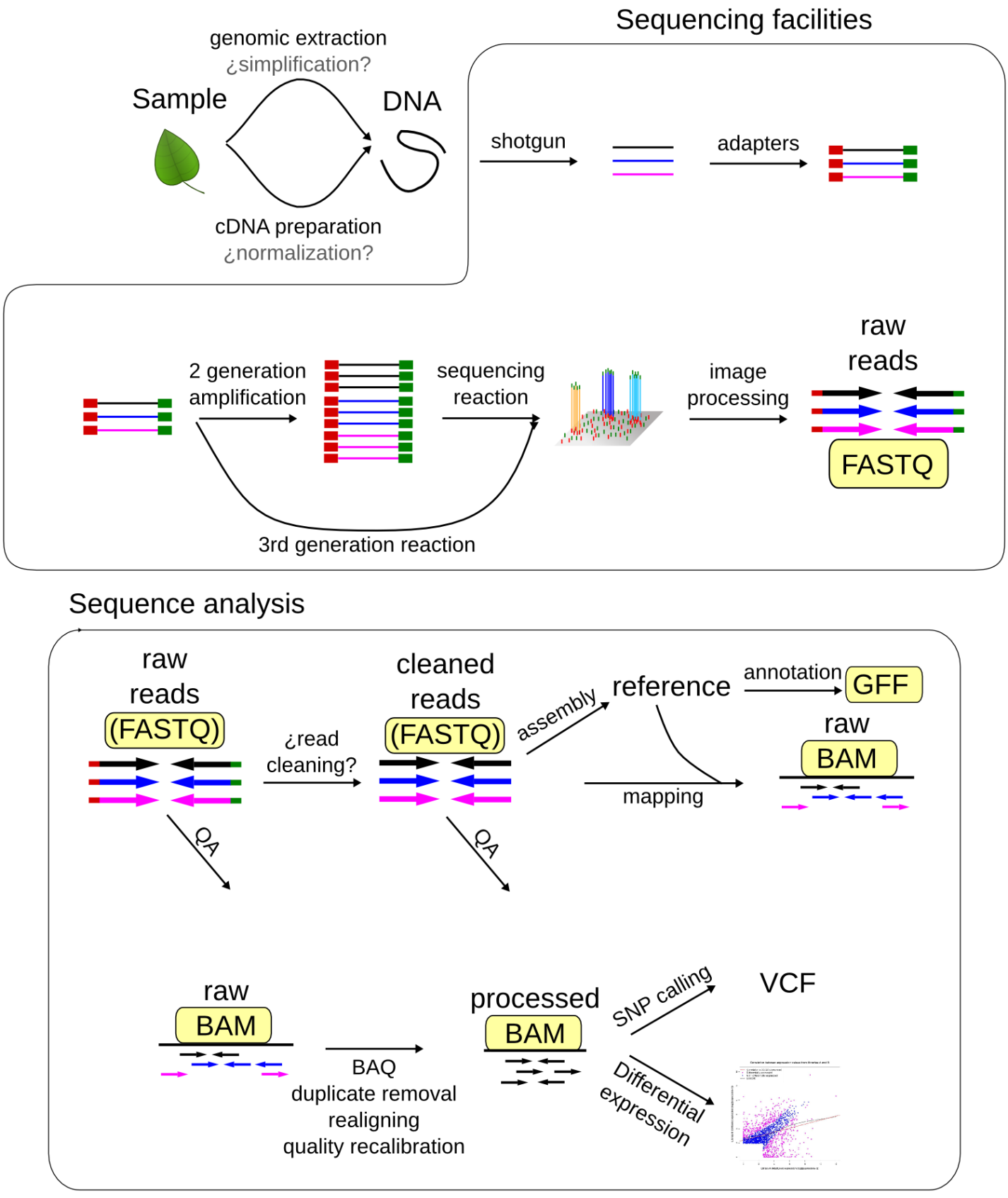
Workshop 5 Sequence formats

Introduction

> The Road to Hell is Paved with Bioinformatics Formats

Biology, computational biology, bioinformatics, genomics, genetics are fields with their own history of data managment and requirenement. Each lab, each softawere used generated 1000s of files ALL of different formats...



(c) Bioinformatics at COMAV

The classics

FASTA

A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (`>`) symbol at the beginning. It is recommended that all lines of text be shorter than 80 characters in length. An example sequence in FASTA format is:

```
>P01013 GENE X PROTEIN (OVALBUMIN-RELATED)
QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPFHVTKQESKPVQMMCMNNSFNVATLPAE
KMKILELPFASGDLSMLVLLPDEVSDLERIEKTINFEKLTEWTNPNTMEKRRVKVYLPQMKIEEKYNLTS
VLMALGMTDLFIPSANLTGISSAESLKISQAVHGAFMELSEDGIEMAGSTGVIEDIKHSPESEQFRADHP
FLFLIKHNPTNTIVYFGRYWSP
```

Sequences are expected to be represented in the standard IUB/IUPAC amino acid and nucleic acid codes, with these exceptions: lower-case letters are accepted and are mapped into upper-case; a single hyphen or dash can be used to represent a gap of indeterminate length; and in amino acid sequences, U and * are acceptable letters. The nucleic acid codes are:

```
A  adenosine          C  cytidine             G  guanine
T  thymidine          N  A/G/C/T (any)        U  uridine
K  G/T (keto)         S  G/C (strong)         Y  T/C (pyrimidine)
M  A/C (amino)        W  A/T (weak)           R  G/A (purine)
B  G/T/C              D  G/A/T                H  A/C/T
V  G/C/A              -  gap of indeterminate length
```

The accepted amino acid codes are:

```
A  alanine               P  proline
B  aspartate/asparagine  Q  glutamine
C  cystine               R  arginine
D  aspartate             S  serine
E  glutamate             T  threonine
F  phenylalanine         U  selenocysteine
G  glycine               V  valine
H  histidine             W  tryptophan
I  isoleucine            Y  tyrosine
K  lysine                Z  glutamate/glutamine
L  leucine               X  any
M  methionine            *  translation stop
N  asparagine            -  gap of indeterminate length
```

FASTQ

The fastq format was developed to provide a convenient way of storing the sequence and the quality scores in the same file. These are text files and they look like:

```
@seq_1
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
@seq_2
ATCGTAGTCTAGTCTATGCTAGTGCGATGCTAGTGCTAGTCGTATGCATGGCTATGTGTG
+
208DA8308AD8SF83FH0SD8F08APFIDJFN34JW830UDS8UFDSADPFIJ3N8DAA
```

In this file every sequence has 4 lines. In the first line we get the sequence name after the symbol `@` and, optionally, the description. The second line has the sequence and the fourth line has the quality scores encoded as letters.

Genbank

One sequence in GenBank format starts with a line containing the word `LOCUS` and a number of annotation lines. The start of the sequence is marked by a line containing `ORIGIN` and the end of the sequence is marked by two slashes (`//`).

An example sequence in GenBank format is:

```
LOCUS       AB000263                368 bp    mRNA    linear   PRI 05-FEB-1999
DEFINITION  Homo sapiens mRNA for prepro cortistatin like peptide, complete
            cds.
ACCESSION   AB000263
ORIGIN
        1 acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg
       61 ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg
      121 caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc
      181 aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
      241 gcgcacccca ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga
      301 agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca
      361 gacctgaa
//
```

BAM/SAM

SAM stands for Sequence Alignment/Map format. It is a Tabulation-delimited text format consisting of a header section, which is optional, and an alignment section. If present, the header must be prior to the alignments. Header lines start with `@` , while alignment lines do not. Each alignment line has **11**

mandatory fields for essential alignment information such as mapping position, and variable number of optional fields for flexible or aligner specific information.

Each Alignment has:

- query name, **QNAME** (SAM)/read_name (BAM).
- a bitwise set of information describing the alignment, **FLAG**. Provides the following information:
    - are there multiple fragments?
    - are all fragments properly aligned?
    - is this fragment unmapped?
    - is the next fragment unmapped?
    - is this query the reverse strand?
    - is the next fragment the reverse strand?
    - is this the 1st fragment?
    - is this the last fragment?
    - is this a secondary alignment?
    - did this read fail quality controls?
    - is this read a PCR or optical duplicate?
- reference sequence name, **RNAME**, often contains the Chromosome name.
- leftmost position of where this alignment maps to the reference, **POS**. For SAM, the reference starts at 1.
- mapping quality, **MAPQ**, which contains the "phred-scaled posterior probability that the mapping position" is wrong...
- string indicating alignment information that allows the storing of clipped, **CIGAR**.
- the reference sequence name of the next alignment in this group, **MRNM** or **RNEXT**. In paired alignments, it is the mate's reference sequence name. (A group is alignments with the same query name.)
- leftmost position of where the next alignment in this group maps to the reference, **MPOS** or **PNEXT**.
- length of this group from the leftmost position to the rightmost position, **ISIZE** or **TLEN**
- the query sequence for this alignment, **SEQ**
- the query quality for this alignment, **QUAL**, one for each base in the query sequence.
- Additional optional information is also contained within the alignment, **TAG**s. A bunch of different information can be stored here and they appear as key/value pairs. See the spec for a detailed list of commonly used tags and what they mean.

Example Header Lines

```
@HD    VN:1.0  SO:coordinate
@SQ    SN:1    LN:249250621    AS:NCBI37   UR:file:/data/local/ref/GATK/human_g1k_v
@SQ    SN:2    LN:243199373    AS:NCBI37   UR:file:/data/local/ref/GATK/human_g1k_v
@PG    ID:bwa  VN:0.5.4
```

Example Alignments

This is what the alignment section of a SAM file looks like:

```
1:497:R:-272+13M17D24M 113 1   497 37  37M 15  100338662   0   CGGGTCTGACCTGAGGAGAA
19:20389:F:275+18M2D19M    99  1   17644   0   37M =   17919   314 TATGACTGCTAATAAT
19:20389:F:275+18M2D19M    147 1   17919   0   18M2D19M    =   17644   -314    GTAC
9:21597+10M2I25M:R:-209    83  1   21678   0   8M2I27M =   21469   -244    CACCACAT
```

`FLAG` field

The FLAG field encodes various pieces of information about the individual read, which is particularly important for PE reads. It contains an integer that is generated from a sequence of Boolean bits (0, 1). This way, answers to multiple binary (Yes/No) questions can be compactly stored as a series of bits, where each of the single bits can be addressed and assigned separately.

The following table gives an overview of the different properties that can be encoded in the FLAG field. The developers of the SAM format and samtools tend to use the hexadecimal encoding as a means to refer to the different bits in their documentation. The value of the FLAG field in a given SAM file, however, will always be the decimal representation of the sum of the underlying binary values (as shown in Table below, row 2).

Binary (Decimal)	Hex	Description
00000000001 (1)	0x1	Is the read paired?
00000000010 (2)	0x2	Are both reads in a pair mapped "properly" (i.e., in the correct orientation with respect to one another)?
00000000100 (4)	0x4	Is the read itself unmapped?
00000001000 (8)	0x8	Is the mate read unmapped?
00000010000 (16)	0x10	Has the read been mapped to the reverse strand?
00000100000 (32)	0x20	Has the mate read been mapped to the reverse strand?
00001000000 (64)	0x40	Is the read the first read in a pair?
00010000000 (128)	0x80	Is the read the second read in a pair?
00100000000 (256)	0x100	Is the alignment not primary? (A read with split matches may have multiple primary alignment records.)
01000000000 (512)	0x200	Does the read fail platform/vendor quality checks?
10000000000 (1024)	0x400	Is the read a PCR or optical duplicate?

From tutorial by Friederike Dündar, Luce Skrabanek, and Paul Zumbo

In a run with single reads, the flags you most commonly see are:

- 0: This read has been mapped to the forward strand. (None of the bit-wise flags have been set.)
- 4: The read is unmapped (`0x4` is set).
- 16: The read is mapped to the reverse strand (`0x10` is set)

(`0x100` , `0x200` and `0x400` are not used by most aligners/mappers, but could, in principle be set for single reads.) Some common `FLAG` values that you may see in a PE experiment include:

69 (= 1 + 4 + 64)	The read is paired, is the first read in the pair, and is unmapped.
77 (= 1 + 4 + 8 + 64)	The read is paired, is the first read in the pair, both are unmapped.
83 (= 1 + 2 + 16 + 64)	The read is paired, mapped in a proper pair, is the first read in the pair, and it is mapped to the reverse strand.
99 (= 1 + 2 + 32 + 64)	The read is paired, mapped in a proper pair, is the first read in the pair, and its mate is mapped to the reverse strand.
133 (= 1 + 4 + 128)	The read is paired, is the second read in the pair, and it is unmapped.
137 (= 1 + 8 + 128)	The read is paired, is the second read in the pair, and it is mapped while its mate is not.
141 (= 1 + 4 + 8 + 128)	The read is paired, is the second read in the pair, but both are unmapped.
147 (= 1 + 2 + 16 + 128)	The read is paired, mapped in a proper pair, is the second read in the pair, and mapped to the reverse strand.
163 (= 1 + 2 + 32 + 128)	The read is paired, mapped in a proper pair, is the second read in the pair, and its mate is mapped to the reverse strand.

A useful website for quickly translating the FLAG integers into plain English explanations like the ones shown above is: https://broadinstitute.github.io/picard/explain-flags.html

`CIGAR` string

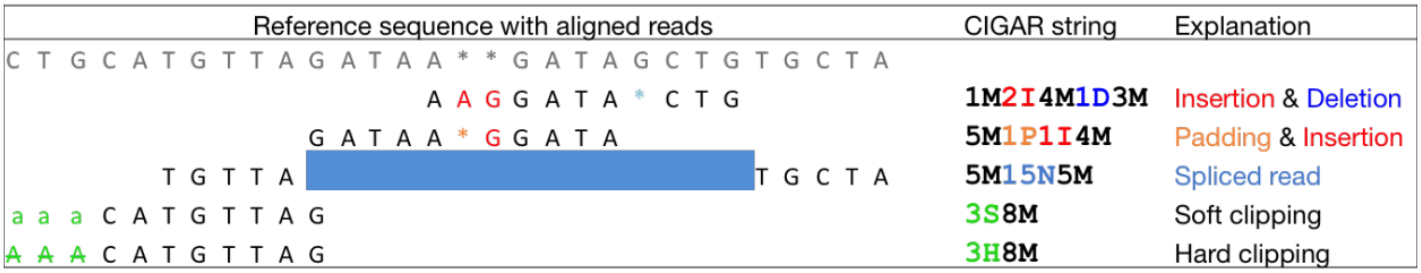
`CIGAR` stands for *Concise Idiosyncratic Gapped Alignment Report*. This sixth field of a SAM file contains a so-called CIGAR string indicating which operations were necessary to map the read to the reference sequence at that particular locus.

The following operations are defined in CIGAR format (also see figure below):

- **M** - Alignment (can be a sequence match or mismatch!)

- **I** - Insertion in the read compared to the reference
- **D** - Deletion in the read compared to the reference
- **N** - Skipped region from the reference. For mRNA-to-genome alignments, an N operation represents an intron. For other types of alignments, the interpretation of N is not defined.
- **S** - Soft clipping (clipped sequences are present in read); S may only have H operations between them and the ends of the string
- **H** - Hard clipping (clipped sequences are NOT present in the alignment record); can only be present as the first and/or last operation
- **P** - Padding (silent deletion from padded reference)
- **=** - Sequence match (not widely used)
- **X** - Sequence mismatch (not widely used)

The sum of lengths of the **M**, **I**, **S**, **=**, **X** operations must equal the length of the read. Here are some examples:



From tutorial by Friederike Dündar, Luce Skrabanek, and Paul Zumbo.

Optional fields

Following the eleven mandatory SAM file fields, the optional fields are presented as key-value pairs in the format of `<TAG>:<TYPE>:<VALUE>`, where `TYPE` is one of:

- `A` - Character
- `i` - Integer
- `f` - Float number
- `Z` - String
- `H` - Hex string

The information stored in these optional fields will vary widely depending on the mapper and new tags can be added freely. In addition, reads within the same SAM file may have different numbers of optional fields, depending on the program that generated the SAM file. Commonly used optional tags include:

- `AS:i` - Alignment score
- `BC:Z` - Barcode sequence
- `HI:i` - Match is i-th hit to the read
- `NH:i` - Number of reported alignments for the query sequence
- `NM:i` - Edit distance of the query to the reference
- `MD:Z` - String that contains the exact positions of mismatches (should complement the CIGAR

string)
- `RG:Z` - Read group (should match the entry after ID if @RG is present in the header.

Thus, for example, we can use the NM:i:0 tag to select only those reads which map perfectly to the reference(i.e., have no mismatches). While the optional fields listed above are fairly standardized, tags that begin with `X`, `Y`, and `Z` are reserved for particularly free usage and will never be part of the official SAM file format specifications. `XS`, for example, is used by TopHat (an RNA-seq analysis tool we will discuss later) to encode the strand information (e.g., `XS:A:+`) while Bowtie2 and BWA use `XS:i:` for reads with multiple alignments to store the alignment score for the next-best-scoring alignment (e.g., `XS:i:30`).

Manipulating SAM/BAM datasets

We support four major toolsets for processing of SAM/BAM datasets:

- DeepTools - a suite of user-friendly tools for the visualization, quality control and normalization of data from deep-sequencing DNA sequencing experiments.
- SAMtools - various utilities for manipulating alignments in the SAM/BAM format, including sorting, merging, indexing and generating alignments in a per-position format.
- BAMtools - a toolkit for reading, writing, and manipulating BAM (genome alignment) files.
- Picard - a set of Java tools for manipulating high-throughput sequencing data (HTS) data and formats.

Complete example: Tiki Wiki CMS Groupware

Full BAM/SAM Specification.

BAM file ares **mostly** compressed SAM files.

VCF

VCF is a text file format (most likely stored in a compressed manner). It contains meta-information lines, a header line, and then data lines each containing information about a position in the genome. The format also has the ability to contain genotype information on samples for each position.

```
##fileformat=VCFv4.0
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=1000GenomesPilot-NCBI36
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF ALT    QUAL FILTER INFO                              FO
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;D
20     1230237 .         T      .       47   PASS   NS=3;DP=13;AA=T
20     1234567 microsat1 GTCT   G,GTACT 50   PASS   NS=3;DP=9;AA=G
```

The header line names the 8 fixed, mandatory columns. These columns are as follows:

```
CHROM
POS
ID
REF
ALT
QUAL
FILTER
INFO
```

Full VCF Specification.

GFF

GFF, or the **General Feature Format** is used to describe genes and other features of DNA, RNA and protein sequences. It comes with the .gff extension.

GFF consists of one line per feature, each containing **9** columns of data. Each column is separated by a tab, making it a tabs-delimited file.

```
<seqname> <source> <feature> <start> <end> <score> <strand> <frame> [attributes]
```

- **refseq name**. Name of chromosome or scaffold. Chromosomes can be given without the 'chr' prefix.
- **source**. Source of annotation, name of program that generated this feature.
- **feature**. Feature type name: "Gene", "variation", "similarity", etc.
- **start**. Start position, starting at 1.
- **end**. End position, starting at 1.
- **score**. Floating point value. For scores such as similarity, identity, etc.
- **strand**. '+' for forward and '-' for reverse.
- **frame**. Either 0, 1 or 2. 0 indicates first base of the feature is first base of codon, 1 indicates second base of feature is the first base of a codon, etc.
- **attribute**. **Semicolon-separated** list of feature attributes in the format tag**=**value.

```
##gff-version 3.2.1
##sequence-region ctg123 1 1497228
ctg123 . gene            1000 9000  .  +  .  ID=gene00001;Name=EDEN
ctg123 . TF_binding_site 1000 1012  .  +  .  Parent=gene00001
ctg123 . mRNA            1050 9000  .  +  .  ID=mRNA00001;Parent=gene00001
ctg123 . mRNA            1050 9000  .  +  .  ID=mRNA00002;Parent=gene00001
ctg123 . mRNA            1300 9000  .  +  .  ID=mRNA00003;Parent=gene00001
ctg123 . exon            1300 1500  .  +  .  Parent=mRNA00003
ctg123 . exon            1050 1500  .  +  .  Parent=mRNA00001,mRNA00002
ctg123 . exon            3000 3902  .  +  .  Parent=mRNA00001,mRNA00003
```

Full GFF3 Specification.

GTF

GTF stands for **Gene transfer format**. It borrows from GFF, but has additional structure that warrants a separate definition and format name.

Structure is as GFF, so the fields are:

```
<seqname> <source> <feature> <start> <end> <score> <strand> <frame> [attributes]
```

Here is a simple example with 3 translated exons. Order of rows is not important.

```
381 Twinscan  CDS          380   401   .   +   0  gene_id "001"; transcript_id "001.
381 Twinscan  CDS          501   650   .   +   2  gene_id "001"; transcript_id "001.
381 Twinscan  CDS          700   707   .   +   2  gene_id "001"; transcript_id "001.
381 Twinscan  start_codon  380   382   .   +   0  gene_id "001"; transcript_id "001.
381 Twinscan  stop_codon   708   710   .   +   0  gene_id "001"; transcript_id "001.
```

Fields must be separated by a single Tabulation and no white space.

...

The following **feature** types are required: "CDS", "start*codon", "stop*codon". The features "5UTR", "3UTR", "inter", "inter*CNS", "intron*CNS" and "exon" are optional. All other features will be ignored.

...

All nine features have the same two mandatory attributes at the end of the record:

- gene_id value; A globally unique identifier for the genomic locus of the transcript. If empty, no gene is associated with this feature.
- transcript_id value; A globally unique identifier for the predicted transcript. If empty, no transcript is associated with this feature.

These attributes are designed for handling multiple transcripts from the same genomic region. Any other attributes or comments must appear after these two and will be ignored.

Attributes must end in a **semicolon** which must then be separated from the start of any subsequent attribute by exactly **one space character** (NOT a tab character).

Full GTF 2.2 Specifications.

Hope

The tools kit EMBOSS is providing in "mirable" tools `Seqret` that converts most of the sequence format... most of the time.

Here is the EBI online implementation: Seqret

From NGS to genomeics

Sequencer produce natively various file format:

- Illumina: FASTQ (single or paired)
- PacBio: BAX often converted in BAM
- Oxford Nanopore: HDF5 can be manually converted BAM or FASTQ

Database store and report structured data

- NCBI/Genbank: Genbank
- EBI: EMBL

Software... well they do whatever they are designed to...

> "The first step in developing a new genetic analysis algorithm is to decide how to make the input data file format different from all pre-existing analysis data file formats."

A prime exemplar of this Law is the use of different codes to signify the sex of animals. For example, crimap uses '0' to represent female and '1' to represent male. The algorithm designed by Keightly et al. uses the same codes to mean the opposite sexes.The Knott & Haley QTL analysis algorithm uses codes '1' and '2'. The list goes on.