Command Line Tools

https://rstudio-pubs-static.s3.amazonaws.com/100050_1883e518f0ac4e46975feb1b56939604.html

Docker Commands:

- Docker is a tool which allows us to download docker images, which can be opened in Docker containers. Docker containers have all the dependencies needed to run some software regardless of what operating system your computer is on. A docker container is a small computer within a computer.

```
load
      docker load -i (path to tar file of image) # loads the image
      encoded in the tar file. Should return image name.
exec
      docker exec -it (name of container) /bin/bash # opens a bash
      terminal in the container
images
      docker images # checks all possible images
run
      docker run -it --name (name of container) (name of image)
      -it launches an interactive terminal within the container.

      docker run --rm -v ~/sra-data:/data ncbi/sra-tools prefetch
      SRR29400501
            # --rm removes container as soon as executing is done
            # -v maps ~/sra-data (local dir) to /data (container dir),
            which means any changes you make in one are reflected in the
            other. The /data directory is created in the container with
            this command
            # ncbi/sra-tools prefetch SRR29400501 is the docker image,
ps
      docker ps - a # shows all containers, open or not
      docker ps # shows only the open containers
stop
      docker stop (name of container or container post id) # stop the
      container
start
```

```
      docker start (name of container or container post id) # restart a
      container that you closed down
cp
      # copying files to and from the docker container
      # recommend put project files in containers /media folder

      # local file to container
      docker cp (local file) (name of container):(path in container to
      cp to)

      # container to local file
      docker cp (container name):(path in container):(path to local
      dest)


Unix Commands
date
      returns the date
echo
      echo (thing) # returns thing
pwd
      # returns current working directory
cd
      # changes where we are in the directory
.
      # current directory
..
      # parent dir
ls
      # lists all files in cd in alphabetical order
      ls -l # gives additional file info.
            # drwxr-xr-x marks a dir, -rw-r--r--@ is a file.
            # number of links to the file
            # user name of the file
            # group owner belongs to
            # file size
            # date of modification
            # file or dir name
```

```
total 3605416
drwxr-xr-x  2 arshmeetkaur  staff          64 Jun 30 16:38 Plants
-rw-r--r--@ 1 arshmeetkaur  staff  1834819584 Jun 30 10:49
gencommand_image.tar
-rw-r--r--@ 1 arshmeetkaur  staff           0 Jun 30 14:36
gencommand_image.txt
```

```
ls -lt # gives files in order in which they were created
(old->new)

# * things are called wildcards.

ls (name).* # gives any files with that name
ls (?restofname).* # gives any files with part of the name
ls[a-z]*.genome # gives you any files with .genome extension
ls p*.genome # gives you anything starting with p and .genome
ls {pear,peach}.genome # lists all files with either pear or
peach and .genome
```

```
man(command)
    # lets you see info on command options, press q to quit.
mkdir
    # makes a directory in cwd
    mkdir dir1 dir2
cp
    cp apple.* apple # copy files into a directory
    # this copies, doesn't take it out of whatever dir the file is
    already in
    cp file1 file2 apple # can copy multiple at one time.
    # if a file exists in apple, will not be copied again, also will
    not give any error
mv
    # moves files, takes them out of the directory they were in
    before
    ❯ mv pear.* pear
    ❯ ls pear
    pear.genes   pear.genome  pear.samples
```

```
rm
      # removes files
      rm f1 f2…

      # not reversible.
      rm -i # asks before removing
rmdir
      # removes directories which are empty
      rm -r Old.stuff # recursively removes everything from Old.stuff
      and then delete Old.stuff
more
      # view file
      enter to see more lines.
      q to quit.

      /> # looks for the next “>” in the file
      # Pattern not found if it doesn’t exist in the file.
less
      # view file, you can scroll up and down
head
      head (file) # returns first ten lines
      head (-number) (file) # returns first number lines
tail
      tail (file) # returns last ten lines
      tail (-number) (file) # returns last number lines
cat
      # concatenate multiple files together
      cat (files) # shows all the contents of all files together.
      # or show the content of one file
      cat peach.genes # showed peach.genes

      # store the output of the cat command.
wc
      # displays num lines, num words, num chars, name file
      wc -l # displays only num lines
>
      # redirects output to a file
```

```
❯ wc peach.genome
   4499    4499  363699 peach.genome
❯ wc peach.genome > nlines
❯ cat nlines
   4499    4499  363699 peach.genome
```

<

```
# puts input from one file into the command line
❯ cat < nlines
   4499    4499  363699 peach.genome
```

|

```
# called piping.
# the output of one command becomes the input of the next

# output of ls is the input of “cat”
❯ ls
nlines        peach.genes   peach.genome  peach.samples
❯ ls | cat
nlines
peach.genes
peach.genome
peach.samples
```

sort

```
# sorts by alphabetical order

sort -r months # sorts reverse alphabetical

sort -k 2 months # sorts in terms of column 2
# for numbers, still doing alphabetical unless you run this
sort -k 2n months
sort -k 2nr months # reverse order.
sort -k 3 -k 2n months # sorts by col 3 first, then col 2

sort -u seasons # if there are multiple occurrences of each
thing, it’ll only list one
```

vi

```
# opens text editor
```

```
    # press i to edit
nano
    # opens text editor
cut
    cut -d' ' -f1,2 months # cuts col 1 and 2, only shows those
    cut -d' ' -f1-3 months # cuts col 1 through 3, only shows those
    # ' ' because everything is separated by spaces, not tabs
    # can choose whatever you want as the delimiter
uniq
    # uniq will collapse multiple lines in a row containing the same
    stuff into just one line with it.
```

```
❯ uniq seasons
winter
spring
summer
fall
winter
```

```
    uniq -c seasons # tells you the occurrences of each string in the
    seasons file, but it only does this if identical lines are next
    to each other (file is already sorted)
```

```
❯ touch test
❯ uniq -c test
   1 Apple
   1 Banana
   1 Apple %
❯ sort test
Apple
Apple
Banana
❯ -uniq c test
zsh: command not found: -uniq
❯ uniq -c test
   1 Apple
```

```
   1 Banana
   1 Apple %
```

grep

grep root */*.samples # shows us everything from the root containing samples

grep “ 12 winter” months # looks for pattern in files

```
❯ grep "12 winter" months
december 12 winter

❯ grep -n "12 winter" months # gives you the line number.
12:december 12 winter
```

When you use grep with the -c option (grep -c "pattern" filename), it counts the total number of lines that contain the specified pattern, regardless of how many times the pattern appears on each line.

grep -c “ “ # counts lines containing that char
grep -o “ “ # searches for each occurrence of that character and prints it on a newline

diff

diff file1 file2
compare files line by line

```
❯ diff orchard orchard.1
2d1 # line 2 was deleted in the file1 right after line 1 in file 2
< The smell of cool, damp earth beneath my feet
❯ nano orchard.1
❯ diff orchard orchard.1
1,2c1
# change lines 1,2 in file2 to get to file1
< Under the evening stars I walk
< The smell of cool, damp earth beneath my feet
---
```

```
> Under the bright night stars I walk
```

comm

```
# shows you what is common between two files

❯ comm apple/apple.samples pear/pear.samples
# leaf and fruit for file 1, flower leaf in file 2, root in both
     flower
     leaf
          root
leaf
fruit

# by default, only shows them to be common if files are sorted
```

th

```
❯ sort apple/apple.samples > apple/apples.samples.sorted
❯ sort pear/pear.samples > pear/pear.samples.sorted
❯ comm apple/apples.samples.sorted pear/pear.samples.sorted
     flower
fruit
          leaf
          root


# ignore certain lines that only occur in one file.
comm -3 file1 file2 # ignores all common lines
comm -1 -2 file1 file2 # ignore lines from file1 and file2, only
report common lines.
# same thing all around.
```

gzip

```
#compress file
gzip apple.genome # results in apple.genome.gz
# gets rid of original file too
```

gunzip

```
# retrieves the gzip file.
```

bzip2

```
# makes more compression than gzip.
```

bunzip2

```
# reverses bzip2
```

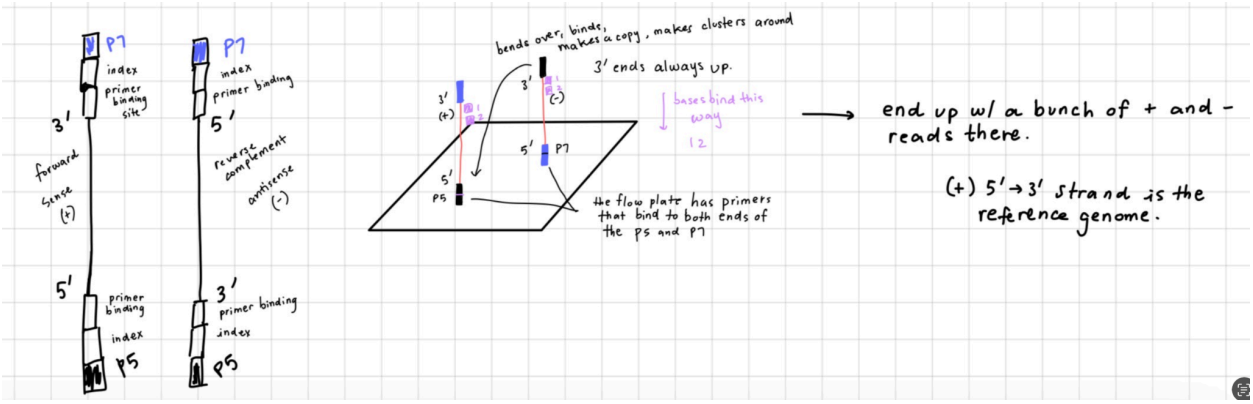
```
tar
      # -cvf compress, verbose, file after this option
      # makes a tar file containing bolded files in apple.tar.gz
      tar -cvf apple.tar apple.genes apple.genome apple.samples
      gzip apple.tar # makes apple.tar.gz

      tar gunzip apple.tar.gz # makes apple.tar
      # -xvf extract, verbose, file after this option
      tar -xvf apple.tar # returns the three files in this .tar

      # tar a whole directory (apple/ directory)
      tar -cvf AppleD.tar apple/
      # unzip with -xvf
sed
      # removes any lines in a file containing a pattern
      sed '/^$/d'# remove all newlines at the end of a file
nohup
      # no hangup - runs this process even if terminal is closed and
      puts puts output in nohup.out
&
      # runs this process in the background so the shell can be used
      for other commands in the meantime.
```

Sequences and Genomic Features 1: Molecular Bio Primer

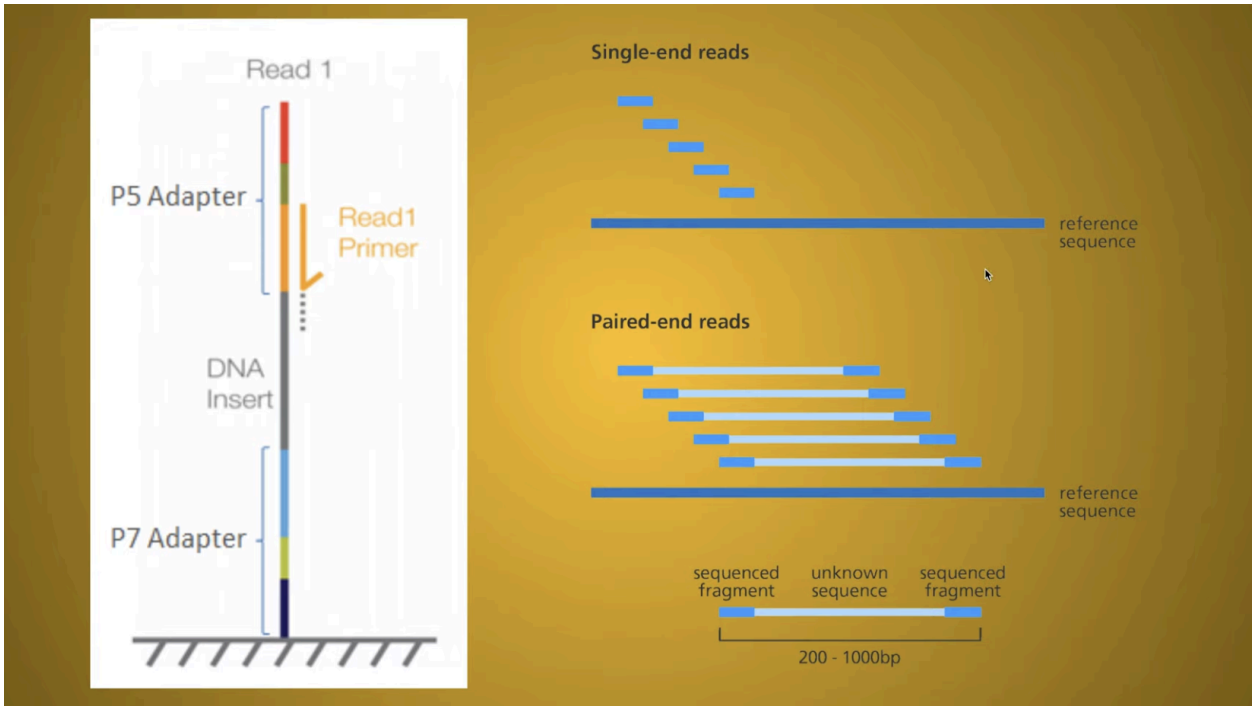
- main idea: there is a copy of the genome in the nucleus, each expressed gene will have an mRNA and the mRNA may be spliced differently (alternatively spliced) depending on which type of cell it is in in the body (liver and brain cell may express themselves differently)
- goal as computational biologist: genome sequences, mRNA, protein sequences => analyze these to understand the function in the cell

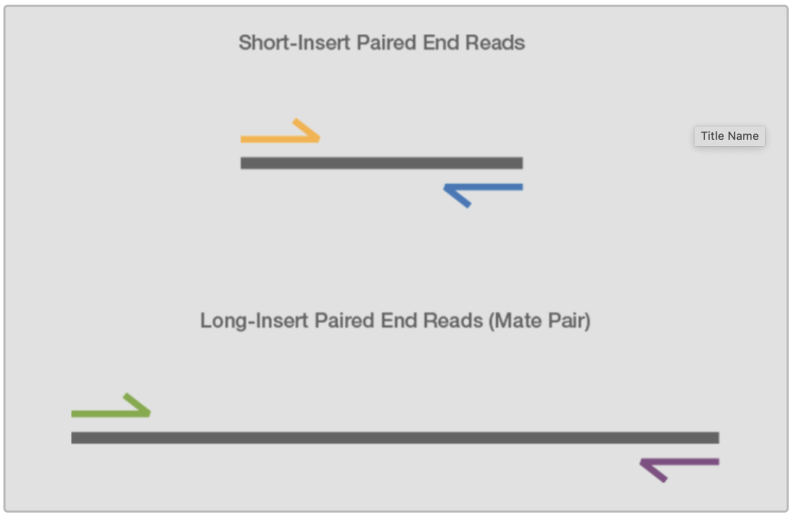
Reminder: Paired End Reads

- The sequencing process in general



- Single end sequencing reads: the sequence is read in only one direction. Reads from start to end and then it's done
- Pair end sequence reads: the sequence is read in both directions. Reads from start to end and then it reads from end to start again.
- The longer the reads, the better the alignment.



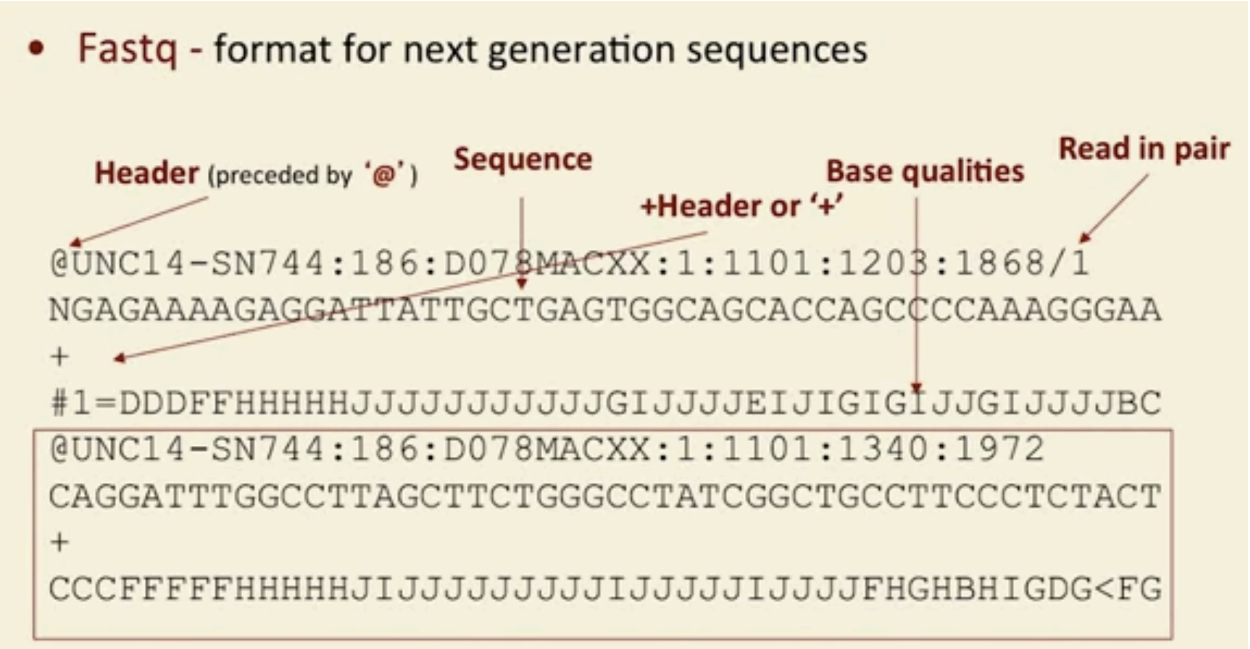


- How it works: We take a longer sequence and get two reads on the ends with some d = unknown distance between them. It's called paired end reads when the d between them is small and mate pair when the d between them is larger.
  - mates refer to the two ends of the DNA fragment that was sequenced
  - the "orientation" of these reads should always be oriented towards each other

Sequences and Genomic Features 2: Sequence Representation and Generation

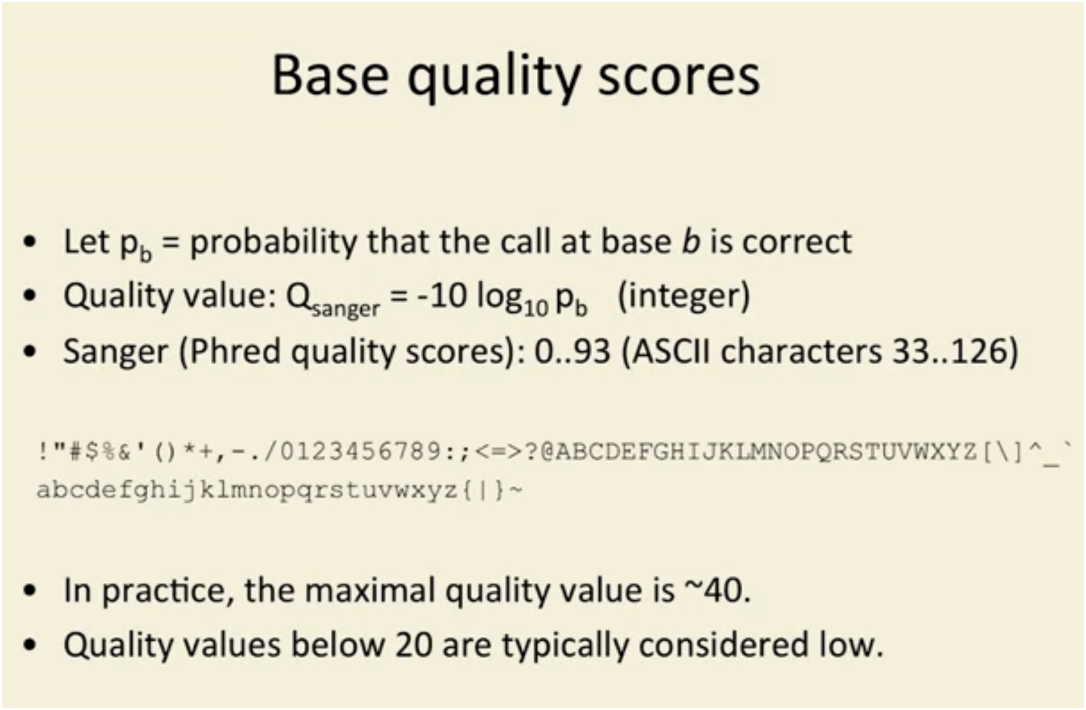
- eukaryotic cell = each cell has the full genome, for the expressed genes we have 1 or several copies of the mRNA in the nucleus or cytosol and proteins in nucleus and cytosol
- question: what do sequences look like, how many are there in the cell?

- Think of nucleotide and protein sequences as 1D. DNA and mRNA are both written in ACTG for simplicity.
- Sagner-assembled sequences are put into FASTA format with a header and then text. If there are multiple sequences, we use the multi-FASTA format.
- FASTq files:



  - Header: @identifier, the identifier gives information on location and geometry of the cluster within the sequencing array. Might have a "/1" or "/2" which indicates what read it is in a paired end read. Might also have length in the header.
  - Sequences: ACTG and N
  - +, either repeats header or just a +
  - Base qualities: one symbol for each of the bases in the sequence. This is the ASCII-encoded PHRED_33 score from before.

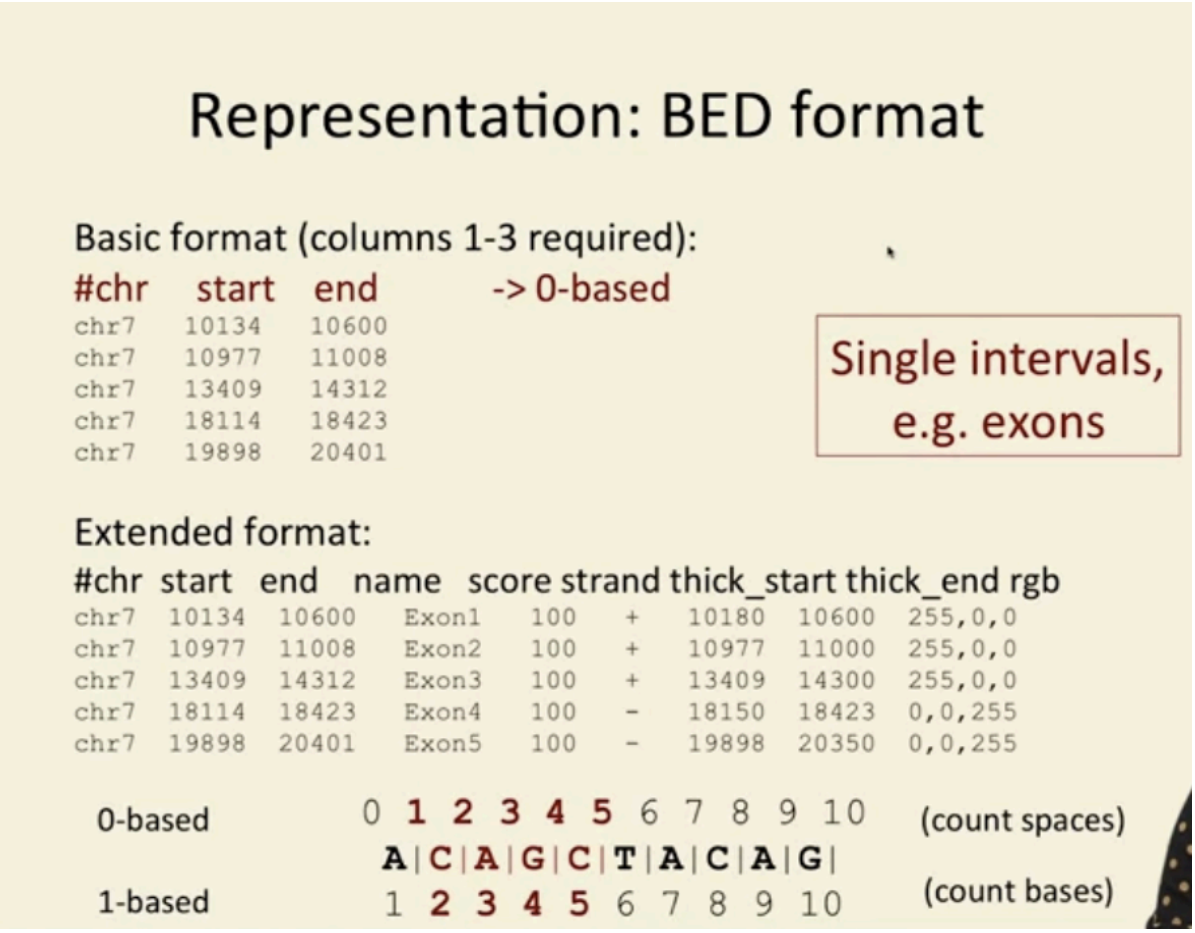
    PHRED_33 = chr(-10 log(p(call being wrong)) + 33)



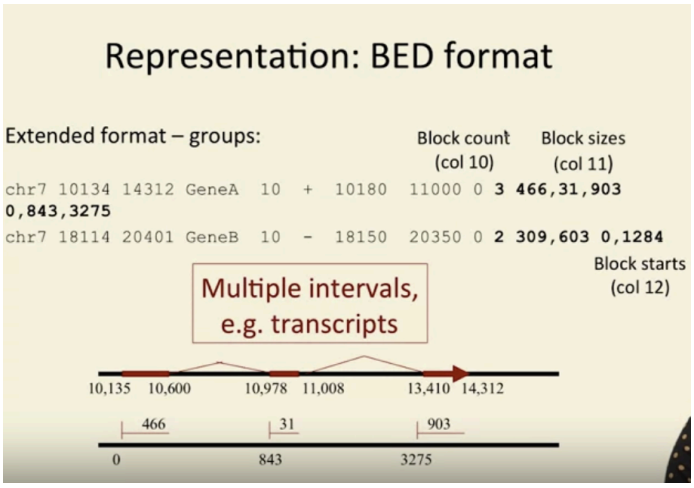
Sequences and Genomic Features 3: Annotation

- Genome annotation = determine precise location and structure of genomic features along the genome
    - genomic features = genes, promoters, protein binding sites, start/stop translation sites, etc
    - Example: when you annotate a gene, you want to know
        - exons and introns
        - + or - strand
        - start and end sites for translation

- BED format for genome features: [Frequently Asked Questions: Data File Formats]

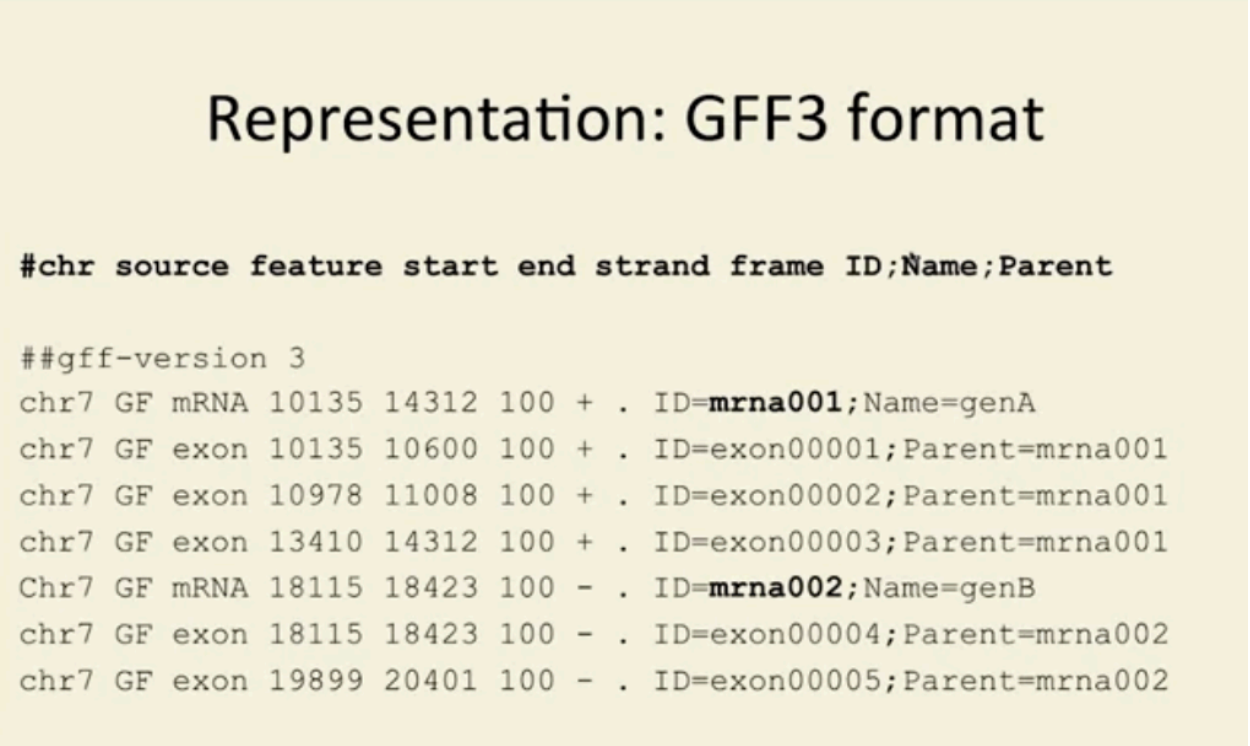


  - chr = the chromosome
  - start and end: https://genome-blog.gi.ucsc.edu/blog/2016/12/12/the-ucsc-genome-browser-coordinate-counting-systems/
    - depends on whether you're using the UCSC browser online or the BED format
    - UCSC browser: 1-start, fully-closed system
      - normal counting system. The first base is 1 and bases 1-5 refer to [1:5] so 1, 2, 3, 4, 5.
      - size = (end + 1) - start
    - BED format: 0 start
      - computer counting system. The first base is 0 and bases 1-5 refer to [0:5) so 0, 1, 2, 3, 4.
      - size = end - start
    - So if you're converting between what you see in BED format and what you see on the browser, you're going to go from:
      [start, end) to [start + 1: end]
  - name
  - score: between 0-1000, indicates how intense the color should be to the browser.

    - strand: + or -. “.” means no strand.
    - thick start and thick end: tell browser where the “direct handle” representing the feature can become thick
    - rgb: rgb rep for the color of the feature.
- We would like to show the exons that belong to the same gene together
    - put additional fields in the BED format.
    - Instead of treating each exon as its own genomic feature, make each gene a feature with a start and stop, and then add additional fields at the end with information on the gene’s exon
    - **Block count:** The number of exons in each record.
    - **Block sizes:** The size of each exon (comma separated)
    - **Block starts:** The offset of each exon, relative to each other. First exon is 0, the second is some offset, etc.
    - So a sample record could look like (regular info for gene) | 2 | 309, 603 | 0, 1284
        - 2 exons
        - exon1 length: 309, exon2 length: 603
        - exon1 position: 0, exon2 position: 1284
        - 

- GTF format = genome transfer format
    - chr
    - program = the source from a program, website, etc
    - feature = type of feature
    - start and end: [start: end] (1 start, fully closed like UCSC browser)
    - score = shading and confidence in feature
    - strand = + or -
    - frame = 0 1 or 2 or “.” if we don’t know
    - last column. gene_id; transcript id. Within the column, these are separated by “;” or spaces
    - The exons are grouped together by the gene that they are on.

- GFF3 format: Genomic feature format.

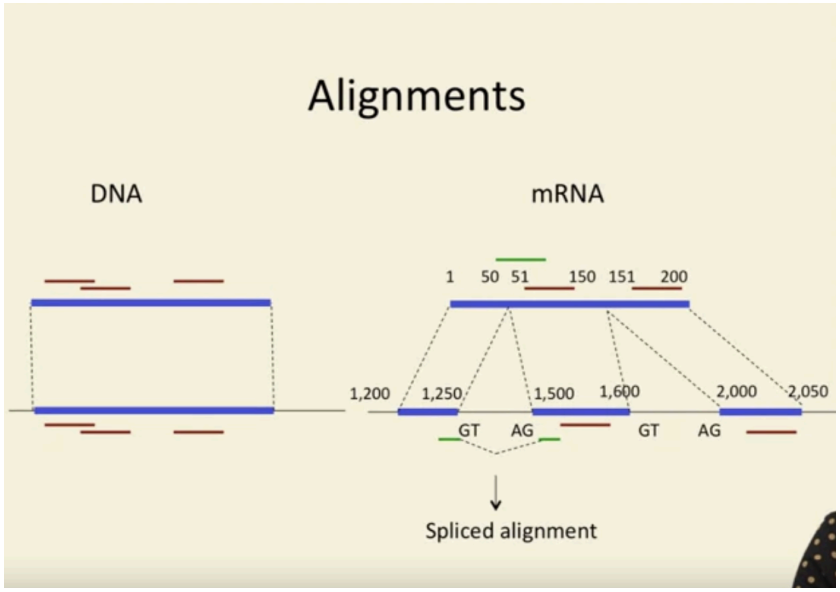


  - chrom = chr7
  - source
  - type
  - start and end [start: end]
  - strand
  - frame ID
  - Name; Parent
  - clustering: the mrna is put first and all of its exons are listed below it.

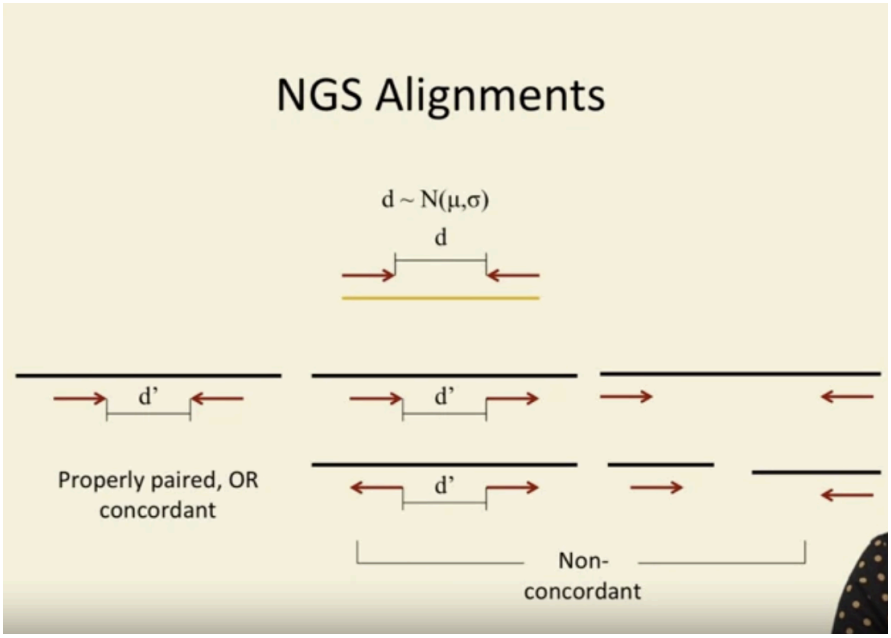
Sequences and Genomic Features 4.1: Alignment I

- We start by isolating DNA/RNA and sequencing it, and then aligning it, mapping it to a reference genome. (Read alignment - finding places in the reference genome where the sequence matches approximately with the reference genome)
  - Approximately the same because we take into account polyploidy, sequencing errors and introns
- If we start with DNA, we produce some reads and map them to the reference genome

- if we start with mRNA, we start of with an RNA that was made from spaced out parts of the DNA

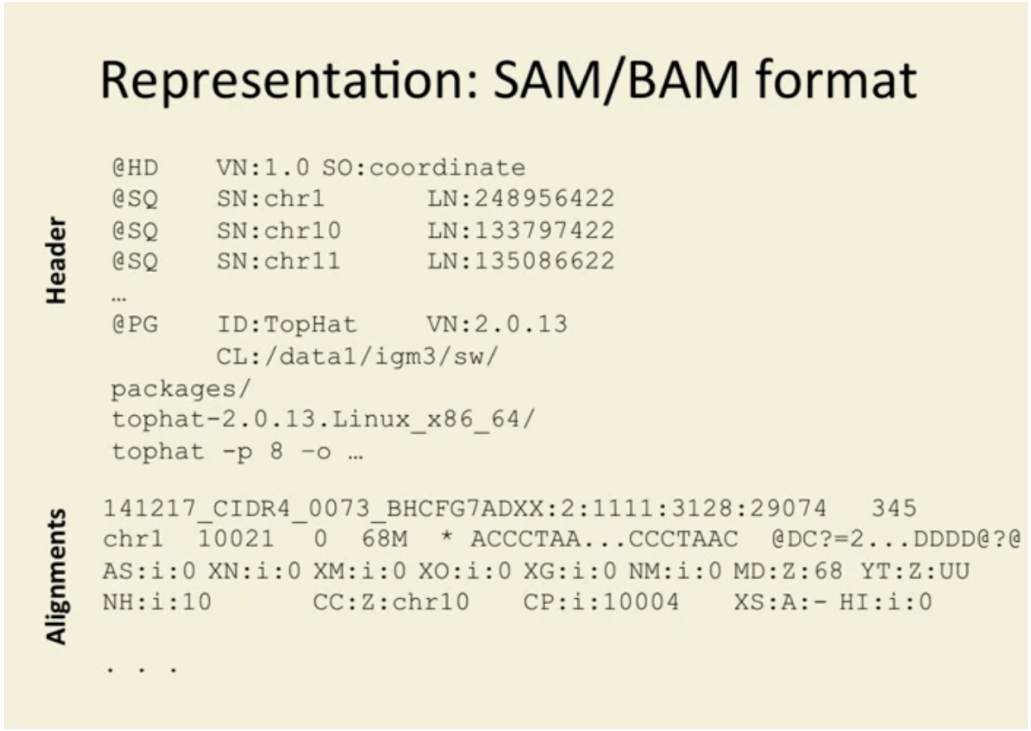


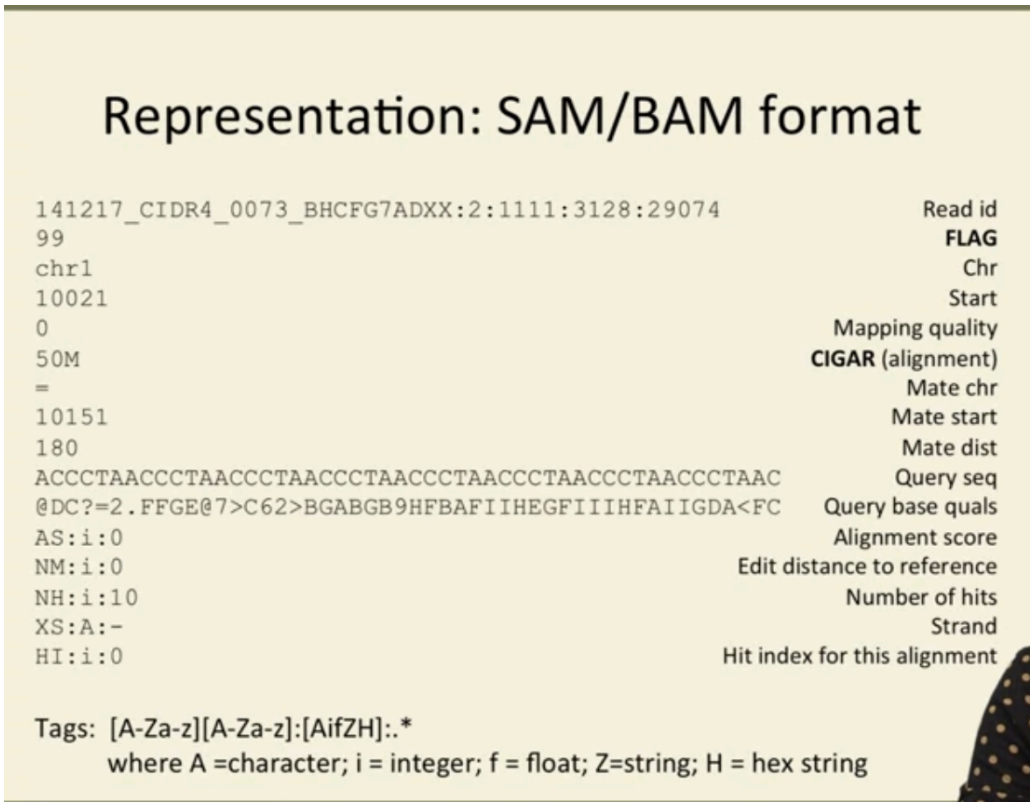
  - Challenge: spliced alignment, aligning parts/reads of mRNA to the parts of DNA that made them
- NGS alignments



  - we'd expect a fixed distance between reads.
  - If the reads point toward each other with expected distance between them, we call that concordant or properly paired. anything else is nonconcordant.
- SAM, BAM are the formats for alignments

- SAM



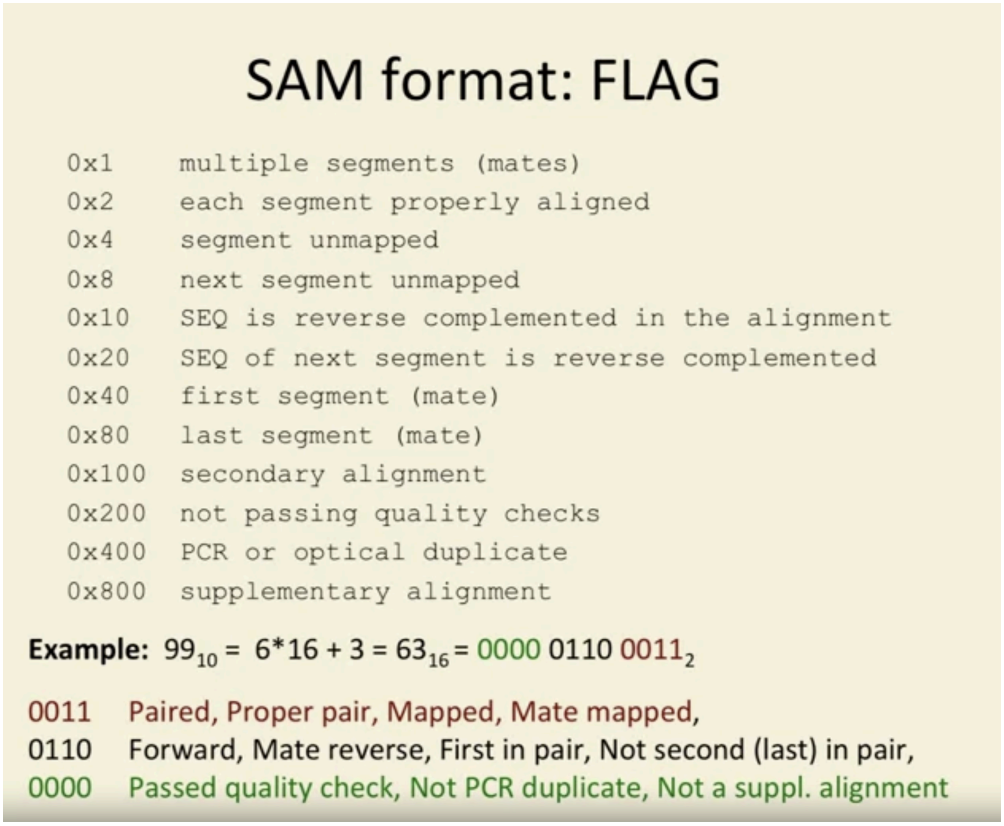


(each line contains all of these fields (some can be specified to be left out too but some art standard))

    - each line starts with a header which tells you which type of line and some info like if it's sorted or not
    - every line after that is @SQ correspond to genomic sequences with their chromosome identifier in the order which they are listed in the genome (chr1, chr10, chr11) and then their length (LN:length)
    - @PG line tells you the program which generated the SAM file. following this is the version and parameters
    - Following all of the header, we have the actual alignments, one line per each alignment
        - read id = fastq header
        - CIGAR (alignment) compressed representation of the alignment
        - Mate chr = what chr is the mate of this gene? "*" not known "=" same chr as Chr or name of other chr
        - Mate start = where would the mate start
        - Mate distance
        - Query seq
        - Query base qualities (phred 33)
        - A bunch of other fields that are either standard or user specified/program specific
            - look at the tags thing in the image

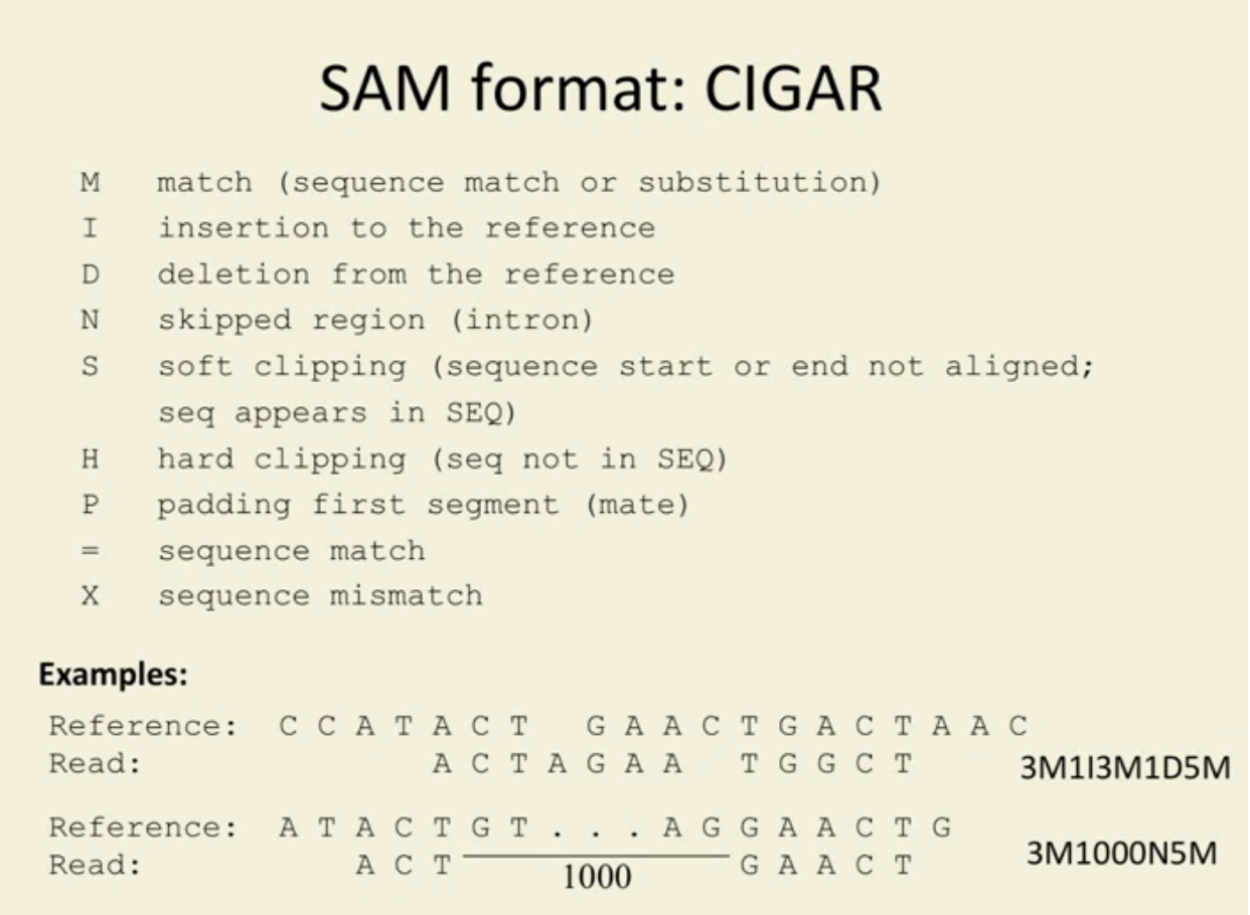
Sequences and Genomic Features 4.2: Alignment II

- wanted to look at FLAG and CIGAR more
- FLAG is a number, it's binary equivalent's digits encode information



- 0x1 rightmost bit tells you if there are multiple segments (paired end reads or not)
- 0x2 tells you if paired end read segments are concordant
- 0x4 segment is mapped or unmapped, aligned
- 0x8 tells you if the next segment (it's mate in the paired end reads) is mapped
- 0x10 and tell you if the current read had to be reverse complemented to get a match, 0x20 tell you if mate was reverse complemented
- 0x40 and 0x80 tell you which mate it is
- 0x100 tells you if its a secondary alignment: A secondary alignment occurs when a given read could align reasonably well to more than one place. One of the possible reported alignments is termed "primary" and the others will be marked as "secondary".
- 0x200 tells you if it's not passing quality checks
- 0x400 PCR or optical sequence
- 0x800 whether it's a supplementary alignment (supplementary alignment is when parts of a read match to different parts of the reference genome

- CIGAR - tells you about the pattern of matches and mismatches



  - it goes through the alignment pattern. If there are 3 matches, it'll write 3M, if there's an insertion it'll write 1I, and on and on through the alignment
  - S - soft clipping means that the start or end of sequence could not be aligned, tells us how many bases couldn't be aligned, but the bases still show in the seq field
  - H - hard clipping same but bases wont show in seq field
  - = and X tell you if M was a match or substitution respectively
  - 3M1I3M1D5M
    - 3M 3 matches or subs
    - 1I insertion
    - 3M 3 matches or subs
    - 1D deletion
    - 5M 5 matches or subs

Downloading Data

- From GenBank, just hit the fasta button and then do it like that
- From SRA, there are command line tools
  - https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&page_size=10&acc=SRR29400501&display=reads

- We have to do it differently from how she did it because the SRA has been updated since the video was made.
- How she did in the video
  - used wget with the ftp link
  - converted the file to fastq with: nohup fastq-dump (file.sra) &
    - nohup- command will be run even if the terminal is closed
    - & means command is being run in the background, output is appended to file nohup.out

Commands:

```
docker pull ncbi/sra-tools # downloads the SRA-tools image.
     Using default tag: latest
     latest: Pulling from ncbi/sra-tools
     bca4290a9639: Pull complete
     fa7067787f1d: Pull complete
     de20feac4765: Pull complete
     243d4c40cb71: Pull complete
     d07ce96ae9b2: Pull complete

     # this string is the image
     Digest:
     sha256:d47e08028655bcc4b41ccd24ac991ec21f8adb435d871a44f610cd3dd9
     c26892
     Status: Downloaded newer image for ncbi/sra-tools:latest

     docker.io/ncbi/sra-tools:latest

mkdir -p ~/sra_data # make the directory on your computer

docker run -it -v ~/sra-data:/data ncbi/sra-tools

     # -v maps ~/sra-data to /data within the container (/data is
     created if it doesn't already exist with this command.) This just
     means what's added to the /data folder in the container is added
     to ~/sra-data

/ # prefetch --progress data/SRR1107997 # downloads the files
```

```
/ # fastq-dump /data/SRR110799/SRR1107997.sra # dumps a fastq in the SRR110799 folder in data
```

Sequences and Genomic Features 6: Genomic Feature Retrieval

UCSC Table Browser for Genomic Features

- go to the table browser
- enter all the information (can get reads, genes and genomes)
- get output
- save file from there

Sequences and Genomic Features 7: SAMtools I

- have to use a docker container: docker run -it --name gencommand_image 6309146af95b

```
samtools

flagstat
     (base) [root@fa83044c5d9c /]# samtools flagstat example.bam

     # total reads: (QC-passed reads + QC-failed reads)
     2912771 + 0 in total (QC-passed reads + QC-failed reads)

     # secondary alignments: (QC-passed + QC failed secondary alignments).
     # secondary alignments - when a read maps to multiple locations in the reference genome
     0 + 0 secondary

     # Supplementary alignments are used in cases like split-read alignments, where a single read maps to multiple locations. (QC-passed, QC-failed)
     0 + 0 supplementary

     # Duplicates are multiple reads that appear to be copies of the same original DNA fragment, often arising during PCR amplification. Basically duplicate reads.
     0 + 0 duplicates

     # % of reads successfully aligned (passed QC, failed QC)
```

```
      0 + 0 mapped (0.00%:nan%)

      # paired end reads
      2883950 + 0 paired in sequencing

      # number of read1
      1441975 + 0 read1
      # number of read2
      1441975 + 0 read2

      # concordant paired reads
      0 + 0 properly paired (0.00%:nan%)

      # number of reads where both the read and its mate are mapped.
      0 + 0 with itself and mate mapped

      # number of reads that are mapped while their mate is not.
      0 + 0 singletons (0.00%:nan%)

      # number of reads where the mate is mapped to a different
      chromosome, where each has a different quality threshold.
      0 + 0 with mate mapped to a different chr
      0 + 0 with mate mapped to a different chr (mapQ>=5)

sort
      samtools sort example.bam example.sorted # sorts a file
      # fairly expensive process so hide with nohup and &
index
      samtools index (sorted file) # creates a .bai file
merge
      # merges multiple alignment files, simple concatenation
      merge (bam file1) (bam file 2)
zcat
      # view file without opening zip
      zcat ().fastq.gz | wc -l # number of lines in fastq
view
      Usage:   samtools view [options] <in.bam>|<in.sam>|<in.cram> [region ...]

      Options: -b       output BAM
```

```
              -C       output CRAM (requires -T)
              -1       use fast BAM compression (implies -b)
              -u       uncompressed BAM output (implies -b)
              -h       include header in SAM output
              -H       print SAM header only (no alignments)
              -c       print only the count of matching records
              -o FILE  output file name [stdout]
              -U FILE  output reads not selected by filters to FILE [null]
              -t FILE  FILE listing reference names and lengths (see long help)
      [null]
              -T FILE  reference sequence FASTA FILE [null]
              -L FILE  only include reads overlapping this BED FILE [null]
              -r STR   only include reads in read group STR [null]
              -R FILE  only include reads with read group listed in FILE [null]
              -q INT   only include reads with mapping quality >= INT [0]
              -l STR   only include reads in library STR [null]
              -m INT   only include reads with number of CIGAR operations
                       consuming query sequence >= INT [0]
              -f INT   only include reads with all bits set in INT set in FLAG [0]
              -F INT   only include reads with none of the bits set in INT
                       set in FLAG [0]
              -x STR   read tag to strip (repeatable) [null]
              -B       collapse the backward CIGAR operation
              -s FLOAT integer part sets seed of random number generator [0];
                       rest sets fraction of templates to subsample [no subsampling]
              -@ INT   number of BAM compression threads [0]
              -?       print long help, including note about region specification
              -S       ignored (input format is auto-detected)
```

```
convert SAM to BAM
      samtools view -bT (reference file fasta) (sam file) > (bam file)
extract alignments within a range
      # want to look at alignments on chr() between pos 59373560-
      59373566
      samtools view example.sorted.bam “chr():59373560-59373566”
      # file must be indexed.
```

BEDTools

- Inter Biological questions:
    - Give me all exons or genes or transcript that contain the Alus (so basically find the intersecting intervals of positions between Alus and other features)
    - Give all the alignments that overlap Alus

- answer first question: find the intersecting intervals of position between Alus.bed and RefSeq.bed or RefSeq.gtf

    - -wo will write A and B file entries and the number of bases that overlap between them.
    - How many exons exons (transcripts/genes) in the reference annotation contain Alus: bedtools intersect -wo -a RefSeq.gtf -b Alus.bed | more
    - Output

BEDtools

- Inter-Biological questions:
    - Give me all exons or genes or transcripts that contain Alus
    - Give me all expressed Alus
- How many exons exons (transcripts/genes) in the reference annotation contain Alus
- When we talk about "overlapping regions" in the context of bedtools intersect, we are referring to the genomic coordinates that overlap between the features in the two files (GTF and BED) based on their start and end positions.
- bedtools intersect -wo -a RefSeq.gtf -b Alus.bed | more

```
(base) [root@12018efd4a72 bedtools_demo]# bedtools intersect -wo -a RefSeq.gtf -b Alus.bed | more
chr22   hg38_refGene    exon    11955271        11956534        0.000000        +       .       gene_id "NR_110761"; transcript_id "NR_110761"; chr22   11955470        11955779        AluSp   2273    +       309
chr22   hg38_refGene    exon    15791010        15791152        0.000000        +       .       gene_id "NR_122113"; transcript_id "NR_122113"; chr22   15790994        15791287        AluY    2357    -       143
chr22   hg38_refGene    exon    15826142        15827434        0.000000        +       .       gene_id "NR_122113"; transcript_id "NR_122113"; chr22   15827312        15827445        AluJo   904     +       122
chr22   hg38_refGene    exon    16648527        16648830        0.000000        +       .       gene_id "NR_001591"; transcript_id "NR_001591"; chr22   16648504        16648805        AluSz   2156    -       279
```

- first block of cols is the RefSeq entry, highlighted block is the Alus.bed entry, ending number 309 is the number of bases that overlap between them
- Subquestion: How many genes do these correspond to?

```
bedtools intersect -wo -a RefSeq.gtf -b Alus.bed | more | cut -f9 |
cut -d' ' -f2 | sort | uniq -c | wc -l
```

259

- cut gene id column, cut gene id, sort them, compress unique lines, count lines.

There's another way to do it: use the RefSeq.bed.

What this means…

(base) [root@12018efd4a72 bedtools_demo]# more RefSeq.bed | wc -l
1147
(base) [root@12018efd4a72 bedtools_demo]# more RefSeq.bed | cut -f4 | sort | uniq -c | wc -l
1136

(base) [root@12018efd4a72 bedtools_demo]# more RefSeq.gtf | wc -l
20732
(base) [root@12018efd4a72 bedtools_demo]# more RefSeq.gtf | cut -f9 | cut -d' ' -f2 | sort | uniq -c | wc -l
1136

is that even though both have the same number of unique genes (1136), the gtf file has way more features for each gene. This is because each exon gets its own row in the gtf file whereas in the bed file, multiple are grouped together under one gene.

- Number of Unique Genes (1136): This indicates that there are 1136 unique gene IDs. Since there are 1147 lines but only 1136 unique genes, some genes are represented by more than one line, but overall, the BED file has fewer features per gene.
- Number of Lines (20732): This indicates there are 20732 entries in the GTF file. Number of Unique Genes (1136): This indicates that there are 1136 unique gene IDs, the same as in the BED file. However, the significantly larger number of lines compared to the BED file suggests that each gene is represented by many more individual features (e.g., exons, CDS).

Essentially, the bed format is one line per gene for the most part

- more chance for overlap with gtf than for bed

(base) [root@12018efd4a72 bedtools_demo]# bedtools intersect -split -wo -a RefSeq.bed -b Alus.bed | cut -f4 | sort | uniq -c | wc -l
259

Sequences and Genomic Features 10: BEDtools II

- Converting from BAM to BED format

```
bedtools bamtobed -split -i NA12814.bam | grep ERR188081.370400

bedtools bamtobed -split -i NA12814.bam > NA12814.bed
bedtools bamtobed -cigar -i NA12814.bam > NA12814.cigar.bed

Difference between these two files?
The first contains less entries because the exons weren't split into
each individual line.

(base) [root@12018efd4a72 bedtools_demo]# more NA12814.cigar.bed |
grep ERR188081.370400
chr1  14806 15021 ERR188081.370400 0     +     23M140N52M
chr1  14806 185542     ERR188081.370400 0     +     23M170661N52M
chr1  15028 186381     ERR188081.370400 0     -     10M171278N65M
chr1  185327     185542     ERR188081.370400 0     +     23M140N52M
chr1  185549     186381     ERR188081.370400 0     -     10M757N65M
chr12 15143 15977 ERR188081.370400 0     -     10M759N65M
chr15 101975940  101976155  ERR188081.370400 0     -     52M140N23M
chr16 14488 14703 ERR188081.370400 0     +     23M140N52M
```

```
chr2  113597573  113598407  ERR188081.370400 0    +    65M759N10M
chr2  113598414  113598629  ERR188081.370400 0    -    52M140N23M
chr9  14917 15132 ERR188081.370400 0    +    23M140N52M
chrX  156024207  156025041  ERR188081.370400 0    +    65M759N10M
chrX  156025048  156025263  ERR188081.370400 0    -    52M140N23M
chrY  57210727   57211561   ERR188081.370400 0    +    65M759N10M
chrY  57211568   57211783   ERR188081.370400 0    -    52M140N23M

(base) [root@12018efd4a72 bedtools_demo]# more NA12814.bed | grep
ERR188081.370400
chr1  14806 14829 ERR188081.370400 0    +
chr1  14969 15021 ERR188081.370400 0    +
chr1  14806 14829 ERR188081.370400 0    +
chr1  185490     185542     ERR188081.370400 0    +
chr1  15028 15038 ERR188081.370400 0    -
chr1  186316     186381     ERR188081.370400 0    -
chr1  185327     185350     ERR188081.370400 0    +
chr1  185490     185542     ERR188081.370400 0    +
chr1  185549     185559     ERR188081.370400 0    -
chr1  186316     186381     ERR188081.370400 0    -
chr12 15143 15153 ERR188081.370400 0    -
chr12 15912 15977 ERR188081.370400 0    -
chr15 101975940  101975992  ERR188081.370400 0    -
chr15 101976132  101976155  ERR188081.370400 0    -
chr16 14488 14511 ERR188081.370400 0    +
chr16 14651 14703 ERR188081.370400 0    +
chr2  113597573  113597638  ERR188081.370400 0    +
chr2  113598397  113598407  ERR188081.370400 0    +
chr2  113598414  113598466  ERR188081.370400 0    -
chr2  113598606  113598629  ERR188081.370400 0    -
chr9  14917 14940 ERR188081.370400 0    +
chr9  15080 15132 ERR188081.370400 0    +
chrX  156024207  156024272  ERR188081.370400 0    +
chrX  156025031  156025041  ERR188081.370400 0    +
chrX  156025048  156025100  ERR188081.370400 0    -
chrX  156025240  156025263  ERR188081.370400 0    -
chrY  57210727   57210792   ERR188081.370400 0    +
chrY  57211551   57211561   ERR188081.370400 0    +
```

```
chrY  57211568   57211620   ERR188081.370400 0    -
chrY  57211760   57211783   ERR188081.370400 0    -
```

```
bedtools bedtobam -i RefSeq.gtf -g hg38c.hdrs > Refseq.bam
```

bed to bam file

```
bedtools bedtobam -i RefSeq.bed -g hg38c.hdrs > refseq.bam
```

NR_110761

59129M

- When converting a standard BED file to BAM, bedtools bedtobam treats each feature as a single continuous region.
- Example: If the feature NR_110761 spans from 1000 to 60129 with no introns specified, the BAM output will show a single continuous match (M), e.g., 59129M, indicating that the feature spans 59129 bases continuously.

```
bedtools bedtobam -bed12 -i RefSeq.bed -g hg38c.hdrs >
refseq.bed12.bam # shows a more realistic view if the input is in
bed12 format
```

NR_110761

62M12698N54M45051N1264M

- Shows all the exons and introns
- When converting a BED12 file, bedtools bedtobam takes into account the block structure (exons and introns).
- Example: If NR_110761 has exons and introns specified in the BED12 format, the BAM output will reflect this structure with matches (M) and skipped regions (N, representing introns).
- The output 62M12698N54M45051N1264M indicates:
    - 62M: 62 bases of exon.
    - 12698N: 12698 bases of intron (skipped).
    - 54M: 54 bases of exon.
    - 45051N: 45051 bases of intron (skipped).
    - 1264M: 1264 bases of exon.

Get FASTA files

```
bedtools getfasta -fi hg38c.fa -bed RefSeq.gtf -fo RefSeq.gtf.fasta # one line for every feature in the genome
```

```
bedtools getfasta -fi hg38c.fa -bed RefSeq.bed -fo RefSeq.gtf.fasta # one line for every feature in the genome
```

* we did not specify that each line contains of blocks, all the exons and introns were put together and each gene is one line

```
bedtools getfasta -fi hg38c.fa -split -bed RefSeq.bed -fo RefSeq.gtf.fasta # one line for every feature in the genome
```

- split specifies that we have exon,intron blocks for each entry

- `output: the exons for each gene have been concatenated together for each gene.`

Alignment & Sequence Variation 1: Overview

- sequence variation
- workflow for determining sequence variation

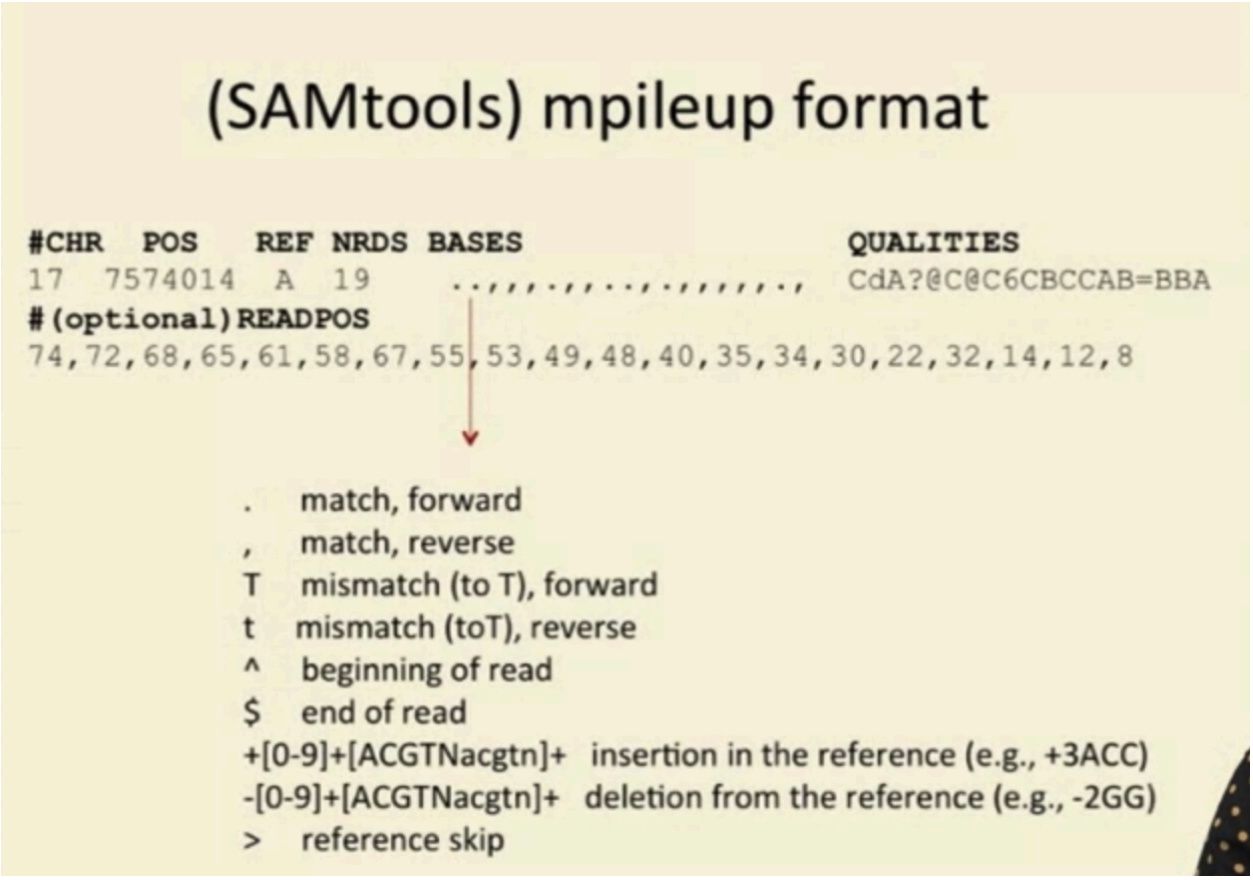
- sequence variation - minor differences in sequences between all people.
    - substitutions
    - indels
    - one base, small blocks up to 1000bp
- 0.1% difference between any two individuals ⇒ sounds small but that's a lot of bases considering the size of the genome
- projects aim to capture sequence variation from the reference genome
- polymorphisms - a variant occuring in 1% of people

How to identify sequence variations in an individual

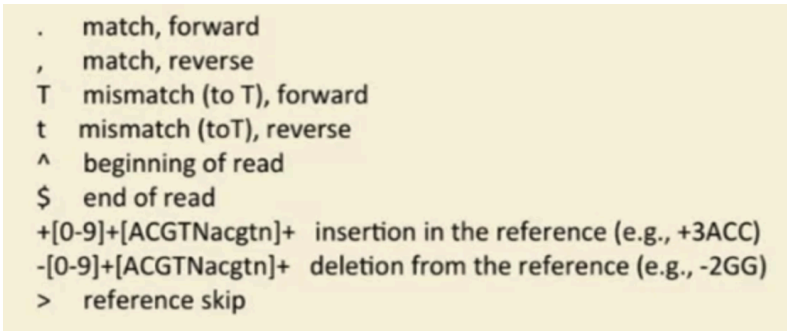
- take reads, align them, analyze the assembled genome.
- We expect disease-causing variants are within the protein coding parts of the genes.
    - whole exome shotgun sequencing: only sequence DNA within portions of the genome that produce genes, coding material
- We'll start with alignment, then we do variant calling.

Alignment & Sequence Variation 2: Alignment & Variant Detection Tools

- SAM/BAM ⇒ standard formats for alignments
    - Header
        - information for the file, sequence lines that show us the sequences present, and then the program and command line options used to create the alignment.
    - Each line represents the alignment of one read to the reference genome.
        - Contains all these fields
- New format: SAMtools mpileup, used to represent variants



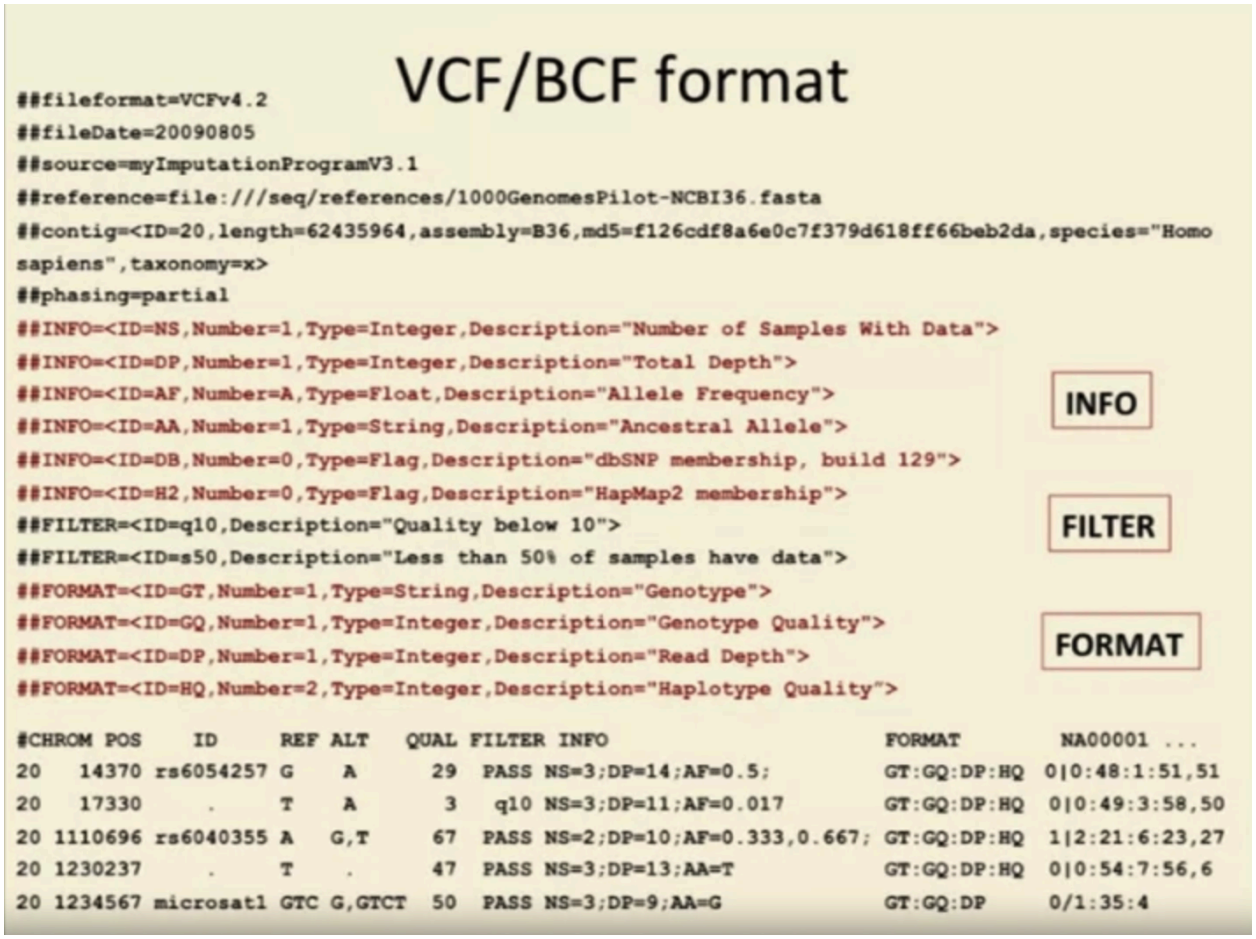
- each line = one position in the genome, the columns provide information about the alignment at that position.
    - chromosome
    - position (1-based)
    - letter
    - depth/coverage of reads at that pos: in this case, 19 reads at that position.
    - BASES string, tells you about each of the reads (coverage) at that position



        -
        - the beginning of the read is usually lower quality than the rest of the read.
    - Base qualities
    - positions of that letter in each read.

Alignment & Sequence Variation 3: VCF

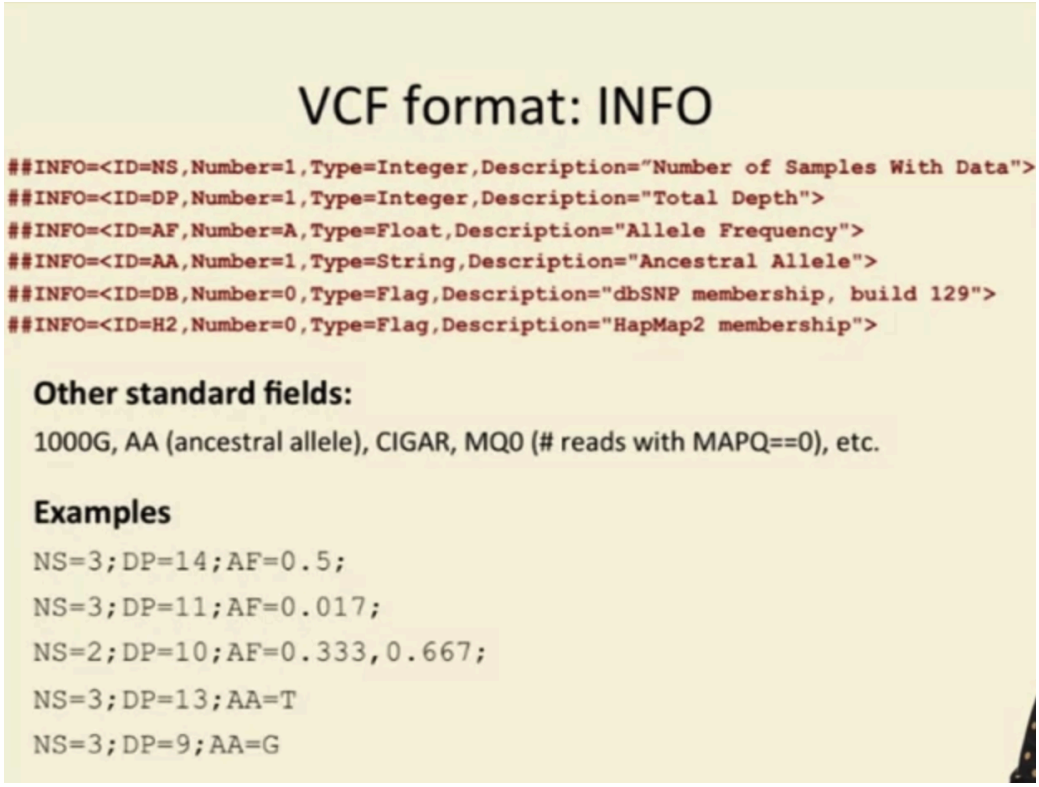
- The VCF format describes sequence variation
- can be compressed to BCF
- detailed information on only those positions where a variation was observed.



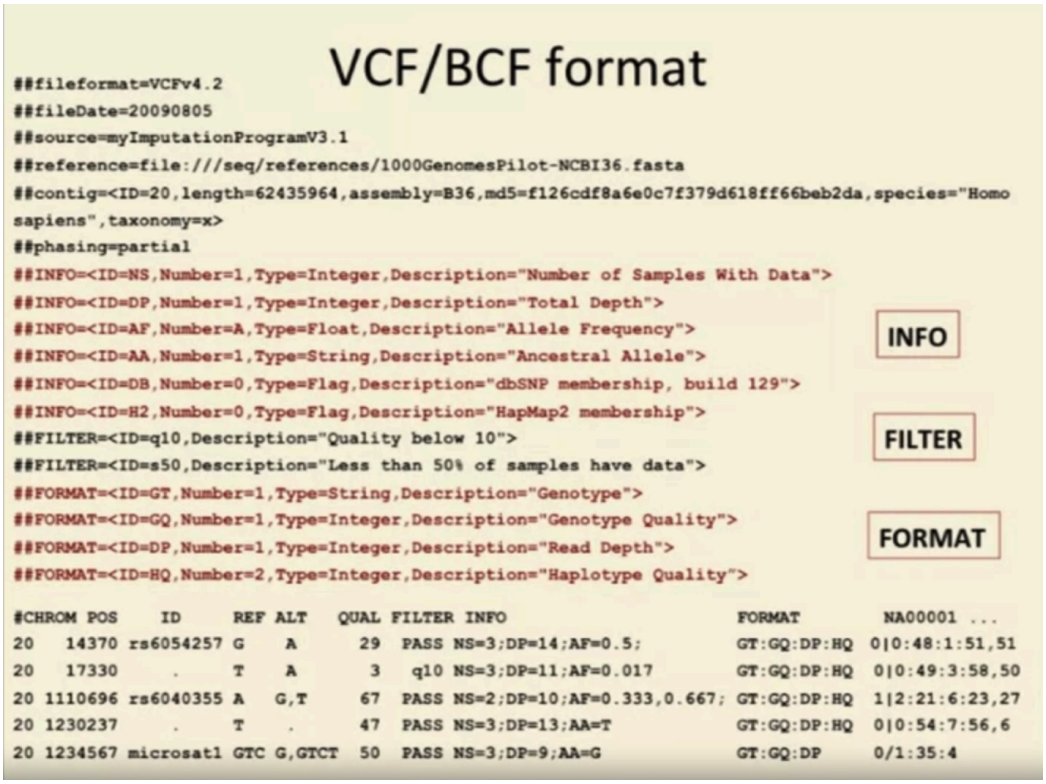
- preamble (top lines ##): general information.
- Info portion (INFO)
- FILTER portion
- FORMAT lines
- The actual file.

- chromosome
- genome position (1 based)
- ID if if the variant is in another database
- Reference
- Alternate
- Variant quality
- If it passed the “filter” from FILTER lines

- “info” defined in INFO lines



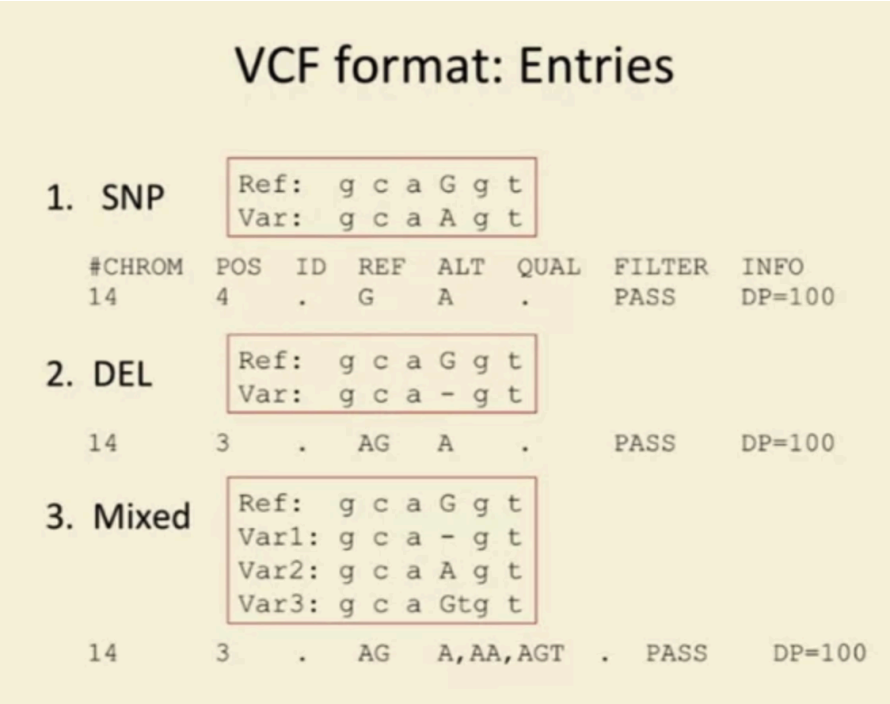
    - NS: Integer, Number of samples with data
    - DP: Integer, Total Depth
- format: How the genome information is represented

- 

  - ### Understanding `GT = 0|0`

    - **`GT` (Genotype)**: This field specifies the alleles of a sample at the variant position.
    - **`0`**: Refers to the reference allele.
    - **`|`**: Indicates that the alleles are phased, meaning the sequencing can determine which alleles are on the same chromosome. (A `/` symbol would indicate unphased alleles, where the phase is unknown.)
    - **`0|0`**: Means that both alleles at this position are the reference allele. This indicates a homozygous reference genotype.

-
- Haplotype: https://www.youtube.com/watch?v=_5imxW1CB2M
  - if we have two haplotype scores, means that the first is the confidence that it belongs to one parent and the second is confidence that it belongs to second parent.



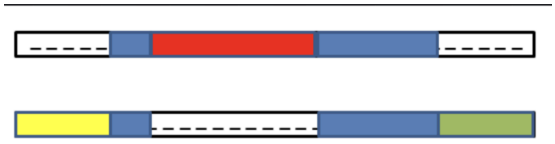
-
    - Mixed = three different reads (maybe from different genomes) that all have different mismatches, in which case, you should have comma separated alternate column

Alignment & Sequence Variation 4: Bowtie

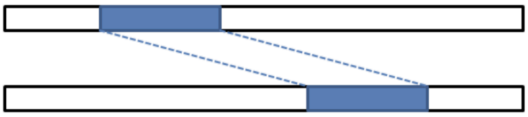
Alignment tools
- bowtie
- BWA
- use an index/hash map to quickly map a read to sequences in the index.
- both of these programs are looking for close matches with only small # of indels / subs

Bowtie
- could not find the files she was working with, will just watch then
- usually bowtie index can be downloaded for whole organism

```
bowtie2-build # builds an index
bowtie2-build HPV_all.fasta hpv/hpv # makes files with prefix hpv,
these collectively represent the genome index.
# once you have the index, you need to perform the alignment.
bowtie2 >& bowtie2.log # redirects error stream to log.
bowtie2 # gives you all the options
# can specify if you want global or local alignments
```



Global Alignment



Local Alignment

```
bowtie2 -p 4 -x (genome) (reads) # aligns reads to genome -S
(name.sam) # creates the alignment SAM file. get a summary.

# convert the sam file to bam
samtools view -bT (fasta file of genome) (SAM file) > (BAM file)

bowtie2 --local -p 4 -x (genome) (reads) # performs local alignments.

BWA

# create the index
bwa index (complete genome fasta) # creates the index

# map the reads
bwa mem # gives you the options
bwa mem -t (complete genome fasta) (reads) > (sam format) # sam format
of alignments
samtools view -bT (complete genome fasta) (sam alignment) > (bam
alignment) # bam format of alignments.

Variant Calling:
```

SAMtools (mpileup option)

- different than samtools
- creates a tally of information for each base in the genome at that positon.
- creates a “genotype likelihood” that is used by further tools to do variant calling

samtools mpileup [options] in1.bam [in2.bam…]

samtools flagstat sample.bam # the amount of alignments contained.
samtools mpileup -f (reference genome) (sorted and indexed bamfile) > (.mpileup file)

- mpileup file will have the chrom, position, reference letter, number of reads, letter string, qualities.

samtools mpileup -v -u -f (reference genome) (sorted and indexed bamfile) > (.vcf file)
-v - u says uncompressed v and u

samtools mpileup -g -u -f (reference genome) (sorted and indexed bamfile) > (.bcf file)
compressed format of vcf.

Make variant calls with bcftools.

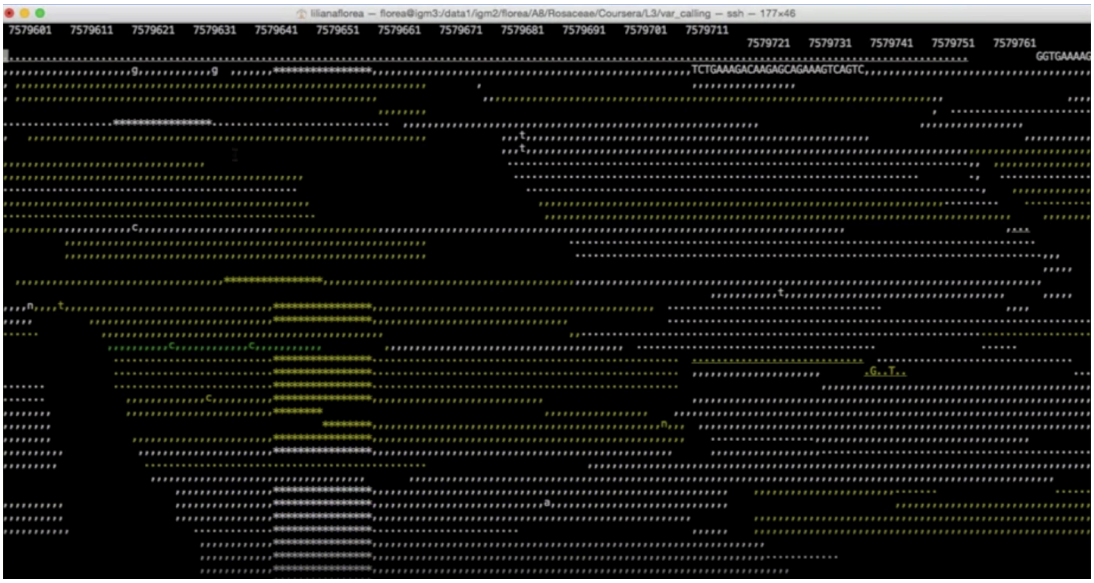
- looking at the view and call commands

bcftools view # look at the type of output options
bcftools view (.bcf file in binary)

bcftools call # look at the options for variant calling
bcftools call -v -m -O z -o (output into sample.vcf.gz) (input file sample.bcf) # gives a vcf file
-v: only write lines corresponding to variants
-m: mutliallelic something
-O z: specify format, vcf compress
-o: output directory

Alignment & Sequence Variation 8: Variant Calling

- put it all together, do some variant calling
- take a bam file
- index the bam file
- create a bcf file (samtools mpileup -g -u -f (ref) (sorted, indexed bam file) > (.bcf file)
- bcftools call -v - m -O z -o (output file sample.vcf.gz) (.bcf file)

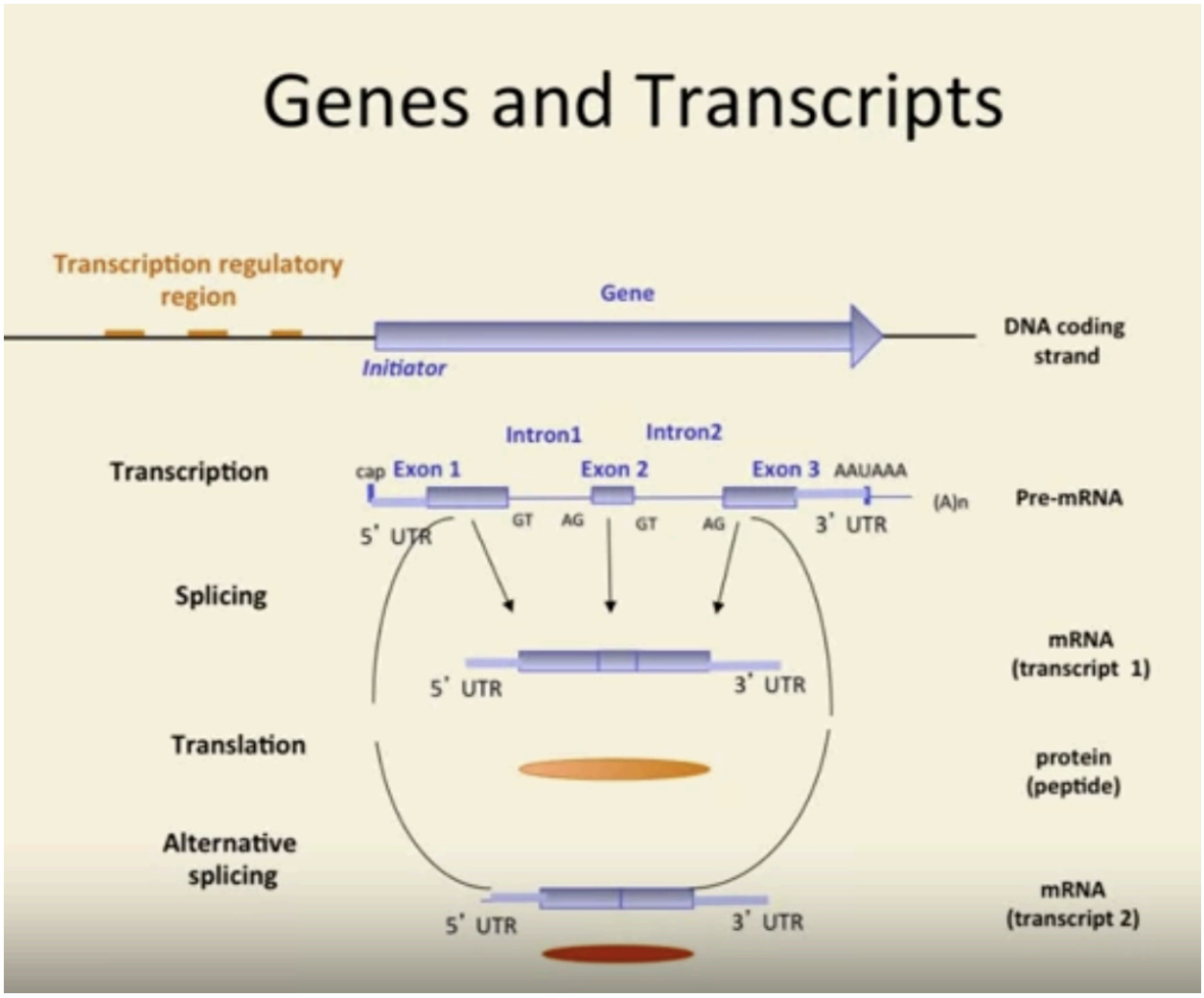
- end up with the sample.vcf.gz
- one can do field tests or visually inspect quality/# of alignments at a position.
- samtools tview -p chr:pos -d T (alignment sample.bam) (fasta sequence of genome) # -d and T specify simple text format output.
  tview text alignment viewer
  - 
    - gives you this monstrosity,what is this: text-based alignment

In a text-based alignment viewer like samtools tview, the display primarily consists of characters and symbols that represent the alignment of sequences from a BAM file (containing aligned reads) against a reference genome sequence (from a FASTA file). Here's a basic guide to what you typically see:

1. **Reference Sequence Line**: Usually displayed at the top or left side, this line shows the reference genome sequence. Each character represents a nucleotide (A, T, C, G) or gaps (dashes -) where there is no sequence in the reference.
2. **Alignment Rows**: Below or alongside the reference sequence, each row represents an aligned read from the BAM file. Each read is aligned against the reference sequence, and characters or symbols in these rows indicate:
   - . or ,: Match or mismatch with the reference (depending on the quality of the alignment).

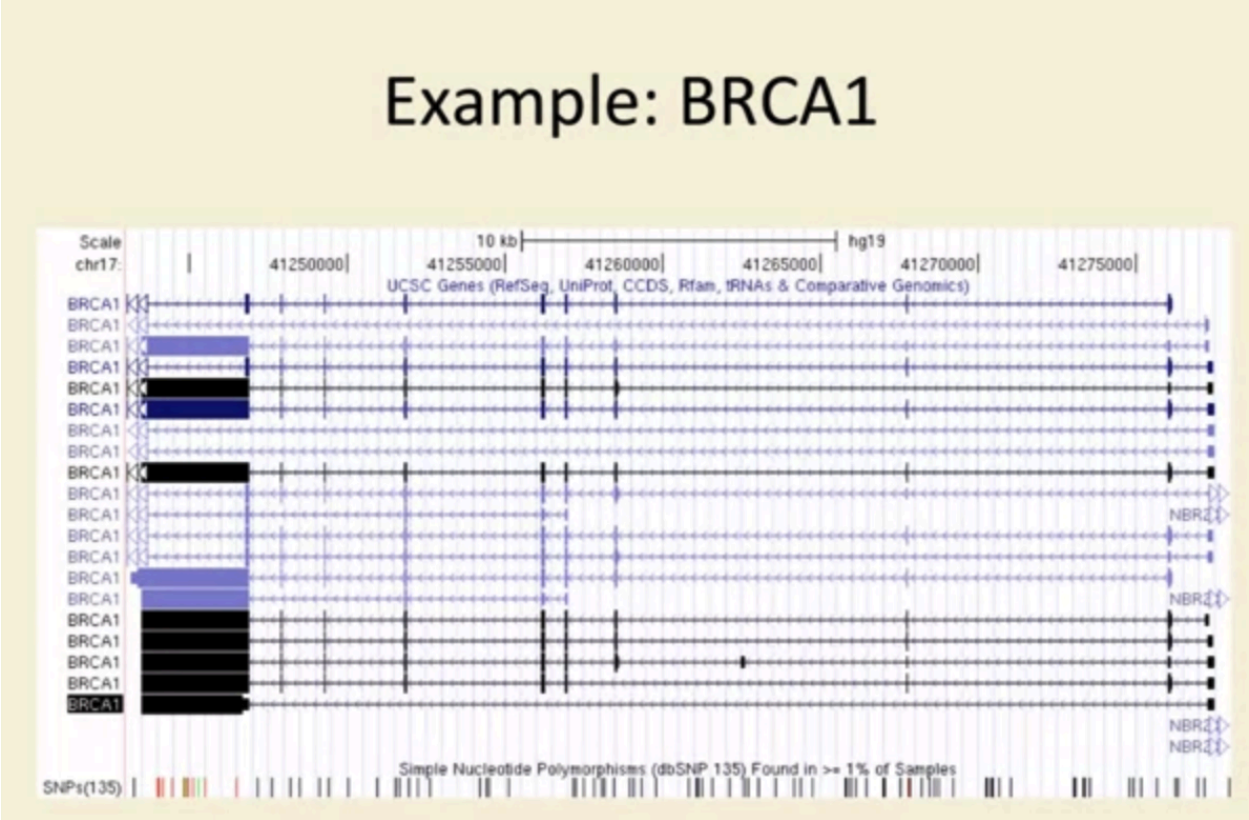
    - A, T, C, G: Specific bases that differ from the reference (depending on the quality of the alignment).
    - ^: Start of a read segment not aligned to the reference.
    - $: End of a read segment not aligned to the reference.
    - *: Indicates a deletion relative to the reference.
    - +: Indicates an insertion relative to the reference.
3. **Quality Indicators**: Numeric scores or symbols that represent the quality of the alignment, such as alignment scores, base quality scores, and mapping quality scores.
4. **Navigation**: Typically, you can navigate through the alignment using arrow keys or specific commands to move along the reference sequence or between reads.
5. **Color Coding**: Some viewers may use color to highlight different types of mismatches or variations from the reference sequence, aiding in visual interpretation.

Tools for Transcriptomics 1: Overview

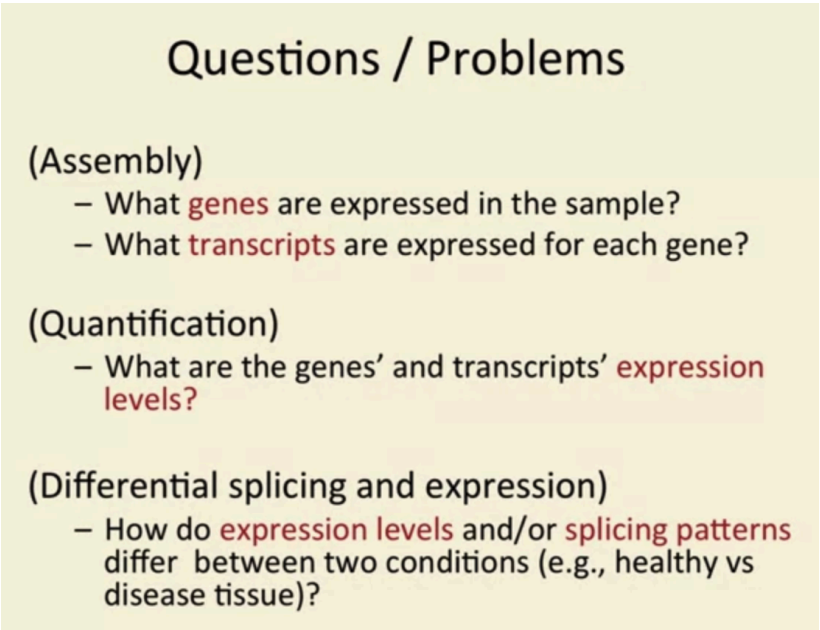


* The process of transcription: mRNA is made after splicing the exons out of the stabilized pre-mRNA.
* Alternative splicing: different mRNA transcripts produced from the same transcription ⇒ making multiple isoforms.

UCSC Browser:



* These are all different transcripts of BRCA1, vertical thick parts are exons, the —------ are introns. Every transcript shows a different combo of exons.
* 90 percent of human genes have 2+ transcripts.

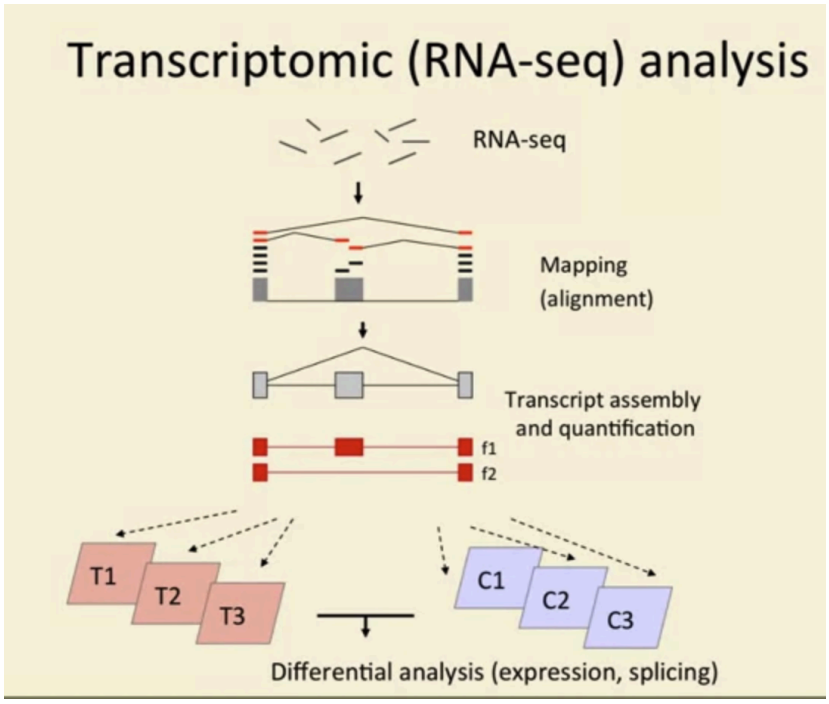


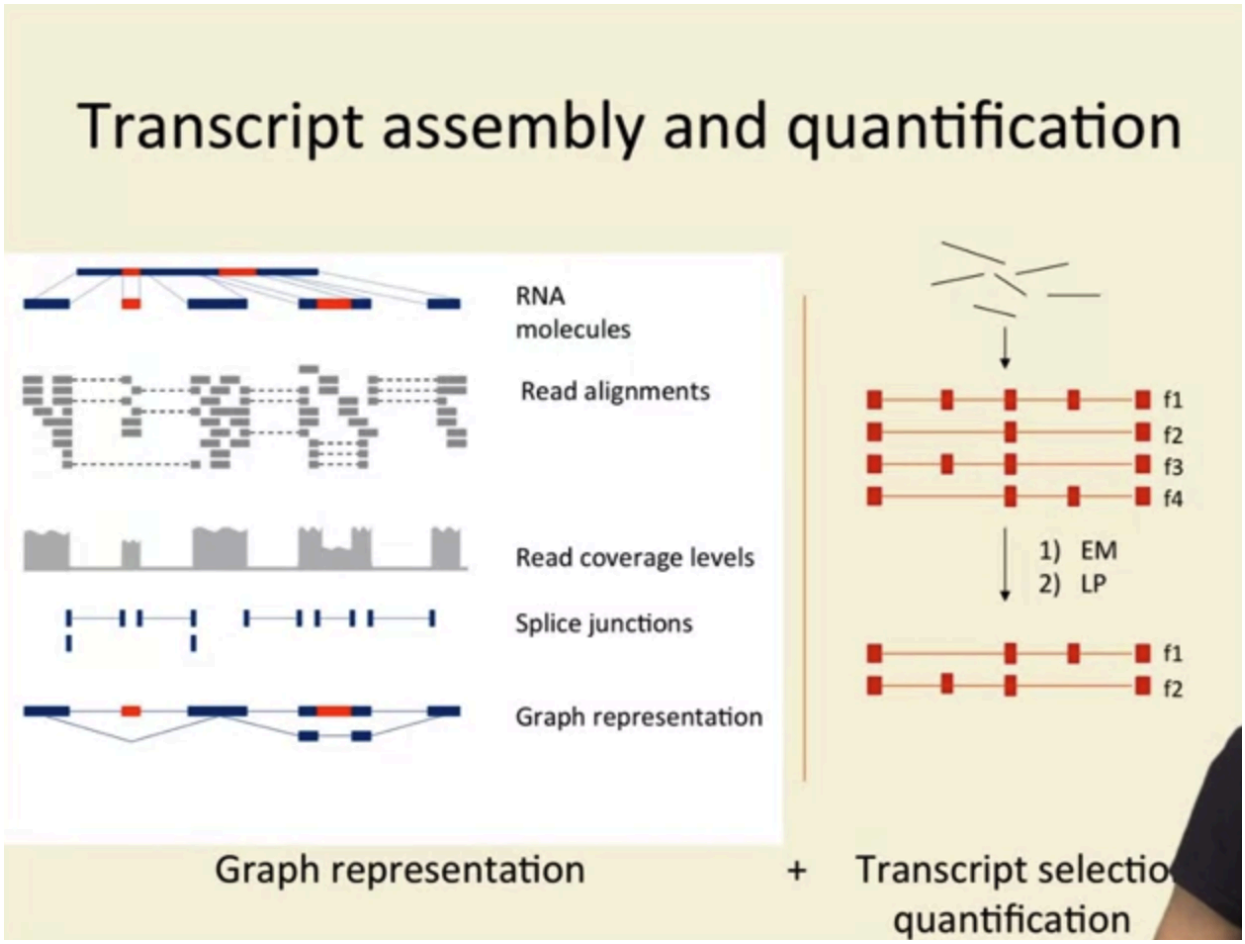
Answers: transcriptomic analysis.

Tools for Transcriptomics 2: RNA-seq

- Answering those 3 categories of questions

Workflow for RNA-seq analysis





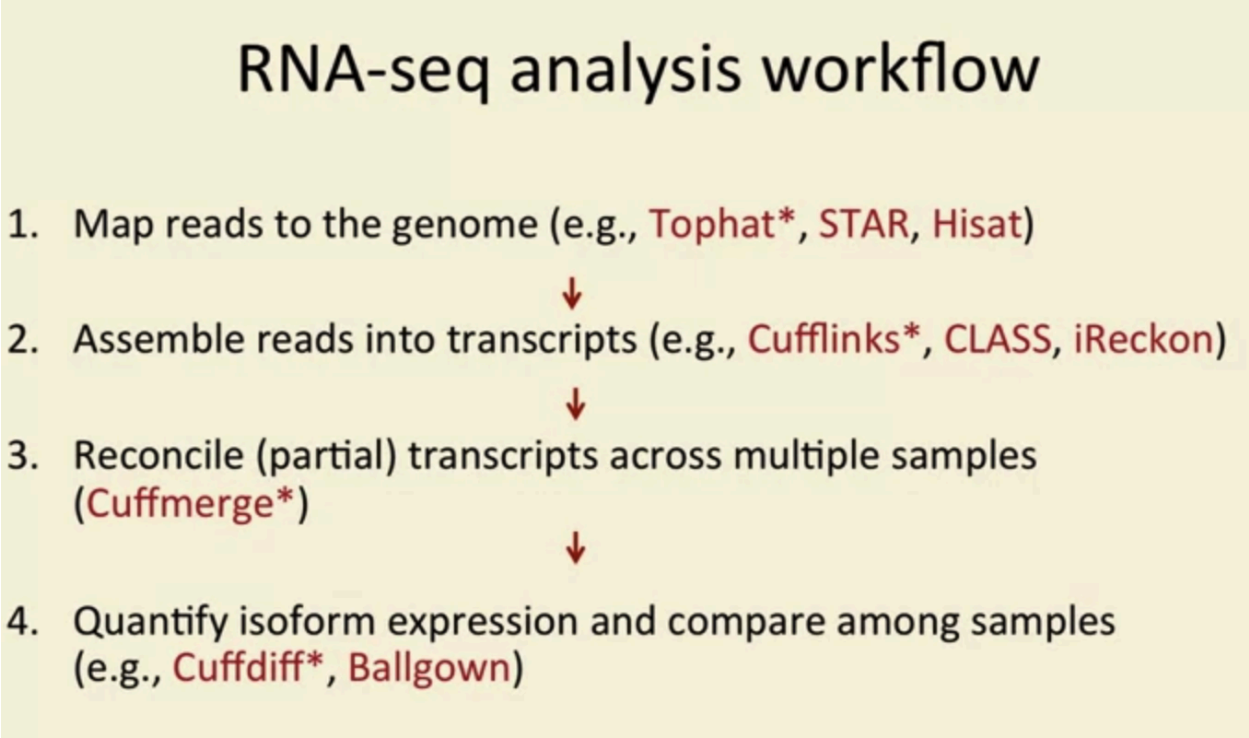
Graph Representation

- Traverse through the exon graph to get every possible transcript.
- Problem: the number of possible transcripts is not usually the number of transcripts that actually exist, so how do you know which exactly exist

Transcript selection qualification:

- Expectation maximization or Linear program to assign each read to the possible transcripts, select out the transcripts that occur.

Entire workflow



* 3 is necessary because there might not be enough reads from one sample.

Tools for Transcriptomics 3.1: Tophat I

Most of the options defaults are calibrated to best performance, leave as is.

Create a bash script:

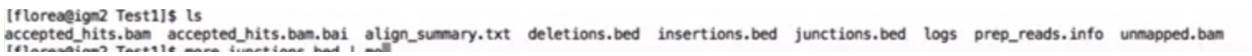
```
mkdir -p (output directory)
tophat -o (output directory) -p 10 --max-multihits=10 (40 by default, maximum number of hits) -G (annotation file gtf/gff)
--transcriptome-index (bowtie produced index .bt2 files) -r 120
(distance between mates) --mate_std-dev 30
(bowtie index)
(reads, probably fastq)
```

-r and --mate_std-dev not generally needed.

- in a bash file, you can assign variables and use them in commands
- take advantage, assign variables to each component and then plug them in like ANNOT or ANNOTIDX

- to run a script: nohup bash com.tophat >& com.tophat.log

Results of running tophat:



- junctions.bed contains the splice junctions: Splice junctions are the specific sites on a pre-mRNA transcript where splicing occurs, meaning they are the locations where the cutting and joining of RNA takes place. They are not genes themselves but rather specific sequences within the pre-mRNA that signal where introns should be removed and exons should be joined together during the splicing process.
- align_summary.txt: gives us basic information on the alignment.
    - concordant alignment rate.
    - can grep for overall read mapping rate

Bowtie just mapped transcripts to the genome, now we want to assemble transcripts.

- Cufflinks creates that graph representation and then walks through it to create all possible transcripts.

cufflinks [options] <hits.sam>
--label: gives assembled transcripts/genes a prefix, allows us to distinguish between this sample and other samples.
--min-isoform-fraction: suppress transcripts below this abundance level, dont report them.

```
mkdir Cufflinks

create a bashfile, com.cufflinks:

THDIR=(wherever your tophat results are.)
WORKDIR=(cufflinks directory)

mkdir -p WORKDIR/Test1
cd WORKDIR/Test1 cufflinks --label BJ-A022 -p 8
THDIR/accepted_hits.bam # creates set of file in cufflinks directory.

nohup bash com.cufflinks >& com.cufflinks.log # takes up to 1 day
sometimes.
```


```
* transcripts.gtf: assembled transcripts
* skipped.gtf: small number of transcripts that could not be
processed.
* .fpkm_tracking files give you estimates of the level of expression
```

Tools for Transcriptomics 5: Cuffdiff

```
cuffmerge - reconcile gene structure across samples and compare to a
reference genome.

touch (cufflinks files .txt) # in here, write transcripts.gtf for each
folder.

ANNOT=(path to annotation)
cuffmerge -g ANNOT -p 8 -o (output directory) (txt file that lists the
transcript files created by cufflinks) # calls cufflinks to assemble
these with the reference genome.

# merged.gtf will have new gene id and transcript id and gene_name,
old name as well.
```

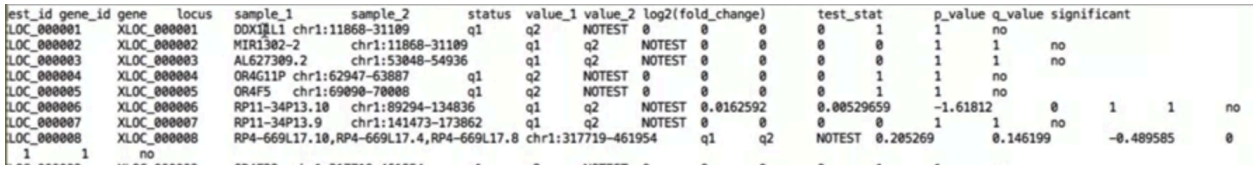
cuffdiff first assigns reads to each of the samples transcripts. Then it analyzes everything to see which transcripts are present.

THDIR=(directory to tophat results)
cuffdiff -o (output directory) -p 10 (merged.gtf) (List all accepted_hits.bam files for all samples here, control and test.)

What output looks like:

```
cds.diff  cds_exp.diff  gene_exp.diff  isoform_exp.diff  promoters.diff  splicing.diff  tss_group_exp.diff
```

- gene_exp.diff and isoform_exp.diff show if the genes are significantly different
- gene_exp.diff: expression level analysis.



test_id	gene_id	gene	locus	sample_1	sample_2	status	value_1	value_2	log2(fold_change)	test_stat	p_value	q_value	significant
LOC_000001	XLOC_000001	DDX11L1	chr1:11868-31109	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000002	XLOC_000002	MIR1302-2	chr1:11868-31109	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000003	XLOC_000003	AL627309.2	chr1:53048-54936	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000004	XLOC_000004	OR4G11P	chr1:62947-63887	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000005	XLOC_000005	OR4F5	chr1:69090-70008	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000006	XLOC_000006	RP11-34P13.10	chr1:89294-134836	q1	q2	NOTEST	0.0162592	0.00529659	-1.61812	0	1	1	no
LOC_000007	XLOC_000007	RP11-34P13.9	chr1:141473-173862	q1	q2	NOTEST	0	0	0	0	1	1	no
LOC_000008	XLOC_000008	RP4-669L17.10,RP4-669L17.4,RP4-669L17.8	chr1:317719-461954	q1	q2	NOTEST	0.205269	0.146199	-0.489585	0	1	1	no

  * locus (assigned by cuffmerge), gene_id, name of gene (reference annotation), locus again, sample1 and sample2, status (NOTEST, no test could be performed, HIDATA, too many reads for test to be precise, LOWDATA too little reads, OK), value_1 (avg fkpm for first dataset), value_2, log2 fold change (control vs test), test value, interpretation pvalue and q-value, whether difference is statistically significant or not ⇒ don't get it?

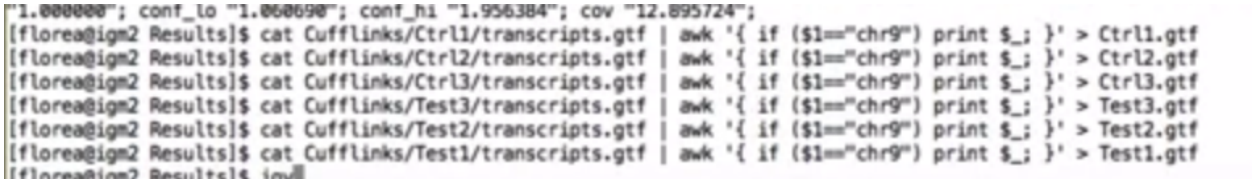
- to obtain list of genes differentially expressed between control and test, just look at the last column

  grep -c "yes" # gives you the number of genes differentially expressed. Can do same analysis on any .diff file.

- splicing.diff file is different: expression percentages for the test datasets and control datasets. File compares expression percentages.

Tools for Transcriptomics 6.1: Integrated Genomics Viewer I (IGV)

extract only the chromosome you need from the transcripts.gtf from cufflinks.



igvtools sort (.gtf file of the part you want to show from transcripts.gtf from cufflinks) (name of the sorted file) # do this for each of the files above

igvtools index (sorted .gtf file) # creates a .idx file, do for all of the files above.

get the accepted_hits.bam and .bai files from tophat
samtools view -b (accepted_hits.bam) “chr9” > chr9.test1.bam # repeat this command for the other 5 files.

samtools index chr9.test1.bam # repeat for the other 5 files.

IGV

Download from here:
https://igv.org/doc/desktop/#DownloadPage/

load all the sorted .gtf files (igvtools sort (.gtf file of the part you want to show from transcripts.gtf from cufflinks) (name of the sorted file) # do this for each of the files above) <- from here

load all the bam files (samtools view -b (accepted_hits.bam) “chr9” > chr9.test1.bam # repeat this command for the other 5 files.) <- from here