are differentially expressed. What can you say about that?

Other than looking for genes you expect to be different or unchanged, one of the first things to do is look at **Gene Ontology (GO) term enrichment**. There are many different algorithms for this, but you could annotate your genes with functional terms from GO using for instance [Blast2GO](#) (Conesa et al., 2005; PMID: [16081474](#)) and then use [TopGO](#) (Alexa et al., 2005; PMID: [16606683](#)) to determine whether any particular sorts of genes occur more than expected in your differentially expressed genes.

---

#### 8.5 Congratulations, you have reached the end of this tutorial!

We hope you've enjoyed our RNA-Seq tutorial. You can find the answers to all of the questions in this tutorial [here](#). To revisit the previous section, click [here](#). Or, to go back to the beginning click [here](#).

## 9 Normalisation

### 9.1 Introduction

In the previous section, we looked at estimating transcript abundance with Kallisto. The abundances are reported as **transcripts per million (TPM)**, but what does TPM mean and how is it calculated?

The objectives of this part of the tutorial are:

- *understand why RNA-Seq normalisation metrics are used*
- *understand the difference between RPKM, FPKM and TPM*
- *calculate RPKM and TPM for a gene of interest*

There are many useful websites, publications and blog posts which go into much more detail about RNA-Seq normalisation methods. Here are just a couple (in no particular order):

- [What the FPKM? A review of RNA-Seq expression units](#)
- [RPKM, FPKM and TPM, clearly explained](#)
- [A survey of best practices for RNA-seq data analysis](#)
- [The RNA-seq abundance zoo](#)

---

### 9.2 Why do we use normalisation units instead of raw counts?

Raw reads counts are the number of reads originating from each transcript which can be affected by several factors:

- **sequencing depth (total number of reads)**  
The more we sequence a sample, the more reads we expect to be assigned.
- **gene/transcript length**  
The longer the gene or transcript, the more reads we expect to be assigned to it.

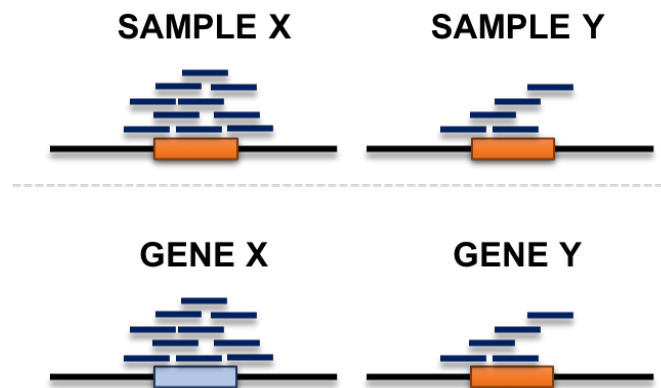


Figure 4. Effect of sequencing depth and gene length on raw read counts

Look at the top part of Figure 4. In which sample, X or Y, is the gene more highly expressed?

Neither, it's the same in both. What we didn't tell you was that the total number of reads generated for sample A was twice the number than for sample B. That meant almost twice the number of reads are assigned to the same gene in sample A than in sample B.

Look at the bottom part of Figure 4. Which gene, X or Y, has the greatest gene level expression?

Neither, they are both expressed at the same level. This time we didn't tell you that gene X is twice the length of gene Y. This meant that almost twice the number reads were assigned to gene X than gene Y.

In the top part of Figure 4, the gene in sample X has twice the number of reads assigned to it than the same gene in sample Y. What isn't shown is that sample X had twice the number or total reads than sample Y so we would expect more reads to be assigned in sample X. Thus, the gene is expressed at roughly the same level in both samples. In the bottom part of Figure 4, gene X has twice the number of reads assigned to it than gene Y. However, gene X is twice the length of gene Y and so we expect more reads to be assigned to gene X. Again, the expression level is roughly the same.

### 9.2.1 Reads per kilobase per million (RPKM)

Reads per kilobase (of exon) per million (reads mapped) or **RPKM** is a **within sample** normalisation method which takes into account sequencing depth and length biases.

To calculate RPKM, you first normalise by sequencing depth and then by gene/transcript length.

1. **Get your *per million* scaling factor**

Count up the total number of reads which have been assigned (mapped) in the sample. Divide this number by 1,000,000 (1 million) to get your *per million* scaling factor (N).

2. **Normalise for sequencing depth**

Divide the number of reads which have been assigned to the gene or transcript (C) by the *per million* scaling factor you calculated in step 1. This will give you your reads per million (RPM).

3. **Get your *per kilobase* scaling factor**

Divide the total length of the exons in your transcript or gene in base pairs by 1,000 (1 thousand) to get your *per kilobase* scaling factor (L).

4. **Normalise for length**

Divide your RPM value from step 2 by your *per kilobase* scaling factor (length of the gene/-transcript in kilobases) from step 3. This will give you your reads per kilobase per million or RPKM.

This can be simplified into the following equation:

$$RPKM = \frac{C}{LN}$$

Where:

- **C** is number of reads mapped to the transcript or gene
- **L** is the total exon length of the transcript or gene in kilobases
- **N** is the total number of reads mapped in millions

### 9.2.2 Fragments per kilobase per million (FPKM)

Fragments per kilobase per million or **FPKM** is essentially the same as RPKM except that:

- RPKM is designed for **single-end** RNA-Seq experiments
- FPKM is designed for **paired-end** RNA-Seq experiments

In a paired-end RNA-Seq experiment, two reads may be assigned to a single fragment (in any orientation). Also, in some cases, only one of those reads will be assigned to a fragment (singleton). The only difference between RPKM and FPKM is that FPKM takes into consideration that two reads may be assigned to the same fragment.

### 9.2.3 Transcripts per million (TPM)

Calculating the **transcripts per million** or **TPM** is a similar process to RPKM and FPKM. The main difference is that you will first normalise for length bias and then for sequencing depth bias. In a nut shell, we are swapping the order of normalisations.

1. **Get your *per kilobase* scaling factor**  
Divide the total length of the exons in your transcript in base pairs by 1,000 (1 thousand) to get your *per kilobase* scaling factor.
  2. **Normalise for length**  
Divide the number of reads which have been assigned to the transcript by the *per kilobase* scaling factor you calculated in step 1. This will give you your reads per kilobase (RPK).
  3. **Get the sum of all RPK values in your sample**  
Calculate the RPK value for all of the transcripts in your sample. Add all of these together to get your total RPK value.
  4. **Get your *per million* scaling factor**  
Divide your total RPK value from step 3 by 1,000,000 (1 million) to get your *per million* scaling factor.
  5. **Normalise for sequencing depth**  
Divide your RPK value calculated in step 2 by the *per million* scaling factor from step 4. You now have your transcripts per millions value or TPM.
-



### 9.3 Calculating RPKM and TPM values

To try and answer this, let's look at a worked example. Here, we have three genes (A-C) and three biological replicates (1-3).

Gene	Length	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	2,000 bases	10	12	30
<b>B</b>	4,000 bases	20	25	60
<b>C</b>	1,000 bases	5	8	15

There are two things to notice in our dataset:

- Gene B has twice number reads mapped than gene A, possibly as it's twice the length
- Replicate 3 has more reads mapped than any of the other replicates, regardless of which gene we look at

#### 9.3.1 Calculating RPKM

**Step 1: get your per million scaling factor** In the table below is the total number of reads which mapped for each of the replicates. To get our *per million* scaling factor, we divide each of these values by 1,000,000 (1 million).

Gene	Replicate 1	Replicate 2	Replicate 3
Total reads mapped	3,500,000	4,500,000	10,600,000
Per million reads	3.5	4.5	10.6

**Step 2: normalise for sequencing depth** We now divide our read counts by the *per million* scaling factor to get our reads per million (RPM).

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	10	12	30
<b>B</b>	2	25	60
<b>C</b>	5	8	15

After:

Gene	Replicate 1 RPM	Replicate 2 RPM	Replicate 3 RPM
<b>A</b>	2.857	2.667	2.830
<b>B</b>	5.714	5.556	5.660
<b>C</b>	1.429	1.778	1.415

**Step 3: get your *per kilobase* scaling factor** Here we have our gene length in base pairs. For our *per kilobase* scaling factor we need to get our gene length in kilobases by dividing it by 1,000.

Gene	Length (base pairs)	Length (kilobases)
<b>A</b>	2,000	2
<b>B</b>	4,000	4
<b>C</b>	1,000	1

**Step 4: normalise for length** Finally, we divide our RPM values from step 2 by our *per kilobase* scaling factor from step 3 to get our reads per kilobase per million (RPKM).

Before:

Gene	Replicate 1 RPM	Replicate 2 RPM	Replicate 3 RPM
<b>A</b>	2.857	2.667	2.830
<b>B</b>	5.714	5.556	5.660
<b>C</b>	1.429	1.778	1.415

After:

Gene	Replicate 1 RPKM	Replicate 2 RPKM	Replicate 3 RPKM
<b>A</b>	1.43	1.33	1.42
<b>B</b>	1.43	1.39	1.42
<b>C</b>	1.43	1.78	1.42

Notice that even though replicate 3 had more reads assigned than the other samples and a greater sequencing depth, its RPKM is quite similar. And, that although gene B had twice the number of reads assigned than gene A, its RPKM is the same. This is because we have normalised by both length and sequencing depth.

### 9.3.2 Calculating TPM

Now we're going to calculate the TPM values for the same example data. As a reminder, here are our three genes (A-C) and three biological replicates (1-3).

Gene	Length	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	2,000 bases	10	12	30
<b>B</b>	4,000 bases	20	25	60
<b>C</b>	1,000 bases	5	8	15

**Step 1: get your *per kilobase* scaling factor** Again, our gene lengths are in base pairs. For our *per kilobase* scaling factor we need to get our gene length in kilobases by dividing it by 1,000.

Gene	Length (base pairs)	Length (kilobases)
<b>A</b>	2,000	2
<b>B</b>	4,000	4
<b>C</b>	1,000	1

**Step 2: normalise for length** Now we divide the number of reads which have been assigned to each gene by the per kilobase scaling factor we just calculated. This will give us our reads per kilobase (RPK).

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	10	12	30
<b>B</b>	2	25	60
<b>C</b>	5	8	15

After:

Gene	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	5	6	15
<b>B</b>	5	6.25	15
<b>C</b>	5	8	15

**Step 3: get the sum of all RPK values in your sample** Next, we sum the RPK values for each of our replicates. This will give us our total RPK value for each replicate. To make this example scalable, we assume there are other genes so the total RPK is made up.

Gene	Replicate 1	Replicate 2	Replicate 3
A	5	6	15
B	5	6.25	15
C	5	8	15
...	...	...	...
<b>Total RPK</b>	<b>150,000</b>	<b>202,500</b>	<b>352,500</b>

**Step 4: get your *per million* scaling factor** Here, instead of dividing our total mapped reads by 1,000,000 (1 million) to get our *per million* scaling factor, we divide our total RPK values by 1,000,000 (1 million).

Gene	Replicate 1	Replicate 2	Replicate 3
Total RPK	150,000	202,500	352,500
Per million RPK	0.1500	0.2025	0.3525

**Step 5: normalise for sequencing depth** Finally, we divide our individual RPK values from step 2 by the *per million* scaling factor in step 4 to give us our TPM values.

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
A	5	6	15
B	5	6.25	15
C	5	8	15

After:

Gene	Replicate 1	Replicate 2	Replicate 3
A	33.33	29.63	31.21
B	33.33	30.86	31.91
C	33.33	39.51	36.88

## 9.4 Which normalisation unit should I use?

Well, there's a lot of debate around this, so let's look at our total normalised values for each replicate.

### 9.4.1 RPKM

Gene	Replicate 1 RPKM	Replicate 2 RPKM	Replicate 3 RPKM
<b>A</b>	1.43	1.33	1.42
<b>B</b>	1.43	1.39	1.42
<b>C</b>	1.43	1.78	1.42
<b>Total RPKM</b>	<b>4.29</b>	<b>4.50</b>	<b>4.25</b>

### 9.4.2 TPM

Gene	Replicate 1	Replicate 2	Replicate 3
<b>A</b>	33.33	29.63	31.21
<b>B</b>	33.33	30.86	31.91
<b>C</b>	33.33	39.51	36.88
<b>Total TPM</b>	<b>100</b>	<b>100</b>	<b>100</b>

Notice that that total TPM value for each of the replicates is the same. This is not true for RPKM and FPKM where the total values differ. With TPM, having the same total value for each replicate makes it easier to compare the proportion of reads mapping to each gene across replicates (although you shouldn't really compare across experiments). With RPKM and FPKM, the differing total values make it much harder to compare replicates.

## 9.5 Questions

Below is the information for each of the five samples. You will need this information to answer the questions. We have put all of commands used to get this information in the [answers](#).

Sample	Total mapped reads	Transcript length	Assigned reads	Total RPK
MT1	2,353,750	3,697	2,541	293,431
MT2	2,292,271	3,709	3,392	675,190
SBP1	2,329,235	3,699	14,605	1,719,970
SBP2	2,187,718	3,696	17,302	1,429,540
SBP3	2,163,979	3,699	14,646	1,561,310

*Note: values have been rounded up to integers to make calculations easier. Assigned reads are the `est_count` from Kallisto for PCHAS\_1402500. Transcript lengths are the `est_length` from Kallisto for PCHAS\_1402500.*

**Q1: Using the `abundance.tsv` files generated by Kallisto and the information above, calculate the RPKM for PCHAS\_1402500 in each of our five samples.**

Sample	Per million scaling factor	RPM	Per kilobase scaling factor	RPKM
MT1				
MT2				
SBP1				
SBP2				
SBP3				

**Q2: Using the `abundance.tsv` files generated by Kallisto and the information above, calculate the TPM for PCHAS\_1402500 in each of our five samples.**

*Hint: don't forget to get your per million scaling factor.*

Sample	Per kilobase scaling factor	Reads per kilobase (RPK)	TPM
MT1			
MT2			
SBP1			
SBP2			
SBP3			

**Q3: Do these match the TPM values from Kallisto?**

*Hint: look at the `abundance.tsv` files for each of your samples.*

**Q4: Do you think PCHAS\_1402500 is differentially expressed between the MT and SBP samples?**

---

## 9.6 What's next?

You can head back to [transcript quantification with Kallisto](#) or continue on to [Identifying differentially expressed genes with Sleuth](#).

## 10 Running commands on multiple samples

Now, fair warning, you're going to wish we'd told you this earlier on. However, then you wouldn't have had the fun of running and updating each of the previous commands, growling at typos and generally wishing that you'd gone for that cup of coffee before starting this tutorial.

Here we go....we can use a **loop** to run the same commands for multiple samples.

There's a great introduction to bash scripting and loops as part of our [Unix tutorial](#). But let's take a look at how we could have generated genome alignments for all of our samples using a single loop.

**First let's go to our data directory.**



```
cd data
```

Whenever you write a loop, it's always a good idea to build it up slowly to check that it's doing what you think.



```
for r in *.fastq
do
    echo $r
done
```

This loop looks for all (\*) files which end with ".fastq". The for loop then executes a sequence of commands for each file name that it finds. In the first iteration its "MT1\_1.fastq", then "MT1\_2.fastq" and so on... In each iteration, we assigned each filename that it found to a variable called "r".

```
for r in *.fastq
```

Then, to check we got what we expected, we printed what the variable "r" represented back to the terminal. Because we want to use the variable ("r") we created we need to use dollar (\$) symbol.

```
echo $r
```

Now, if we left things as they are, we would be running the commands twice for each sample. This is because we have two FASTQ files for each sample i.e. "\_1.fastq" and "\_2.fastq". Let's change our loop so that we only get the "\_1.fastq" files.



```
for r1 in *_1.fastq
do
    echo $r1
done
```

Great! Now, the only problem here is that we're going to want to use both the "\_1.fastq" and the "\_2.fastq" files in our mapping. We can get around this by removing the "\_1.fastq" from the filename to give us our sample name.

```
sample=${r1/_1.fastq/}
```



This will replace the "\_1.fastq" at the end of the filename we stored as "r1" with nothing.

We've added a little descriptive message so that when we run our loop we know which iteration it's on and what it's doing. Let's try adding our HISAT2 mapping command.

*Note: we assume that the HISAT2 index has already been generated as that's a command you'll only need to run once.*



```
for r1 in *_1.fastq
do
  sample=${r1/_1.fastq/}
  echo "Processing sample: "$sample

  echo "Mapping sample: "$sample
  hisat2 -max-intronlen 10000 -x PccAS_v3_hisat2.idx \
  -1 "$sample"_1.fastq" -2 "$sample"_2.fastq" -S "$sample".sam"
done
```

Notice that because we're using a filename which starts with our variable, but ends with a set phrase, we need to write the phrase in double quotes.

```
"$sample"_1.fastq"
```

Now let's add in our samtools commands.



```
for r1 in *_1.fastq
do
  sample=${r1/_1.fastq/}
  echo "Processing sample: "$sample

  echo "Mapping sample: "$sample
  hisat2 -max-intronlen 10000 -x PccAS_v3_hisat2.idx \
  -1 "$sample"_1.fastq" -2 "$sample"_2.fastq" -S "$sample".sam"

  echo "Converting SAM to BAM: "$sample
  samtools view -b -o "$sample".bam" "$sample".sam"

  echo "Sorting BAM: "$sample
  samtools sort -o "$sample"_sorted.bam" "$sample".bam"

  echo "Indexing BAM: "$sample
  samtools index "$sample"_sorted.bam"
done
```

Finally, we don't really want to keep intermediate SAM and unsorted BAM files if we don't have to. They just take up precious space. So, let's make our samtools command a one-liner, passing the stdout from one command to another.



```
for r1 in *_1.fastq
do
  sample=${r1/_1.fastq/}
  echo "Processing sample: "$sample
  hisat2 -max-intronlen 10000 -x PccAS_v3_hisat2.idx \
  -1 $sample"_1.fastq" -2 $sample"_2.fastq" \
  | samtools view -b - \
  | samtools sort -o $sample"_sorted.bam" - \
  && samtools index $sample"_sorted.bam"
done
```

You could also have used this approach for transcript quantification with Kallisto, assuming you had already generated the Kallisto index.

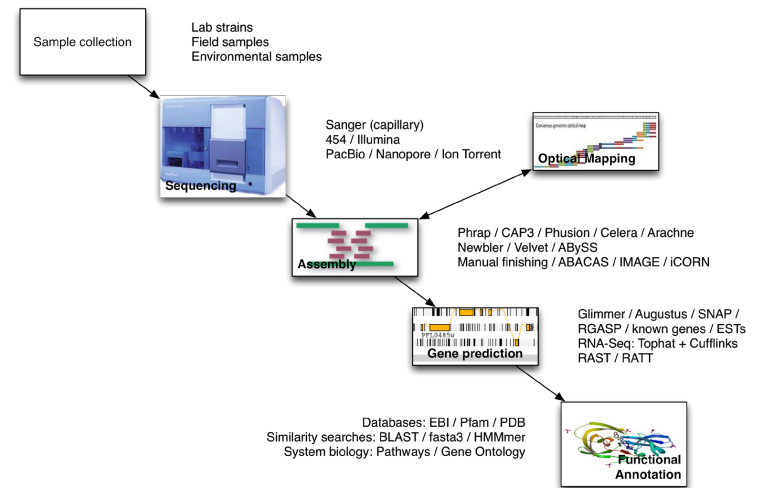


```
for r1 in *_1.fastq
do
  echo $r1
  sample=${r1/_1.fastq/}
  echo "Quantifying transcripts for sample: "$sample
  kallisto quant -i PccAS_v3_kallisto -o $sample -b 100 \
  $sample'_1.fastq' $sample'_2.fastq'
done
```

Wellcome Trust Advance Course:  
 Module 8: Genome assembly  
 2018-10-12

Shane McCarthy  
 Department of Genetics  
 sam68@cam.ac.uk

# Introduction to genome assembly



## What is genome assembly?

- Sequencing technologies can only sequence short stretches of DNA
- Given many (millions or billions) of reads, produce a linear (or perhaps circular) genome
- Clone based
  - Select clones using markers
  - Sequence clones separately
- Whole-genome shotgun
  - Fragment whole-genome and sequence
- De novo assembly

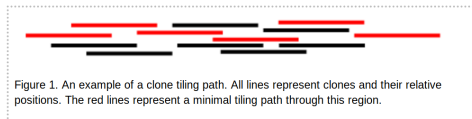
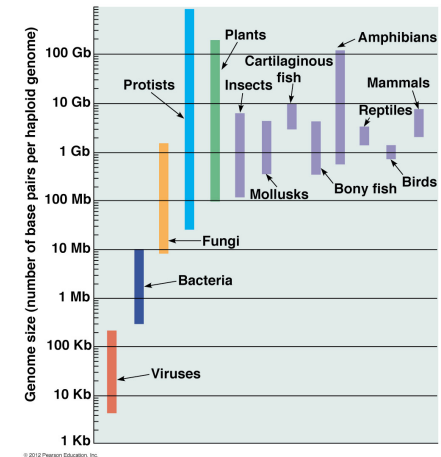


Figure 1. An example of a clone tiling path. All lines represent clones and their relative positions. The red lines represent a minimal tiling path through this region.

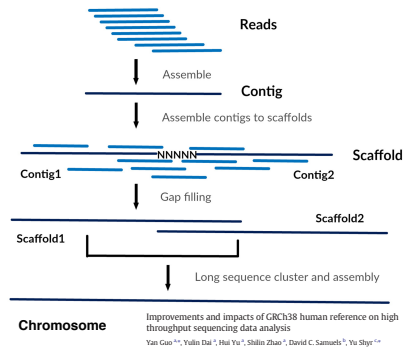


## Key considerations

- sequencing coverage
- errors in reads
- reads
- non-uniqueness of the genome (repeats)
- genome size
- sample heterozygosity
- running time and memory

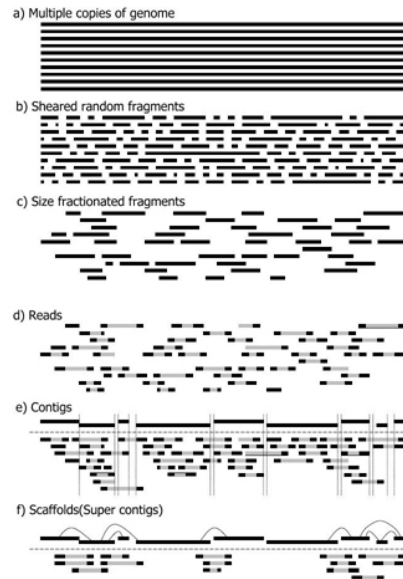


# Terminology



Improvements and impacts of GRCh38 human reference on high throughput sequencing data analysis  
 Yan Gao<sup>1\*</sup>, Yubin Dai<sup>1</sup>, Hai Yu<sup>1</sup>, Shilin Zhao<sup>1</sup>, David C. Samsuels<sup>2</sup>, Yu Shyr<sup>1,2\*</sup>

- **contig**: contiguous stretch of assembled sequence
- **scaffold**: ordered set of contigs with gaps (Ns) place between



# Long-range technologies

Table 1 | Long-range sequencing and mapping platforms

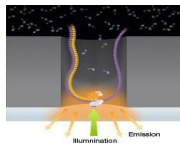
Platform	General characteristics and costs	Major applications	Bioinformatics challenges
PacBio SMRT sequencing	Single-molecule long reads averaging ~10 kb with some approaching 100 kb; several fold more expensive than short reads	De novo genome assembly, structural variant detection, gene isoform resolution and epigenetic modifications	Raw reads have high error rates dominated by false insertions; requires new alignment and error correction algorithms
Oxford Nanopore sequencing	Single-molecule long reads averaging ~10 kb with some >1 Mb; several fold more expensive than short reads	De novo genome assembly, structural variant detection, gene isoform resolution and epigenetic modifications	Raw reads have high error rates dominated by false deletions and homopolymer errors; requires new alignment and error correction algorithms
10X Genomics Chromium	Linked reads spanning ~100 kb derived from a collection of short-read sequences; moderately more expensive than short reads	De novo genome assembly and scaffolding, phasing, detection of large structural variants (>10 kb) and single-cell gene expression	Sparse sequencing rather than true long reads; more complicated to align, with poorer resolution of locally repetitive sequences
Hi-C-based analysis	Pairs of short reads (<100 bp) formed from crosslinking chromatin interactions; moderately more expensive than short reads	Genome scaffolding and phasing	Sparse sequencing with highly variable genomic distance between pairs (1 kb to 1 Mb or longer)
BioNano Genomics optical mapping	Optical mapping of long DNA molecules (~250 kb or longer) labelled with fluorescent probes; less expensive than short reads	Genome scaffolding and detection of large structural variants (>10 kb)	Limited algorithms to discover high-confidence alignment between an optical map and a sequence assembly

PacBio SMRT, Pacific Biosciences single-molecule real time.

Piercing the dark matter: bioinformatics of long-range sequencing and mapping

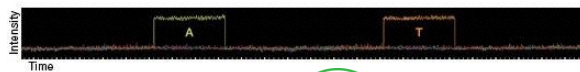
Fitz J, Seifried B, Hagan L, Chen X, A. D. and Mohler C, Schatz G  
 NATURE REVIEWS | GENETICS

# PacBio sequencing



Key points:

- 1 DNA molecule and 1 polymerase in each well (zero-mode waveguide, ZMW)
- 4 colours flash in realtime as polymerase acts
- Methylated bases have distinct pattern
- No *theoretical* limit to DNA fragment length



Slides from Ira Hall, via Jared Simpson

# PacBio sequencing

Advantages:

- Long read lengths
- Few systematic errors
- Can detect some base modifications
- Can get more accuracy, if you loop over the ZMW multiple times (consensus)

Disadvantages:

- High read error rate
- High cost per-base



**Pacbio Sequel**  
 >10kbp read length  
 Up to 10GB yield

Slides from Ira Hall, via Jared Simpson

# Nanopore sequencing

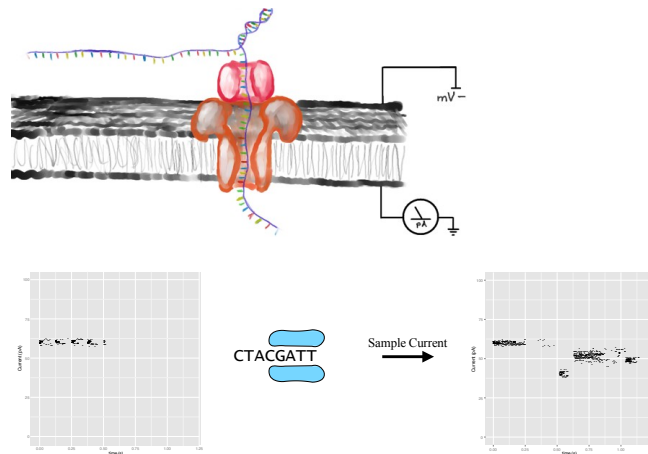


Illustration by David Eccles

# Nanopore sequencing



Oxford Nanopore MinION  
>10kbp read length  
5-15GB yield



Oxford Nanopore PromethION  
>10kbp read length  
30-90GB yield per flowcell

## Advantages:

- Portability and low capital cost
- Read lengths up to 1Mbp reported
- Base modification detection
- Sequence RNA directly

## Disadvantages:

- High error rate
- Systematic errors around homopolymers

Slides from Ira Hall, via Jared Simpson

# 10X Genomics Chromium Genome

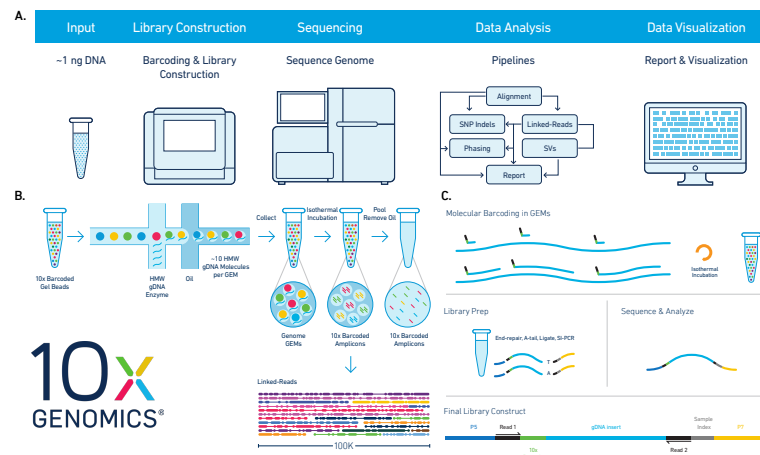
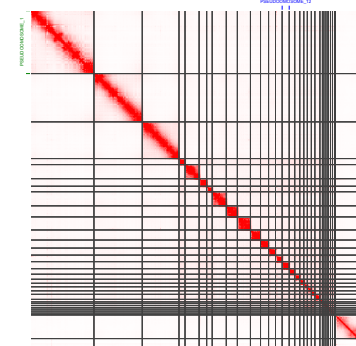
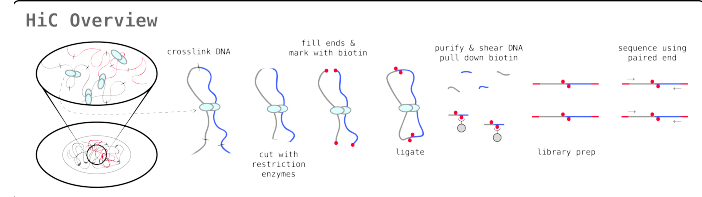
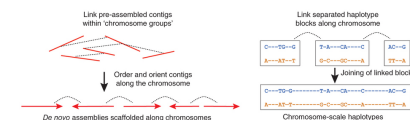


Figure 1. Chromium™ Genome Solution. (a) The Chromium Genome Solution provides a streamlined workflow and easy-to-use informatics pipeline and software for whole genome analysis. (b) An automated microfluidic system allows for functionalized gel beads to be combined with high molecular weight DNA (HMW gDNA) and oil to form a 'Gel Bead in Emulsion' (GEM). Each GEM contains ~10 molecules of HMW gDNA and primers with unique barcodes. Isothermal incubation allows for the addition of a unique barcode to all DNA within the GEM. (c) Assay schematic overview.

# Hi-C



Lieverman-Aiden et al., 2010



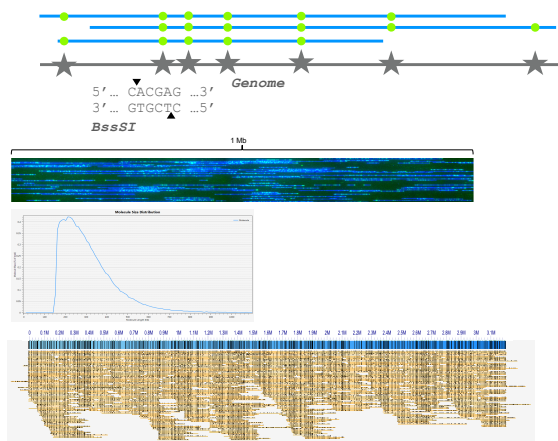
Korbel and Lee, 2013

Juicer provides a one-click system for analyzing loop-resolution Hi-C experiments

Navn C. Durand<sup>1,2,3,4</sup>, Muhammad S. Shamir<sup>1,2,3,4</sup>, Mo Haythya<sup>1,2,3,4</sup>, Suresh S. P. Rao<sup>1,2,3,4</sup>, Mitron H. Hantley<sup>1,2,3,4</sup>, Eric S. Lander<sup>1,2,3,4</sup>, and Eric Lieberman-Aiden<sup>1,2,3,4</sup>

# Bionano optical mapping

- isolate high molecular weight DNA fragments
- label specific sequence motifs across the entire genome
- linearise and image labelled DNA



# Long-range technologies

Table 1 | Long-range sequencing and mapping platforms

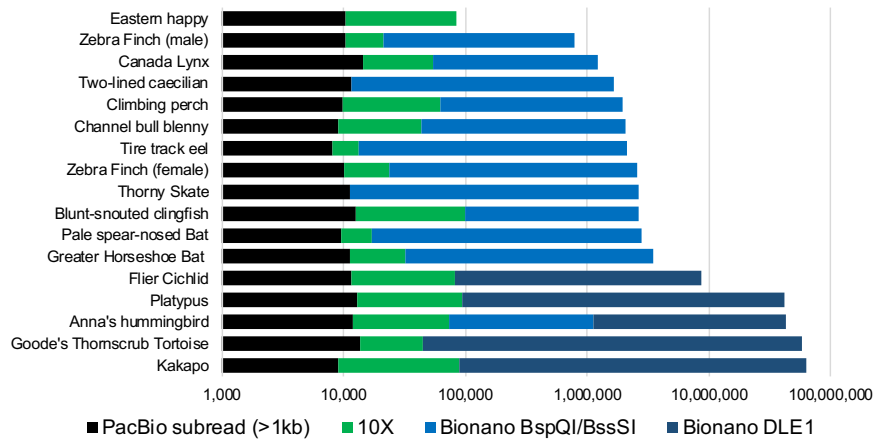
Platform	General characteristics and costs	Major applications	Bioinformatics challenges
PacBio SMRT sequencing	Single-molecule long reads averaging ~10 kb with some approaching 100 kb; several fold more expensive than short reads	De novo genome assembly, structural variant detection, gene isoform resolution and epigenetic modifications	Raw reads have high error rates dominated by false insertions; requires new alignment and error correction algorithms
Oxford Nanopore sequencing	Single-molecule long reads averaging ~10 kb with some >1 Mb; several fold more expensive than short reads	De novo genome assembly, structural variant detection, gene isoform resolution and epigenetic modifications	Raw reads have high error rates dominated by false deletions and homopolymer errors; requires new alignment and error correction algorithms
10X Genomics Chromium	Linked reads spanning ~100 kb derived from a collection of short-read sequences; moderately more expensive than short reads	De novo genome assembly and scaffolding, phasing, detection of large structural variants (>10 kb) and single-cell gene expression	Sparse sequencing rather than true long reads; more complicated to align, with poorer resolution of locally repetitive sequences
Hi-C-based analysis	Pairs of short reads (<100 bp) formed from crosslinking chromatin interactions; moderately more expensive than short reads	Genome scaffolding and phasing	Sparse sequencing with highly variable genomic distance between pairs (1 kb to 1 Mb or longer)
BioNano Genomics optical mapping	Optical mapping of long DNA molecules (~250 kb or longer) labelled with fluorescent probes; less expensive than short reads	Genome scaffolding and detection of large structural variants (>10 kb)	Limited algorithms to discover high-confidence alignment between an optical map and a sequence assembly

PacBio SMRT, Pacific Biosciences single-molecule real time.

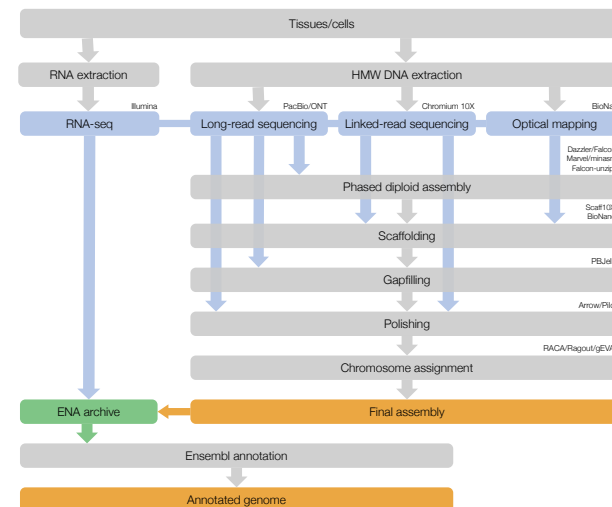
Piercing the dark matter: bioinformatics of long-range sequencing and mapping

Fritz J. Sedlitzki<sup>1,2</sup>, Hagen Loh<sup>1</sup>, Christiane A. Darby<sup>1</sup> and Michael C. Schatz<sup>1,3\*</sup>  
NATURE REVIEWS | GENETICS

# Long-range technologies



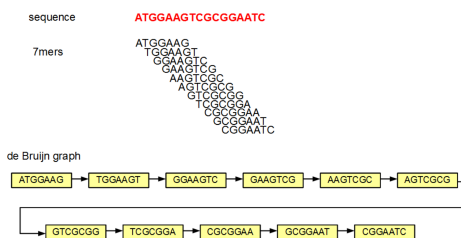
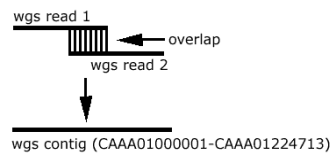
# (Vertebrate) Genome project workflow



# Contig generation

## OLC (Overlap Layout Consensus):

- For all pairs  $x, y$ , of reads
  - determine if there is sufficient overlap
  - bundle stretches of overlap graph into contigs
- Computationally expensive (quadratic scaling with current approaches)
- Assembly software
  - Falcon (PacBio), Canu (PacBio, ONT), minimap/miniasm



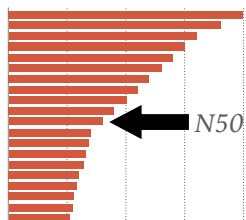
## DBG: de Bruijn graph (k-mer):

- Build a graph of all subsequences of length  $k$ .
- Assembly software
  - velvet, ABySS, SPAdes, wtdbg2

# Assembly metrics

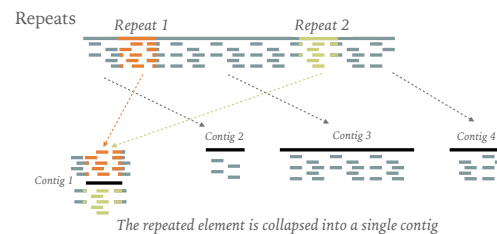
- total length
- number of sequences (contigs and scaffolds)
- average length (contigs and scaffolds)
- largest/smallest (contigs and scaffolds)
- $N50 = X$  means 50% of the genome is in sequences larger than  $X$
- NG50 ( $N50$  scaled by the expected genome size)
- Gene content (% conserved core genes mapped)

$N50 =$  what is the smallest contig at 50% of genome?

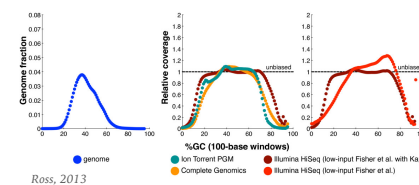
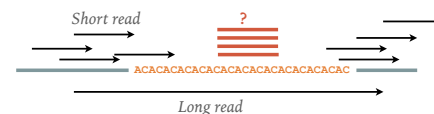


# Hurdles

- Heterozygosity
- Sequencing errors
- Repeats
- Low complexity genomic regions
- Base composition and sequencing bias



Adapted from Torsten Seemann (2014 talk)



Ross, 2013

# Scaffolding

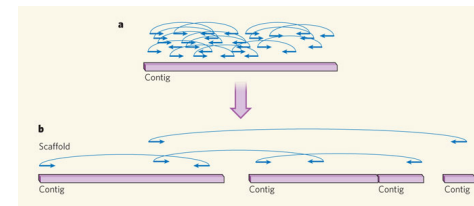
Goal: Order and orient the contigs into larger structure

Construct libraries of varying insert sizes

- Smaller size (2, 4 or 6 Kb), intermediate size (10-40 Kb), and libraries with large insert sequences (>100 Kb)
- Ends of these clones are sequenced, generating sequence reads.

Sources of evidence

- Mate-pair illumina libraries
- Fosmid ends
- Bacterial artificial chromosomes
- 10x Genomics linked reads
- Hi-C
- Optical maps



Gaps consisting of  $N$  (unknown) bases are inserted between contigs

# (Pseudo) chromosome assignment

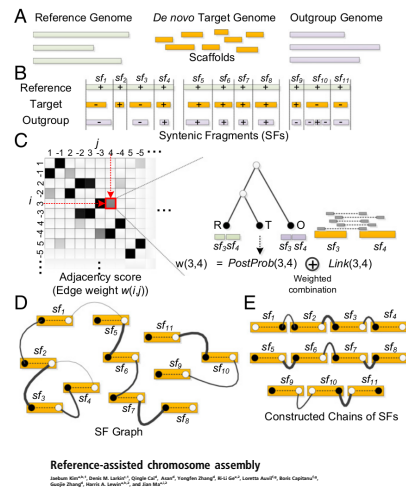
Goal: Assembly of chromosome-scale DNA fragments from scaffolds

Sources of evidence

- A close reference genome and/or outgroup genome
- Genetic markers or map

RACA software

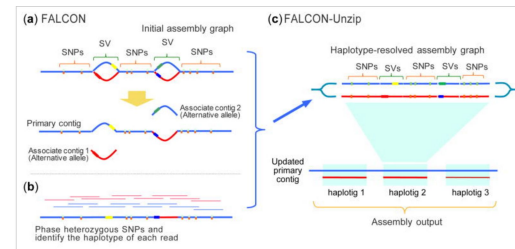
- Input: Reference genome, de novo assembly, and one or more outgroups (A)
- Identify syntenic fragments (B)
- Calculate likelihood of adjacencies (C)



# Diploid assembly

Almost all assemblers assume genome is haploid

- Ignore allelic variation between parental chromosomes
- Diploid assembly: Produce two haplotype phased genomes
- e.g. Falcon-unzip

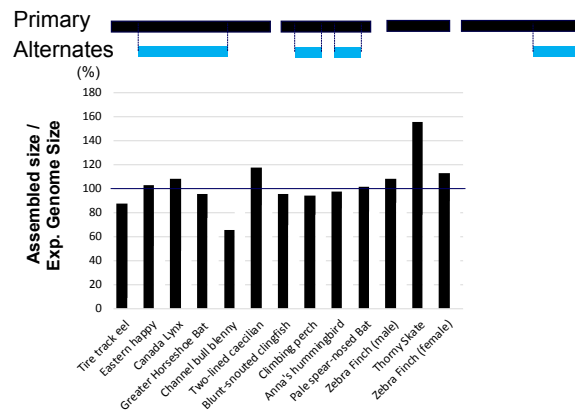


FALCON and FALCON-Unzip overview

(a) The initial assembly is computed by FALCON, which error corrects the raw reads (not shown) and then assembles using a string graph of the read overlaps. The assembled contigs are further refined by FALCON-Unzip into the final set of contigs and haplotigs. (b) Phase heterozygous SNPs and group reads by haplotype (c) The phased reads are used to open up the haplotype-fused path and generate as output a set of primary contigs and associated haplotigs.

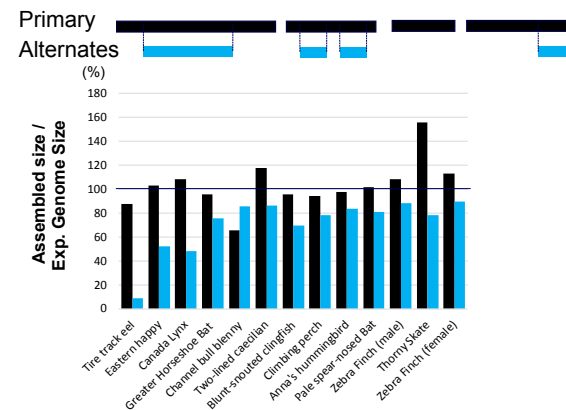
Chin et al. (2016)  
Nat Methods

# Heterozygosity and allelic duplication



Arang Rhie

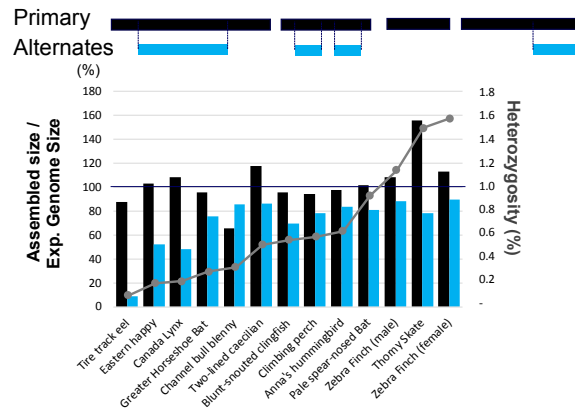
# Heterozygosity and allelic duplication



Arang Rhie

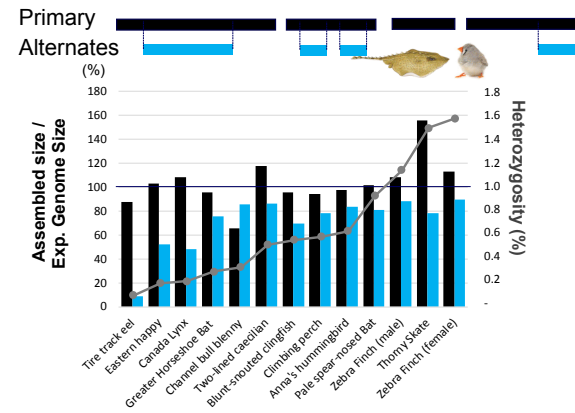


# Heterozygosity and allelic duplication



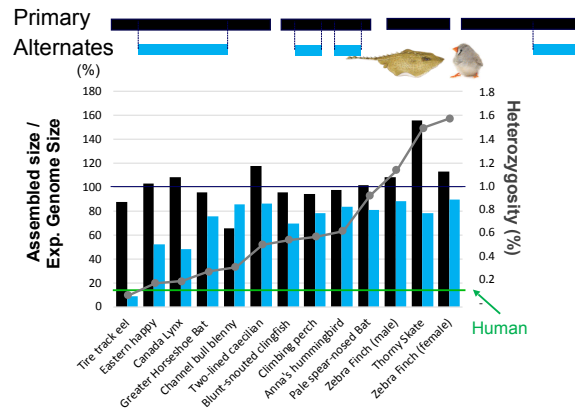
Arang Rhie

# Heterozygosity and allelic duplication



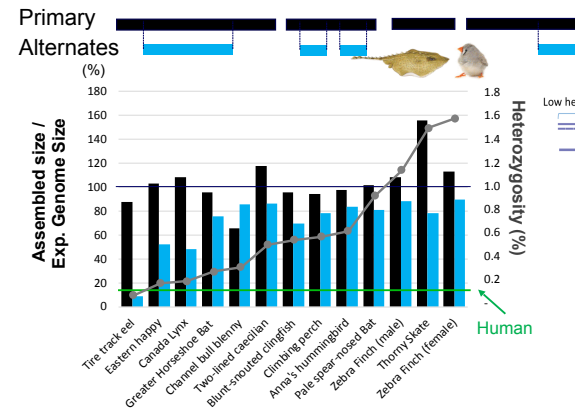
Arang Rhie

# Heterozygosity and allelic duplication



Arang Rhie

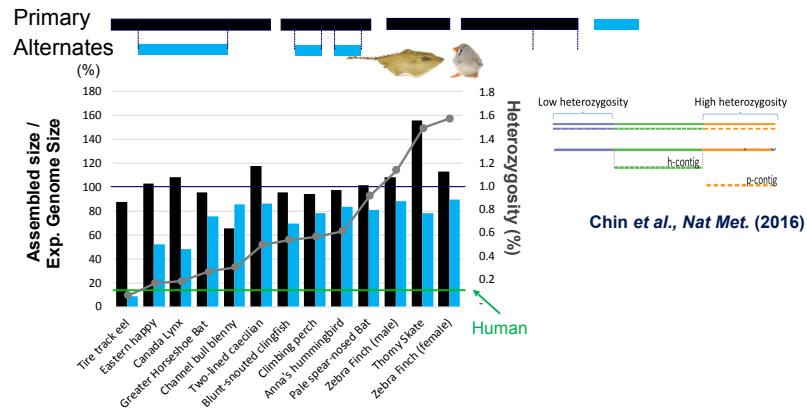
# Heterozygosity and allelic duplication



Chin et al., Nat Met. (2016)

Arang Rhie

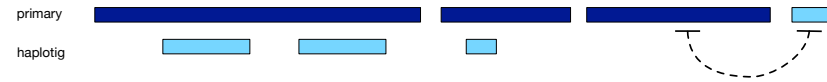
# Heterozygosity and allelic duplication



Arang Rhie

# Dealing with haplotype duplication

For high heterozygosity genomes, Falcon-unzip leaves allelic duplicates in the primary assembly



purge\_haplotigs will identify these duplicates and place them in the haplotig bin. Can remove true paralogous sequence.



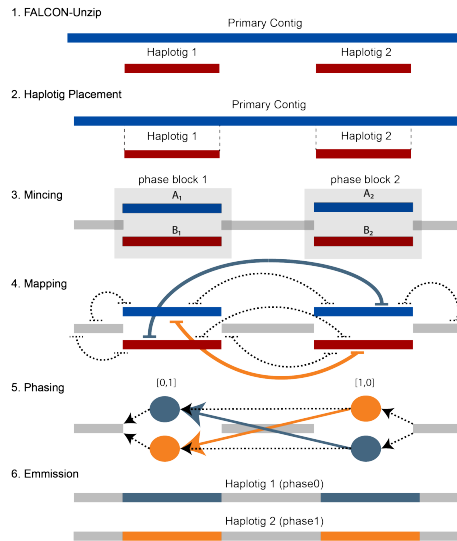
Falcon-phase will use the Hi-C data to phase, switching chunks of the primary with haplotigs to match haplotype phase



Purge Haplotigs: Synteny Reduction for Third-gen Diploid Genome  
Assemblies  
Michael J. Read\*, Simon Schmidt\* and Anthony R. Boneman\*

# Falcon-phase

Use Hi-C data to help phase variants over a longer range than Falcon-unzip can do with SNP calls from PacBio reads alone.

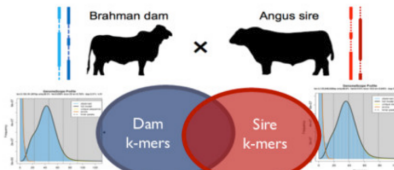


FALCON-Phase: Integrating PacBio and Hi-C data for phased diploid genomes bioRxiv: <https://doi.org/10.1101/327064>

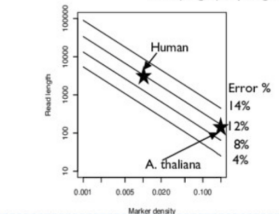
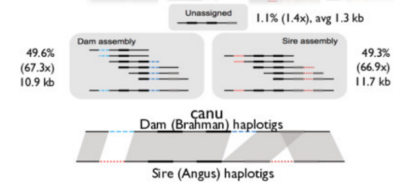
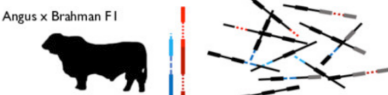
# Diploid assembly

## Trio Binning

• K-mer profiling of each parent (Illumina, 60x)



• K-mer profiling of the F1 (PacBio, 120x)



# Genome assembly QC

## Base Accuracy

- Realign reads from the same species
  - Identify SNPs+indels
  - Indels - known to be enriched in Pacbio+Nanopore assemblies

## Local structure accuracy

- External evidence
- Known adjacencies, e.g. PCR primers or structural variants

## Gene content

- Order and orientation of genes/exons of known genes (e.g. housekeeping genes)
- BUSCO: Quantitative assessment of genome assembly and annotation completeness based on evolutionarily informed expectations of gene content

### **BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs**

Felipe A. Simão, Robert M. Waterhouse, Panagiotis Ioannidis, Evgenia V. Kriventseva, Evgeny M. Zdobnov 

*Bioinformatics*, Volume 31, Issue 19, 1 October 2015, Pages 3210–3212,



# Module 8: Genome assembly exercises

In this module we will look at one chromosome of the lab strain of *Plasmodium falciparum*, the IT clone. We have sequenced the genome with PacBio and Illumina.

## A: Starting the PacBio de novo assembly

First we are going to start the PacBio assembly using the “canu” program. It first corrects the reads and then uses the Celera assembler to merge the long reads into contigs.

- Navigate to the data directory (`~/course_data/Module8_Assembly`)
- The pre-filtered PacBio reads are called **PBReads.fastq** - have a look at the contents of this file (`less -S PBReads.fastq`). [What do you notice compared to the Illumina fastq files you have seen earlier in the week?](#)
- Now we will start the assembly with canu (<https://canu.readthedocs.io/>). This will take some time, so we will start it now in the background and hopefully it will be done while we complete the other exercises.

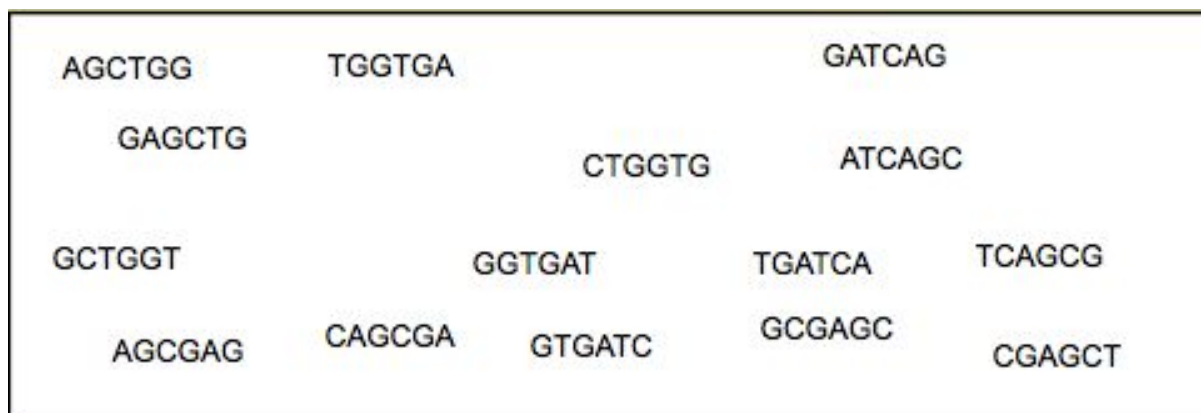
```
$ canu -p PB -d Pacbio -s file.specs -pacbio-raw PBReads.fastq &>
output.txt &
```

The “-p” option sets the prefix of output files to “PB”, while the “-d” option sets the output directory to “PacBio”. The ‘&’ at the end will set this command running in the background while you work on the following sections.

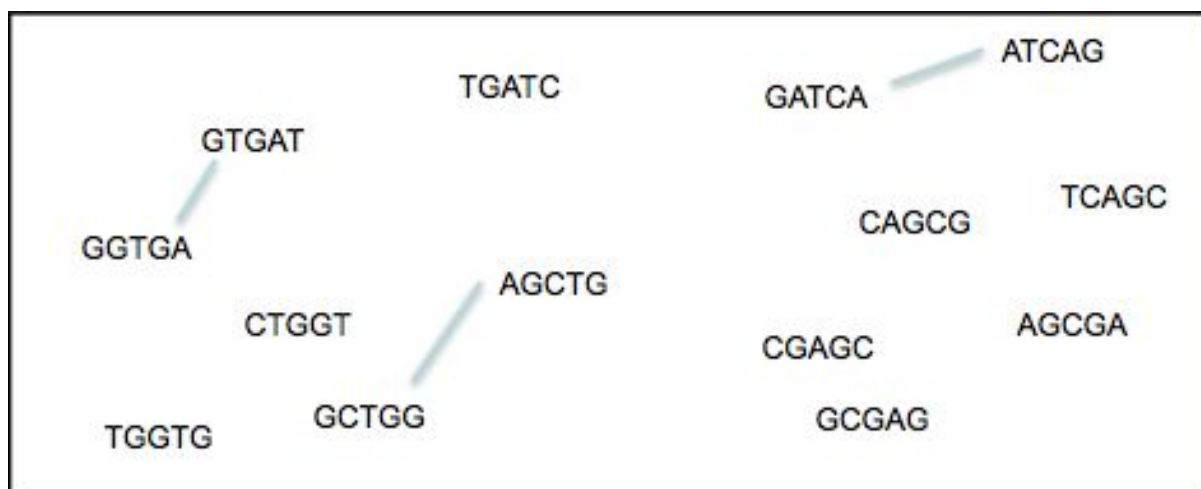
Before we move on, let’s just make sure the program is indeed running. Using the ‘top’ or better ‘htop’ command will show you all processes running on your machine (type “q” to exit top). You should hopefully see processes associated with canu running (maybe something called “meryl”). We can also check the “output.txt” file where the canu logs will be written. If we see error messages in there, then something has gone wrong.

## B: Doing a de Bruijn graph by hand

Here we are going to do an example of the de Bruijn graph by hand! Sit together with your neighbour and build the graph from the reads and find the contig(s).



Use a  $k=5$ , we get the following k-mers from these reads. To finish the graph, join k-mers that overlap by 4 bases.



What is the contig sequence?

What was tricky here?

Where does the contig start?

## C: Making an Illumina assembly

We are going to use the assembler **velvet** (<https://www.ebi.ac.uk/~zerbino/velvet/>) to assemble the Illumina reads. Our Illumina reads are from the same sample we used to generate the PacBio data.

To start

```
$ velveth k.assembly.49 49 -shortPaired -fastq -separate  
IT.Chr5_1.fastq IT.Chr5_2.fastq
```

49 is the k-mer size. “k.assembly.49” is the name of the directory where the results are going to be written. The other options specify the type of the input data. With the following command you can see all possible options, but don’t be afraid, not all must be used.

```
$ velveth
```

Now the assembler has to build the graph and find the path, as we did before in the exercise:

```
$ velvetg k.assembly.49 -exp_cov auto -ins_length 350
```

The first parameter specifies the working directory. The second is to let velvet find the median read coverage rather than specify it yourself. Last, the insert size of the library is given. There is a lot of output, but the most important is in the last line:

**Final graph has 978 nodes and n50 of 10508, max 54529, total 1374552, using 1397134/1510408 reads.**

(Your exact result might differ depending on the velvet version used).

This line first gives you a quick idea of the result. 978 nodes are in the final graph. An n50 of 10508 means that 50% of the assembly is in contigs of at least 10508 bases, it is the median contig size. This number is most commonly used as an indicator of assembly quality. The higher, the better! (but not always!) “Max” is the length of the longest contig. “Total” is the size of the assembly, here 1347kb. The last two numbers tell us how many reads were used from the 7.5 million pairs.

---

That wasn't too bad! Now we have to try to improve the assembly a bit. The kmer size has the biggest impact. Also the "-cov\_cutoff" parameter can play a role. This means that nodes with less than a specific k-mer count are deleted from the graph. More parameters can be changed, but we would run out of time. In the beginning the changes look a bit random, but with more experience, you will get a feeling for them.

First rerun velvet with a k-mer size of 49. As parts of the graph are already done, the program will run far quicker. velveth doesn't need to be rerun.

```
$ velvetg k.assembly.49 -exp_cov auto -ins_length 350  
-min_contig_lgth 200 -cov_cutoff 5
```

Maybe do assemblies for different k-mer sizes i.e. 55, 41, here the example is a k-mer length of 55

```
$ velveth k.assembly.55 55 -shortPaired -fastq -separate \  
IT.Chr5_1.fastq IT.Chr5_2.fastq
```

```
$ velvetg k.assembly.55 -exp_cov auto -ins_length 350  
-min_contig_lgth 200 -cov_cutoff 5
```

Write down the results for each assembly made using different k-mer sizes. Which one looks the best?:

k-mer	nodes	n50	largest contig
41			
49			
55			

If you want to play with other parameters, like the "-min\_pair\_count", go for it. All the options can be seen by typing:

```
$ velvetg
```

---

All the results are written into the directory you specified, e.g. b. The final contigs are in contigs.fa. The **stats.txt** file holds some information about each contig, its length, the coverage, etc. The other files contain information for the assembler.

Another way to get more stats from all the runs is to use a little program called “assembly-stats”. It displays the number of contigs, the mean size and a lot of other numbers. It might help to pick “the best” assembly

Just type:

```
$ assembly-stats k.*/*.fa
```

```
stats for k.assembly.41/contigs.fa
sum = 1435372, n = 199, ave = 7212.92, largest = 75293
N50 = 22282, n = 19
N60 = 16569, n = 27
N70 = 13251, n = 37
N80 = 9535, n = 49
N90 = 4730, n = 69
N100 = 202, n = 199
N_count = 51974
-----
stats for k.assembly.49/contigs.fa
sum = 1452034, n = 175, ave = 8297.34, largest = 85317
N50 = 28400, n = 17
N60 = 26582, n = 23
N70 = 16485, n = 29
N80 = 12065, n = 39
N90 = 6173, n = 55
N100 = 202, n = 175
N_count = 57000
-----
stats for k.assembly.55/contigs.fa
sum = 1461496, n = 181, ave = 8074.56, largest = 71214
N50 = 28059, n = 19
N60 = 22967, n = 25
N70 = 14871, n = 33
N80 = 11360, n = 44
N90 = 4885, n = 64
N100 = 205, n = 181
N_count = 69532
```

It looks that the best choice is a k-mer size of 49. The n50, average contig size and the largest contigs have the highest values, while contig number is the lowest. Before we look at the assembly itself, what could the N\_count mean?

---

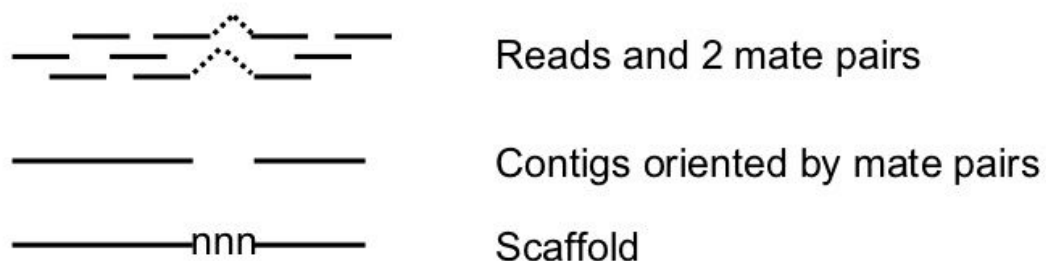


As we discussed before, DNA templates can be sequenced from both ends, resulting in mate pairs. Their outer distance is the insert size. Imagine mapping the reads back onto the assembled contigs. In some cases the two mates don't map onto the same contig. We can use those mates to scaffold the two contigs e.g. orientate them to each other and put N's between them, so that the insert size is correct, if enough mate pairs suggest that join. Velvet does this automatically (although you can turn it off). The number of mates you need to join two contigs is defined by the parameter `-min_pair_count`.

Here is the description:

`-min_pair_count <integer>` : minimum number of paired end connections to justify the scaffolding of two long contigs (default: 5)

Here a schema:



It might be worth mentioning, that incorrect scaffolding is the most common source of error in assembly (so called miss-assemblies). If you lower the `min_pair_count` too much, the likelihood of generating errors increases.

Other errors are due to repeats. In a normal assembly one would expect that the repeats are all collapsed, if they are smaller than the read length. If the repeat unit is smaller than the insert size, than it is possible to scaffold over it, leaving the space for the repeats with N's.

To get the statistic for the contigs, rather than supercontigs, you can use following command:

```
$ seqtk cutN -n1 k.assembly.49/contigs.fa > tmp.contigs.fasta
$ assembly-stats tmp.contigs.fasta
```

depending from which assembly you would like the statistics. [How does the contig N50 compare to the scaffold N50?](#)

## D: What to expect from a genome assembly?

We are lucky with this test dataset in that we have a known reference genome and some expectations about the size and composition of the *P. falciparum* genome. How can we get at this for new genomes we haven't sequenced before? One way is to look at k-mer distributions. Genomescope (<http://qb.cshl.edu/genomescope/>) will model the single copy k-mers as heterozygotes, while double copy kmers will be the homozygous portions of the genome. It will also estimate haploid genome size.

Let's check with our *P. falciparum* Illumina data that the k-mer. To get a distribution of 21-mers, we use "jellyfish":

```
$ jellyfish count -C -m21 -s6G -t4 -o IT.jf <(cat IT.Chr5_1.fastq
IT.Chr5_2.fastq)
$ jellyfish histo IT.jf > IT.histo
```

Then we analyse with genomescope<sup>1</sup>:

```
$ Rscript genomescope.R IT.histo 21 76 IT.jf21
```

Where 76 is the read length of our input Illumina data and IT.jf21 is the output directory. The output is summarised in a "summary.txt" file. [What is the predicted heterozygosity? What is the predicted genome size? Does this seem reasonable?](#)

You should also find an image like below. Notice the bump to right of the main peak. These are the repeated sequences.

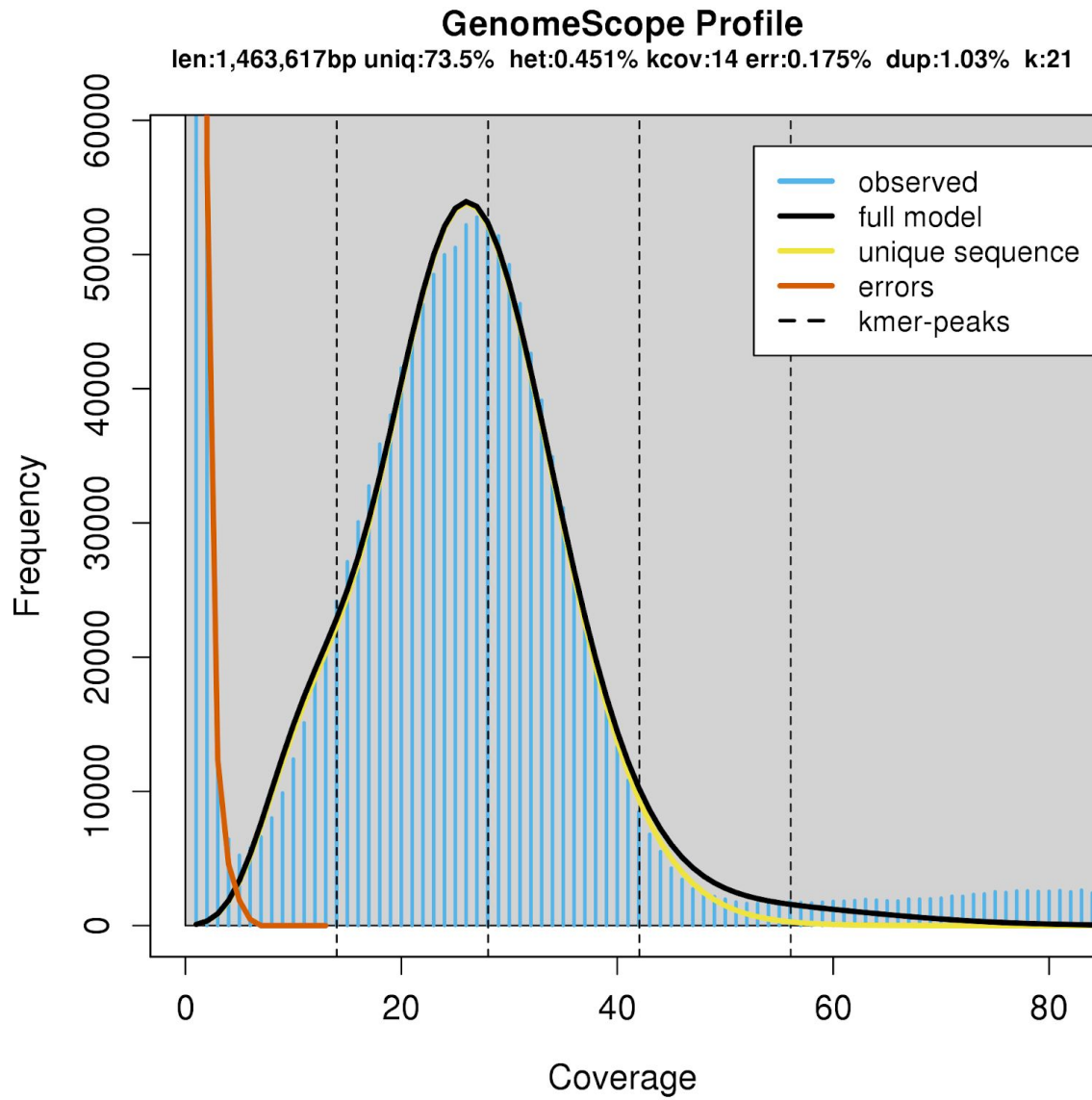
We have some k-mer histogram for a handful of other species in the data directory. Try running genomescope on these. **Read length for all of these is 150bp.**

- fMasArm1.jf21.histo
  - You should see a nice tight diploid peak for this sample. It has very low heterozygosity - similar to human.
- fAnaTes1.jf21.histo
  - [What is the bulge to the left of the main peak here?](#)
- fDreSAT1.jf21.histo
  - [What is the striking feature of this genome?](#)

---

<sup>1</sup> If genomescope does not work, go to the genomescope website above and you can upload your \*.histo files to get the plots

- fSalTru1.jf21.histo
  - This genome was actually haploid. How do we interpret the features in the genomescope profile?



## E: Back to our PacBio assembly

So our Illumina assembly is ok, but not perfect. Let's have a look at the PacBio assembly, which should have hopefully finished by now (check the output.txt log).

Now use the assembly-stats script to look at the stats of the assembly. [What do you think?](#)

```
$ assembly-stats Pacbio/PB.contigs.fasta
```

Another long read assembler based on de Bruijn graphs is "wtdbg" (<https://github.com/ruanjue/wtdbg2>). Let's try to build this assembly too.

```
$ wtdbg2 -t4 -i PBReads.fastq -o wtdbg
$ wtpoa-cns -t4 -i wtdbg.ctg.lay -fo wtdbg.ctg.lay.fa
$ assembly-stats wtdbg.ctg.lay.fa
```

[How does it compare?](#)

They may be similar in contig number and N50, but are they really similar. Let's map the Illumina reads to each, call variants and compare.

```
$ bwa index Pacbio/PB.contigs.fasta
$ samtools faidx Pacbio/PB.contigs.fasta
$ bwa mem -t4 Pacbio/PB.contigs.fasta IT.Chr5_1.fastq
IT.Chr5_2.fastq | samtools sort -@4 - | samtools mpileup -f
Pacbio/PB.contigs.fasta -ug - | bcftools call -mv > PB.vcf
```

Do the same for wtdbg.ctg.lay.fa and then compare some basic stats.

```
$ bcftools stats PB.vcf | grep ^SN
$ bcftools stats wtdbg.vcf | grep ^SN
```

[What do you notice in terms of the number of SNP and indel calls?](#)

The wtdbg assembly has more variants due to having more errors. This is mainly due to a lack of error correction step - something built into the canu assembly pipeline.

## D: Polishing

Even with the canu assembly, there are still deviations from the Illumina data.

```
$ bgzip -c PB.vcf > PB.vcf.gz
$ tabix PB.vcf.gz
$ bcftools consensus -f Pacbio/PB.contigs.fasta PB.vcf.gz >
PB.contigs.polished.fasta
```

Map, and variant call like above (bwa index/bwa mem/samtools sort/samtools mpileup/bcftools call) using this polished reference. [When running on this new output, do we still get variants?](#)

## E: Reference based Assembly

Velvet has an option to use a reference to help to resolve repetitive regions. When velvet cannot resolve a repetitive region in the de Bruijn graph, it can look where read with this k-mer are mapping in the reference. This way it is possible to untangle the graph to simplify finding the path in the graph. This module is called “velvet columbus”.

Although it is still work in progress, the results seem to be better than the normal assembly. The only difference would be to include: “-reference -fasta Pf3D7\_05.fasta” in the velveth call. Important is to map the reads against the reference what we already did.

Run it as follow:

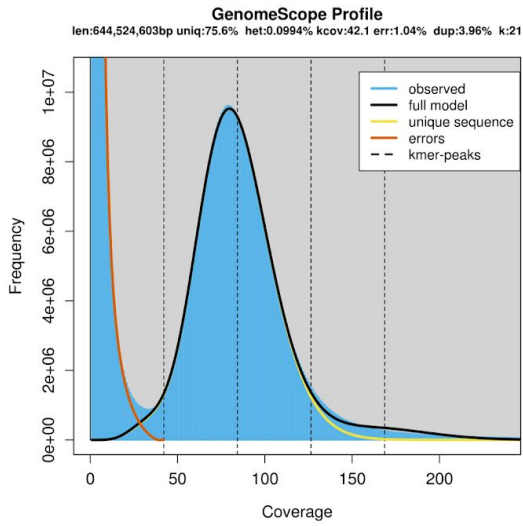
```
$ velveth k.columbus.55 55 -reference -fasta Pf3D7_05.fasta
-shortPaired -fastq -separate IT.Chr5_1.fastq IT.Chr5_2.fastq

$ velvetg k.columbus.55 -exp_cov auto -ins_length 350 -
min_contig_lgth 200 -cov_cutoff 5
```

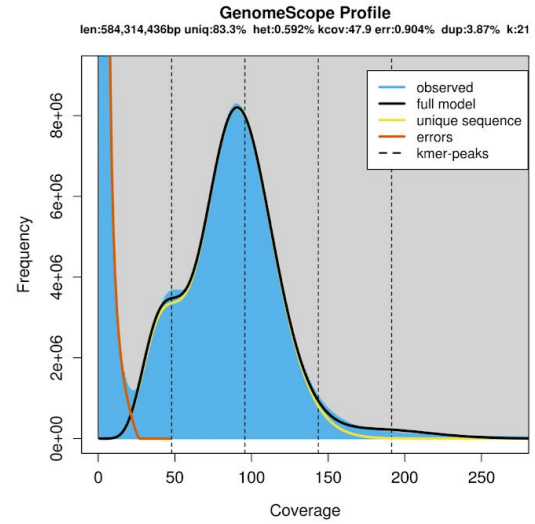
Now compare the results with the assembly-stats program. Are they better?

```
$ assembly-stats k.*/*.fa
```

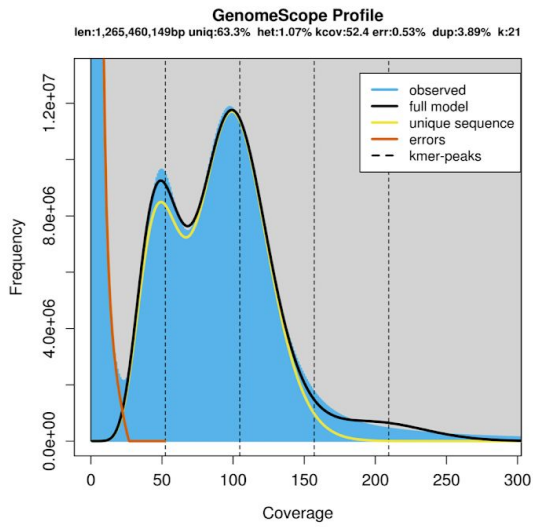
# fMasArm1



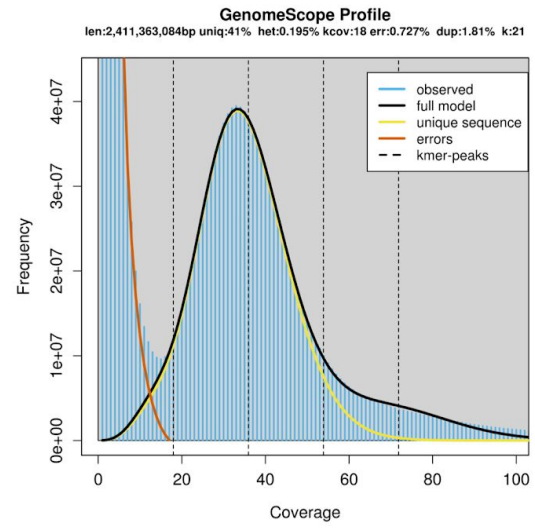
# fAnaTes1



# fDreSAT1



# fSalTru1



# 1 Introduction to Integrated Genome Viewer (IGV)

## 1.1 A quick start guide

## 1.2 Introduction

**Integrative Genome Viewer (IGV)** allows you to visualise genomic datasets. This quick start guide will give you a brief overview of IGV, how to load data, navigate the genome and visualise your data. The IGV user guide is really useful and contains information on many more features than we have the chance go through in this quick start guide.

### **Integrative Genome Viewer (IGV)**

*Broad Institute and the Regents of the University of California*

**Download:** <https://software.broadinstitute.org/software/igv/download>

**User guide:** <http://software.broadinstitute.org/software/igv/UserGuide>

## 1.3 Learning outcomes

By the end of this quick start guide you can expect to be able to:

- Index a reference genome for IGV
- Load a reference genome file into IGV
- Load gene annotations into IGV
- Load alignment files into IGV
- Navigate a genome in IGV

## 1.4 Authors

This tutorial was written by [Victoria Offord](#).

## 1.5 Prerequisites

This guide assumes that you have the following software or packages and their dependencies installed on your computer. The software or packages used in this guide may be updated from time to time so, we have also given you the version which was used when writing the guide.

Package name	Link for download/installation instructions	Version
samtools	<a href="https://github.com/samtools/samtools">https://github.com/samtools/samtools</a>	1.6
IGV	( <a href="https://software.broadinstitute.org/software/igv/">https://software.broadinstitute.org/software/igv/</a> )	2.3.90

## 1.6 Indexing a reference genome for IGV

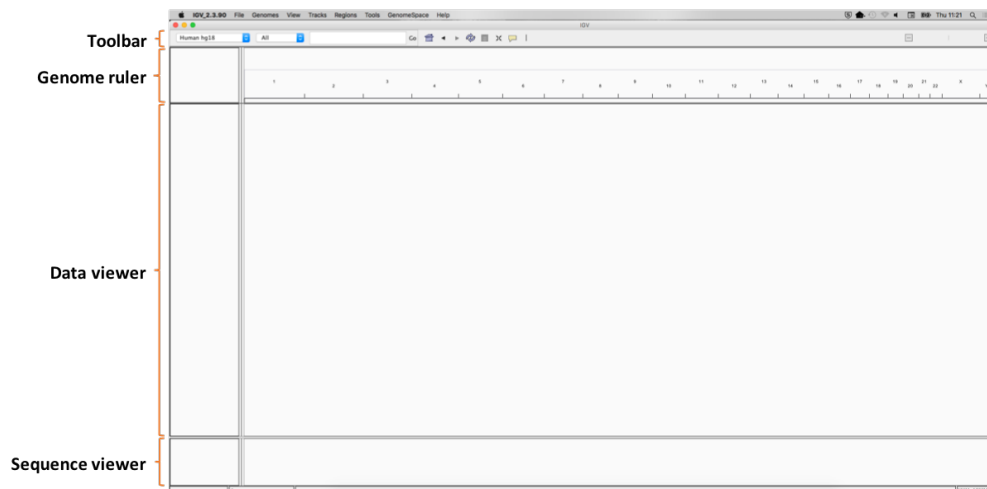
Before you begin, make sure you have an index for the reference genome which IGV will use to traverse the genome. You can do this using [samtools](#).

```
samtools faidx <your_genome_file.fa>
```

The resulting index file will have the extension **.fai** and must be in the same directory as the reference genome.

## 1.7 IGV main window

When you start IGV, it will open a main window. At the top of this window you have a **toolbar** and **genome ruler** for navigation. The largest area in the main window is the **data viewer** where your alignments, annotations and other data will be displayed. To do this, IGV uses horizontal rows called **tracks**. Finally, at the bottom, there is a sequence viewer which contains the base level information for your reference genome.

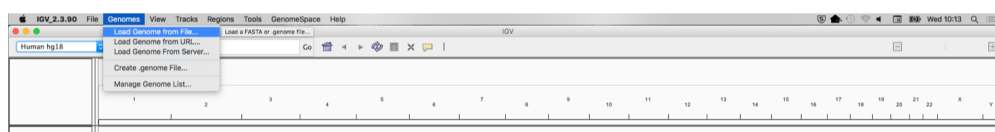


IGV - main window

## 1.8 Loading a reference genome

IGV provides several genomes which can be selected with the "Genome drop-down box" on the toolbar. However, your reference genome may not always be on this list. When your reference is not available, you will need to load it from a FASTA file.

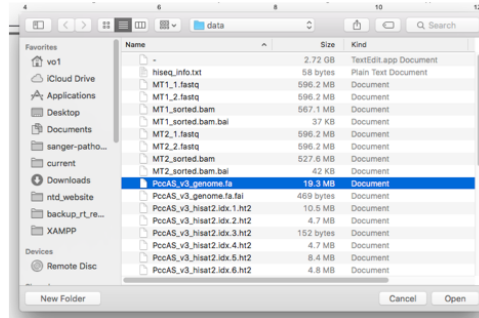
To load a reference genome from file, go to "Genomes -> Load Genome from File...".



IGV - loading genome from file



Select the FASTA file containing the reference genome and click "*Open*".



IGV - loading genome from file

### 1.8.1 IGV toolbar and genome ruler

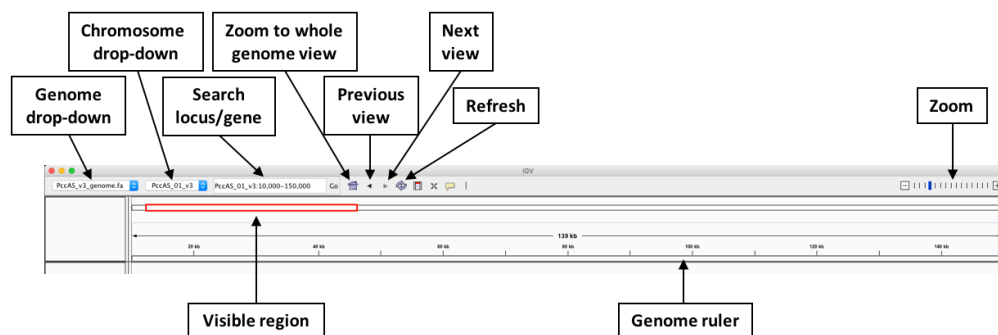
Once the genome has loaded, the chromosomes will be shown on the **genome ruler** with their names/numbers above. When a region is selected, a red box will appear. This represents the visible region of the genome.

Above the genome ruler is the **toolbar** which has a variety of controls for navigating the genome:

- **Genome drop-down** - load a genome
- **Chromosome drop-down** - zoom to a chromosome
- **Search** - zoom to a chromosome, locus or gene

There are several other buttons which can be used to control the visible portion of the genome.

- **Whole genome** - zoom back out to whole genome view
- **Previous/next view** - move backward/forward through views (like the back/forward buttons in a web browser)
- **Refresh** - refresh the display
- **Zoom** - zooms in (+) / out (-) on a chromosome



IGV - toolbar and genome ruler

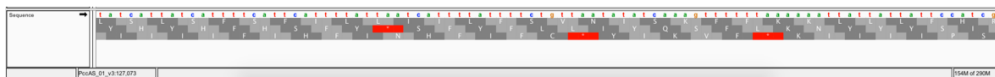
## 1.8.2 Sequence viewer

The **sequence viewer** shows the genome at the single nucleotide level. You won't be able to see the sequence until you are zoomed in. As you start to zoom in (+), you will see that each nucleotide is represented by a coloured bar (red=T, yellow=g, blue=c and green=a). This makes it easier to spot repetitive regions in the genome. Carry on zooming in (+) and you will see the individual nucleotides.



IGV - sequence viewer

If you right-click on "Sequence" at the left-hand side of the sequence viewer and click "Show translation", you will also see the amino acid sequence for the forward three reading frames.



IGV - sequence translation

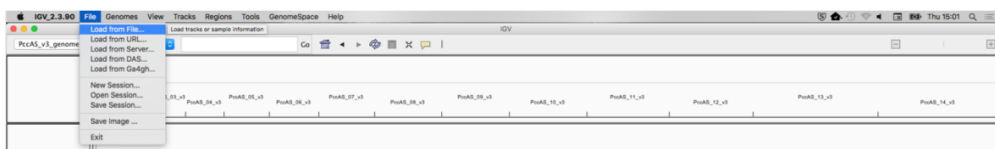
You can also see the reverse three reading frames by right-clicking on the track and selecting "Flip strand".

*Note: at the bottom right of the main window is the amount of memory available to IGV and how much of this it is currently using - always keep a wary eye on this!!*

## 1.9 Loading gene annotations

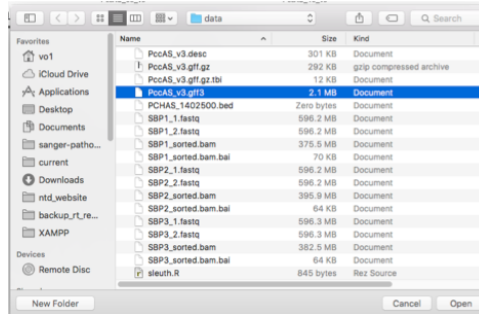
In addition to your genome, you will probably want to load an annotation file that contains information such as gene locations and gene structures (e.g. introns/exons/CDS).

**To load a GFF file containing annotations, go to "File -> Load from File...".**



IGV - loading annotation

Select the annotation file and click "Open".



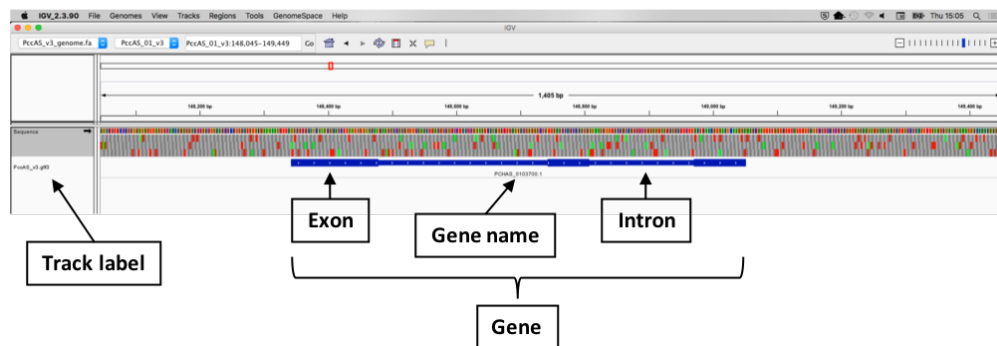
### IGV - loading annotation

This will load the **annotation track**. At the genome level, you will see this shown as a density track for the associated annotation. On the left you will see the track label which is the name of the file you just loaded. You can change this label to something more recognisable by right clicking on the label and selecting "*Rename Track*".



### IGV - annotation density

As you zoom in (+), you will start to be able to see the individual genes (shown in blue).

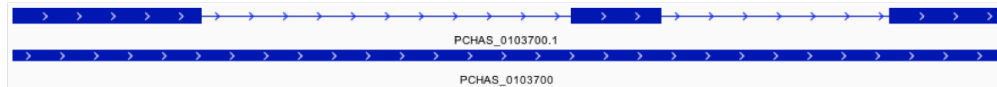


### IGV - annotation track

#### 1.9.1 Gene structure

Genes are represented in blue as boxes (exonic regions) and lines (intronic regions). The arrows indicate the strand of the direction in which the gene will be transcribed. The box height indicates whether the region is a coding sequence (taller) or untranslated region (thinner).

For a clearer view of the gene structure, right click on the annotation track and click "*Expanded*".



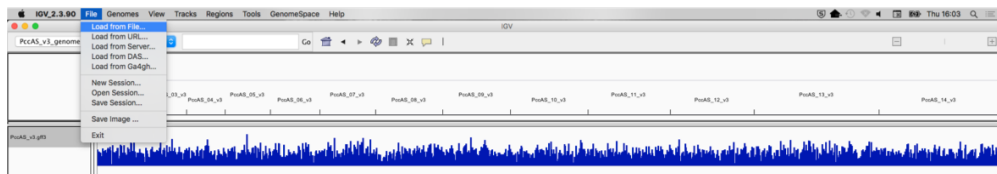
### IGV - expanded annotation

Now you will see the annotated isoforms and can more clearly see the arrows that indicate which strand the gene is on. If you zoom in further, you will also see the amino acid sequence superimposed onto the exons.

## 1.10 Loading alignment files

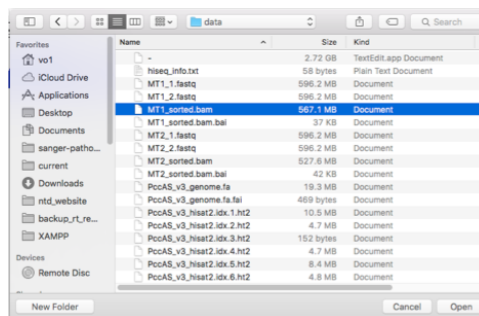
IGV can be used to visualise many different types of data, including read alignments. Each time you load an alignment file it will be added to the **data viewer** as a new major track.

To load a read alignment file, go to "File -> Load from File...".



### IGV - loading alignment from file

Select a sorted BAM file and click "Open".

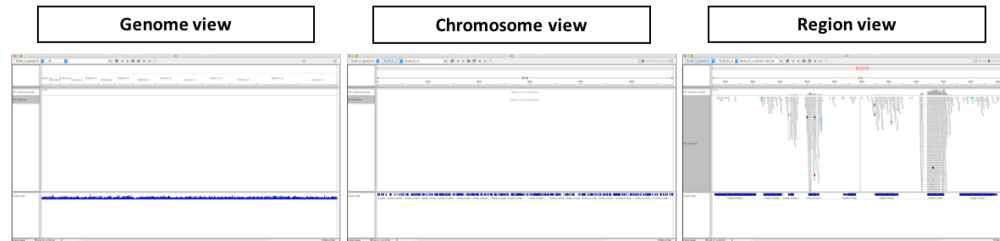


### IGV - loading alignment from file

*Note: BAM files and their corresponding index files must be in the same directory for IGV to load them properly.*

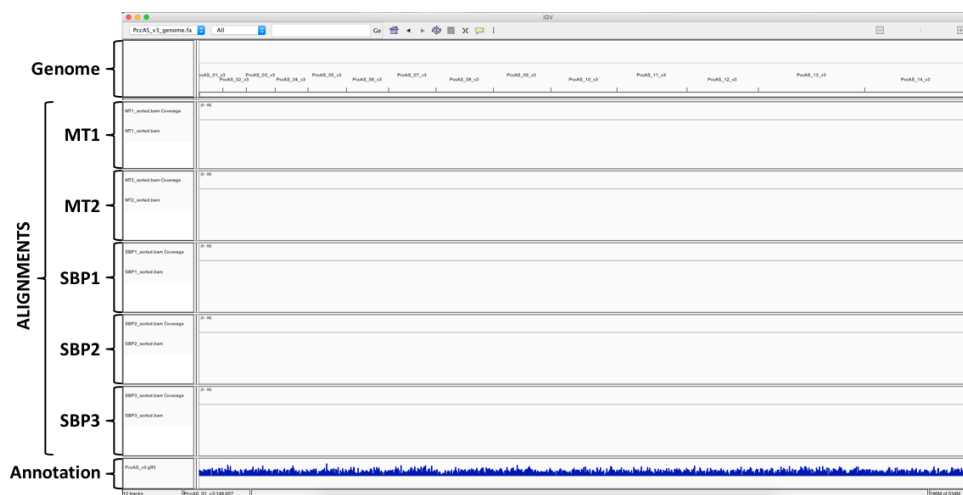
For each read alignment, a major track will appear containing two minor tracks for that sample: coverage statistics and read alignments. For the total number of visible tracks, see the bottom left of main window.

At the genome level, there will be no coverage plot or read alignments visible. At the chromosome level, there are two messages displayed: Zoom in to see coverage/alignments. Finally, once you have zoomed in (+) you will see a density plot in the coverage track and your read alignments.



IGV - alignment views

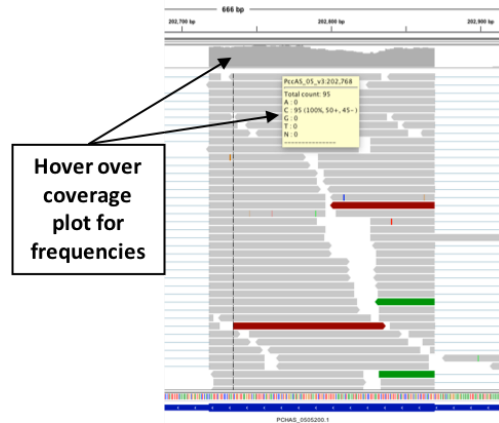
You can open more than one alignment file. Each alignment file will be loaded into a new track with its coverage statistics and read alignments. However, make sure you keep an eye on the memory usage in the bottom right corner or IGV may crash!



IGV - all alignments

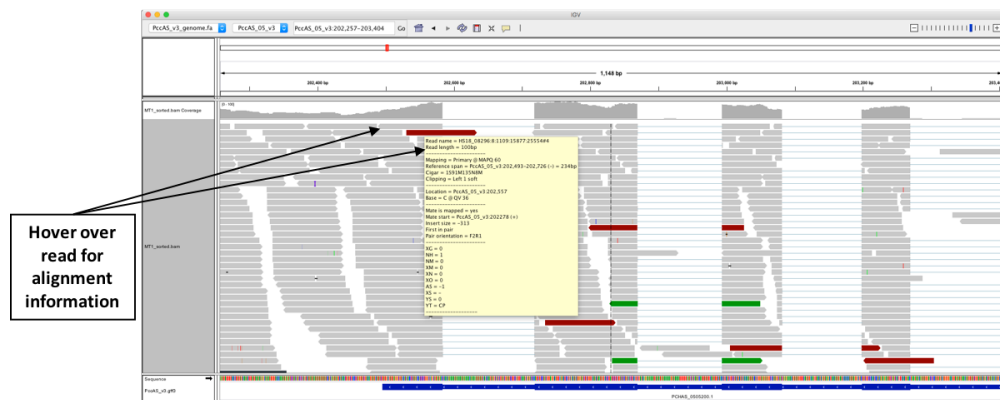
### 1.10.1 Visualising alignments

**Coverage information** When zoomed in to view a region, you can get alignment information for each position in the genome by hovering over the coverage track. This will open a yellow box which tells you the total number of reads mapped at that position, a breakdown of the mapped nucleotide frequencies and the number of reads mapping in a forward/reverse orientation. In our example, 95 reads mapped, 50 forward and 45 reverse, all of which called A at position 202,768 on chromosome PccAS\_05\_v3.



IGV - coverage information

**Viewing individual read alignment information** Reads are represented by grey or transparent/white bars which are stacked together where they align to the reference genome. Reads are pointed to indicate the orientation in which they mapped i.e. on the forward or reverse strand. Hovering over an individual read will display information about its alignment.



IGV - read information

Mismatches occur where the nucleotide in the aligned read is not the same as the nucleotide in that position on the reference genome. A mismatch is indicated by a coloured bar at the relevant position on the read. The colour of the bar represents the mismatched base in the read (red=T, yellow=G, blue=C and green=A).



IGV - mismatch

For more information about how reads are coloured and what this means, see the [IGV user guide](#).

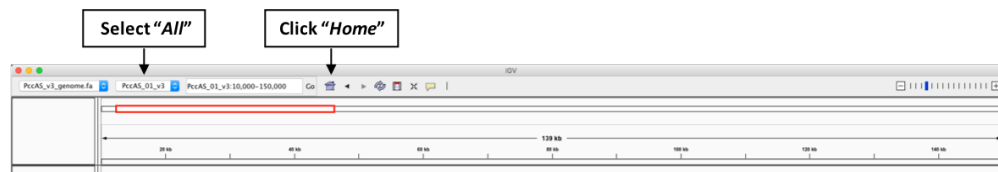
## 1.11 Navigating the genome

### 1.11.1 Whole genome view

In IGV you can navigate through different levels of visualisation, from the whole genome, all the way down to a base level resolution.

To return to the whole genome view:

- Select "All" from the chromosome drop-down
- Click the "Home" button

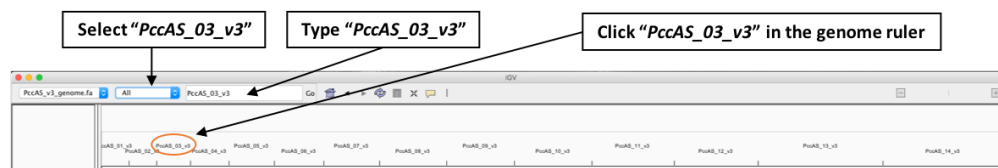


IGV - whole genome view

### 1.11.2 Chromosome view

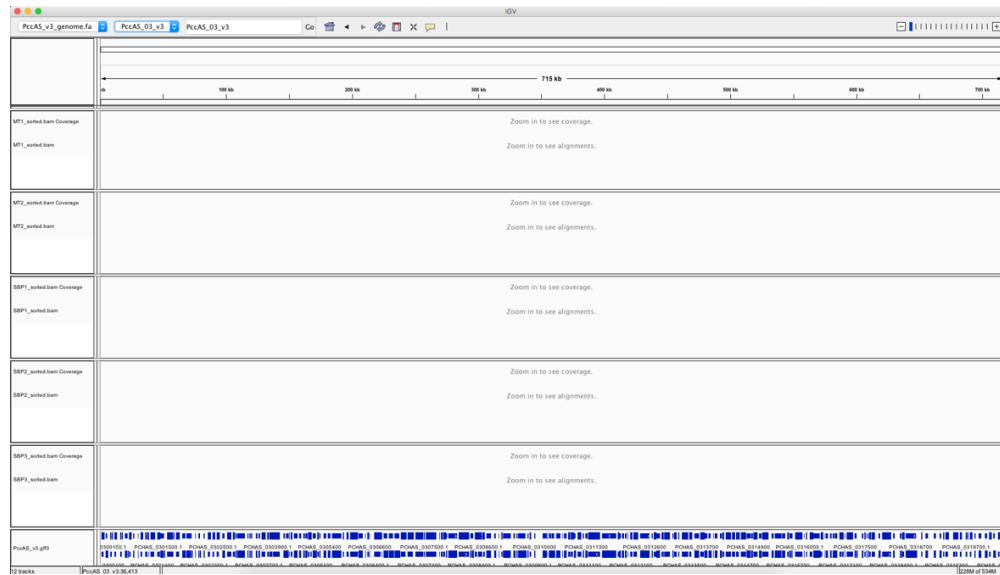
You can also view each of the chromosomes individually. For example, to view PccAS\_03\_v3 you can:

- Select "PccAS\_03\_v3" from the chromosome drop-down
- Type "PccAS\_03\_v3" into the search box
- Click on "PccAS\_03\_v3" on the genome ruler



IGV - chromosome view

In the chromosome view, the alignment track has changed from a density plot to showing individual gene, the genome ruler is now showing co-ordinates instead of chromosome name/numbers. More importantly, the alignment tracks are saying that to see coverage or read alignments we need to zoom in further.

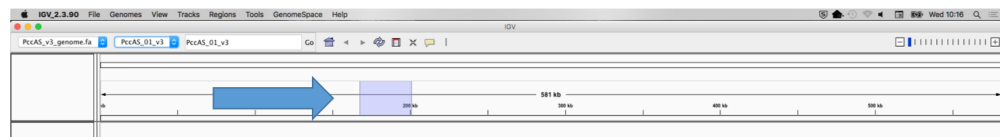


IGV - chromosome view

### 1.11.3 Region view

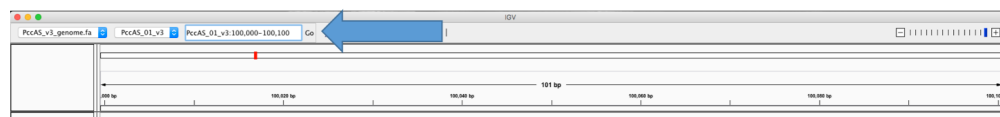
There are several ways to continue to zoom in and view specific regions or base level information.

**Select region** If you don't know the specific co-ordinates of the region you want to look at, you can click and drag to select a region on the genome toolbar.



IGV - select region

**Jump to region** If you know the co-ordinates of the region you want to view, you can enter them into the "Search" and click "Go". The format is chromosome:start-stop. For example, to view from 100,000 to 100,100 on PccAS\_01\_v3, you would enter PccAS\_01\_v3:100,000-100,100 in the search box.

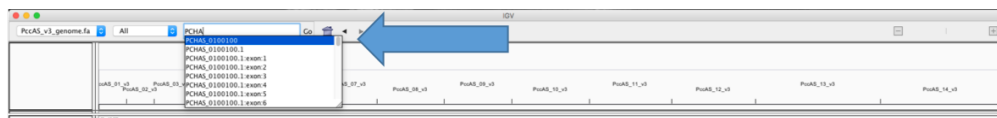


IGV - search region

*Note: the visible region of the chromosome is indicated by the red box on the genome ruler.*



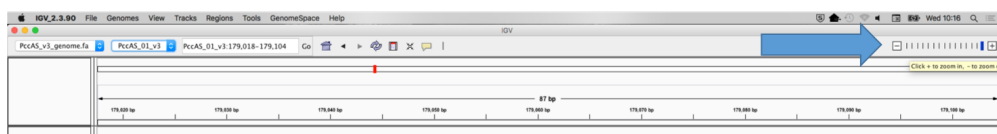
**Jump to gene or locus** Alternatively, if you know the name of the gene you want to view and you have loaded an annotation file, you can enter the gene name into the "Search" and click "Go". For example, to view PCHAS\_0100100, you would enter PccAS\_0100100 in the search box.



IGV - search gene

*Note: the search box will try to help you by listing options to autocomplete the search box.*

**Zooming in and out** You can zoom in and out from each view by using the "+" and "-" buttons on the zoom control at the right-hand side of the toolbar. This will also work with the "+" and "-" keys on your keyboard.



IGV - zoom in/out

#### 1.11.4 Navigating around the view

There are several ways you can move around the view:

- Left-click and hold on a track in the data viewer. Drag to move left or right.
- Move left right using arrow keys on your keyboard.
- Double click on a gene/feature in the annotation track to zoom in and center on that gene/feature.

