Introduction to Linux for Bioinformatics

A practical approach
1st edition

David da Silva Pires
Cell Cycle Laboratory
Butantan Institute

São Paulo
June 17, 2024

Contents

1 FASTA

1.1 The format

The FASTA file format is a widely used file format in the field of bioinformatics. It is text-based and it is used to store both nucleotide (DNA or RNA) and amino acid (protein) sequence data.

The origin of the FASTA format is due to an homonimous software package, developed in the late 1980 by David J. Lipman and William R. Pearson. The software was one of the first widely-used database similarity search tool. It was designed to quickly compare DNA or protein sequences to search for similarities and differences and the FASTA format was used to store and exchange the input sequences. The program can still be used if you are working with limited resources or needs a more simple approach.

Its simplicity allows easy manipulation and analysis of the sequences using text processing tools and script languages like Bash, R and Python.

It is an essential tool for researches working in fields such as genomics, proteomics and evolutionary biology as it provides a convenient way to store, share and analyze large amounts of sequence data.

Since its development, the FASTA format has become a widely used standard in the bioinformatics community and has been implemented in many other software programs and databases.

Some examples of software programs and databases that support FASTA files include:

BLAST: The Basic Local Alignment Search Tool is a widely used software program for searching for similarities between sequences. URL: `https://blast.ncbi.nlm.nih.gov/Blast.cgi`.

ClustalW: A software program for aligning sequences.

GenBank: A database of DNA sequences from many organisms. URL: `https://www.ncbi.nlm.nih.gov/genbank`.

Uniprot: A database of protein sequences from many organisms. URL: `https://www.uniprot.org`.

In a FASTA file, each sequence is represented by a single description line (header) followed by one or more lines of sequence data.

Examples

- Gene (partial coding sequence of SARS-CoV-2 Spike protein):

```
>hCoV-19_Spike_CDS
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTCATGCCGCTGTTT
AATCTTATAACTACAACTCAATCATACACTAATTCTTTCACACGTGGTGTTTATTACCCT
... (60 more lines)
CTCAAGGGCTGTTGTTCTTGTGGATCCTGCTGCAAATTTGATGAAGACGACTCTGAGCCA
GTGCTCAAAGGAGTCAAATTACATTACACATAA
```

- Protein (complete SARS-CoV-2 Spike protein):

```
>hCoV-19_Spike_protein
MFVFLVLLPLVSSQCVMPLFNLITTTQSYTNSFTRGVYYPDKVFRSSVLHLTQDLFLPFF
SNVTWFHAISGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIVN
NATNVFIKVCEFQFCNDPFLDVYHKNNKSWMESESGVYSSANNCTFEYVSQPFLMDLEGK
QGNFKNLREFVFKNIDGYFKIYSKHTPIIGRDFPQGFSALEPLVDLPIGINITRFQTLLA
LNRSYLTPGDSSSGWTAGAADYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETKCTL
KSFTVEKGIYQTSNFRVQPTESIVRFPNVTNLCPFHEVFNATRFASVYAWNRTRISNCVA
DYSVLYNFAPFFAFKCYGVSPTKLNDLCFTNVYADSFVIRGNEVSQIAPGQTGNIADYNY
KLPDDFTGCVIAWNSNKLDSKHSGNYDYWYRSFRKSKLKPFERDISTEIYQAGNKPCKGV
KGPNCYFPLQSYGFRPTYGVGHQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVNFNF
NGLTGTGVLTKSNKKFLPFQQFGRDIVDTTDAVRDPQTLEILDITPCSFGGVSVITPGTN
TSNQVAVLYQGVNCTEVSVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEYVNNSYECD
IPIGAGICASYQTQTKSRRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTISVT
TEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLKRALTGIAVEQDKNTQEVFA
QVKQIYKTPPIKYFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDCLGD
IAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAMQMA
YRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLFSTTSALGKLQDVVNHNAQALNTLV
KQLSSKFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRASAN
LAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPAICH
DGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDPLQL
ELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDLQEL
GKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCCKFDEDDSEP
VLKGVKLHYT
```

- Genome (partial, SARS-CoV-2):

```
>hCoV-19/Brazil/DF-LACENDF/2024|EPI_ISL_19165654|2024-03-11
TACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACG
AACTTTAAAATCTGTGTGGCTGTCACTCGGCTGCATGCTTAGTGCACTCACGCAGTATAA
... (493 more lines)
GAACAATGCTAGGGAGAGCTGCCTATATGGAAGAGCCCTAATGTGTAAAATTAATTTTAG
TA
```

- Adapter:

```
>ONT_LA_TS Oxford Nanopore Ligation Adapter Top Strand
TTTTTTTTCCTGTACTTCGTTCAGTTACGTATTGCT
```

The header

Model:

```
>unique_id optional additional information
```

The format allows identification names and comments preceding the sequences.

FASTA record indicator: '>'. The description line starts with a greater than sign symbol.

Unique sequence identifier: it follows immediately after the '>' symbol, without even a single space character between the two. It can be an overall adopted identificator, used at official databases, or an arbitrary code, like "gene1", "gene2", etc.

Additional information: the field after the unique ID is optional. It can contain the locus, the name of the gene, the functional annotation, the related species, the length of the sequence, data about the sequencing machine, etc.

The sequence data

Model:

```
AGATAATACAAGAGA
```

The sequence data consists of a string of letters representing the nucleotide bases, for DNA or RNA sequences, or amino acids, for protein sequences.

If the sequence is too long, it is usual to limit the number of characters per line. Common limits are 60 and 80. This way, it is easier to read all the sequence by moving the contents of the file vertically instead of horizontally.

Nucleotides Each nucleotide is represented by only one letter.

Extensions: fasta, fa, fna, ffn, frn.

Masking In bioinformatics, masking in FASTA files is a technique used to hide or mark certain regions of a genomic sequence, typically repetitive regions, to facilitate further analysis such as read mapping. There are two main types of masking: hard masking and soft masking.

Hard masking involves replacing the nucleotides in the regions to be masked with the letter 'N'. For example, if a segment of a genome is hard-masked, a sequence that originally reads 'ATGCGTACG' might be converted to 'ATNNNNACG'. This method is used to completely hide repetitive or low-complexity regions from analysis. By replacing nucleotides with 'N', these regions are effectively ignored in downstream processes such as sequence alignment or assembly. This can help prevent mismatches and false positives that might occur due to the repetitive nature of these sequences.

In contrast, soft masking retains the nucleotide information but changes the case of the letters to lowercase. For instance, a sequence that originally reads 'ATGCGTACG' might be converted to 'ATgcgtaCG' in its soft-masked form. This approach is used to mark regions as repetitive or low-complexity while still preserving the nucleotide sequence information. As a result, certain bioinformatics tools can decide whether to use these regions based on the context of the analysis. For example, some aligners might ignore the lowercase regions during the initial alignment phase but still use them to refine the alignment later.

The applications of masking, whether hard or soft, are primarily focused on highlighting repetitive sequences in the genome, such as transposable elements, satellite DNA, or other low-complexity regions. These regions can complicate various genomic analyses because their repetitive nature can lead to ambiguous alignments and false positives. In the context of read mapping, masked regions can be disregarded or treated with caution to improve the accuracy of the mapping. This is particularly important in next-generation sequencing (NGS) where reads are often short and repetitive sequences can cause multiple potential alignment sites, leading to confusion and errors in the mapping process.

Thus, hard masking and soft masking serve crucial roles in managing repetitive regions in genomic analyses. Hard masking hides these regions completely, while soft masking marks them in a way that retains the sequence information. Both methods aim to prevent alignment errors and improve the accuracy of processes such as read mapping, ensuring more reliable and precise genomic analyses.

Amino acids Each aminoacid is represented by a code composed by only one character.

Extension: fasta, fa, faa, pep.

Types

Simple FASTA: A simple FASTA file contains a single sequence.

multiFASTA: In contrast, a multiFASTA file contains multiple sequences, each with its own header and sequence lines. The sequences are concatenated in the same file, and each new sequence starts with a '>' character. This format is useful for storing and managing multiple sequences within a single file, which is often needed in genomic analyses where large datasets comprising many sequences must be processed. An example of a multiFASTA file is shown below:

```
>sequence_1
ATGCGTACGTTAGC
>sequence_2
CGTAGCTAGCTAGC
>sequence_3
TGCATGCATGCA
```

FASTA 2-line: The FASTA 2-line format is a specific variant of the FASTA format where each sequence is represented using exactly two lines: one for the header and one for the sequence itself. In traditional FASTA files, the sequence can span multiple lines, with each line typically being 60-80 characters long. However, in the FASTA 2-line format, the entire sequence is presented on a single line, directly following the header line.

Here's an example of the traditional FASTA format:

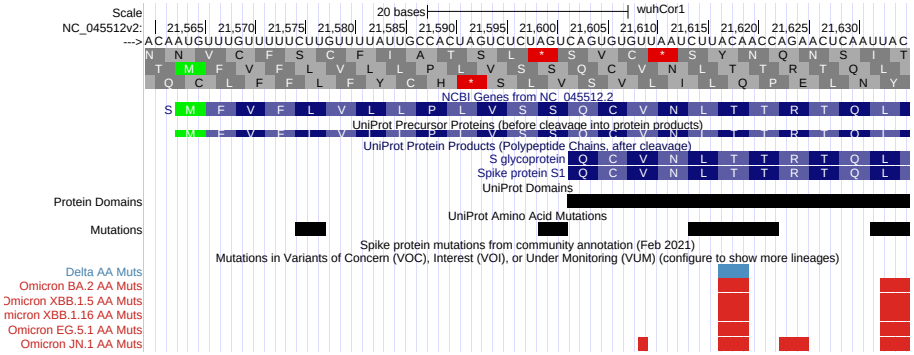
```
>sequence_1
ATGCGTACGTTAGCTAGCGTAGCTAGCTAGCT
AGCGTAGCTAGCTAGCTAGCTAGCTAGCTAGC
>sequence_2
CGTAGCTAGCTAGCTAGCGTAGCTAGCTAGCT
AGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCT
```

In this format, each sequence can be broken into multiple lines for readability and compatibility with certain tools.

In contrast, the FASTA 2-line format would look like this:

```
>sequence_1
ATGCGTACGTTAGCTAGCGTAGCTAGCTAGCTAGCGTAGCTAGCTAGCTAGCTAGCTAGC
>sequence_2
CGTAGCTAGCTAGCTAGCGTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCT
```

1.2 Visualization



1.3 Exercises

① **Creating a file with the command `cat`.** Use the command `cat` to create a file called `methionine.fasta` whose contents is a FASTA record identified as "methionine" at the header and with the sequence "ATG" in the second line.

② **Creating a file with the `nano` editor.** Use the editor `nano` to create a file called `stopCodons.fasta` whose contents are three FASTA records, each one with the sequence of one of the three possible stop codons: amber (TAG), ochre (TAA) and opal (TGA).

③ **Searching for patterns.** List only the header lines of a multiFASTA file.

④ **Masked genomes.** Determine which genome in the `fileExamples` directory is unmasked, which is hard masked, and which is soft masked.

⑤ **Sequence count.** Count the number of records in a multiFASTA file.

⑥ **Processing FASTA metadata.** Remove the description (metadata) from each record in a multiFASTA file, leaving only the identifier and the sequence.

⑦ **List of IDs.** Obtain a list of all identifiers from a multiFASTA file.

⑧ **Locating specific sequences.** Count how many sequences in a genome file contain the motif `AGATAGAGA`. And the motifs `AGATAATACA` and `AGATATAGAGA`?

⑨ **Sequence extraction.** Extract a specific sequence from the FASTA file based on its header.

⑩ **Sort by length.** Sort the sequences in the FASTA file based on their lengths, from shortest to longest.

❶ **Short sequence removal.** Write a script that removes all sequences from the FASTA file that are less than 10000 nucleotides.

❷ **Translation of DNA sequences.** Translate DNA sequences into proteins.

2 FASTQ

2.1 The format

The standard file format of sequences produced by sequencing machines like Illumina is the FASTQ format. Each library generates files with millions of reads. Each read has four lines.

Example

Illumina SARS-CoV-2 sequencing `example.fastq`:

```
@VH00451:1:AAAJ5GWM5:1:1101:43624:1095 2:N:0:CCAAGAGGTG+CTGGCTCGTT
GCATTAATACAGCCACCATCGTAACAATCAAAGTACTTATCAACAACTTCAACTACAAATAGTAGTTGTCT
+
CCCCCCCCCCCCCCCCCCCCCCC;CCCCCCCCCCCCCCC-CCCCCCCCCCCC-CCCCCC;C;CCCCCCCCCC
@VH00451:1:AAAJ5GWM5:1:1101:44855:1114 2:N:0:CCAAGAGGTG+CTGGCTCGTT
ATATCTGGGTTTTCTACAAAATCATACCAGTCCTTTTTATTGAAATAATCATCATCACAACAATTGTATGT
+
CCCCCCCCCCCCCCCCCCC-CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC;CCCCCCCCCCCCC-
```

The sequence header

It contains a unique ID for each read. It may contain the name of the machine that sequenced the read, the name of the project, the number of the lane at the flow cell (from the eight available), the exact location where that sequence was located in that lane. In the case of a paired-end read, it can also contain which of the pairs that read represents.

The character sequence

The second line of the read has the DNA (or cDNA in case of RNA) sequence.

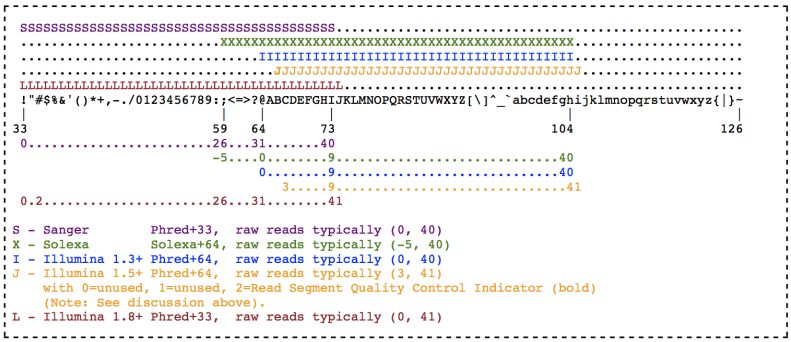
The quality header

Nowadays it contains a '+' signal, which is basically a placeholder. But originally, the plus signal was followed by the same unique ID present at the sequence header.

The quality sequence

The fourth line contains the quality of the sequence.

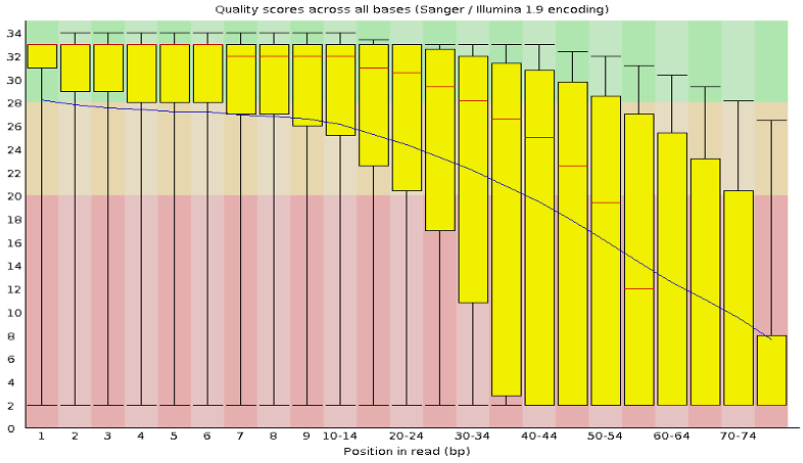
The quality score is usually encoded as Phred+33, which goes from 0 to 41.



It is usual at Illumina sequences that the highest quality bases are in the beginning of the sequence (5' side) and that it decreases as it approaches the end of the sequence (3' side).

Phred+33 is a scoring system that encodes the quality as a probability of the correctness of each base in a sequence. The higher the value, the higher the probability that the base is correct.

2.2 Visualization



2.3 Exercises

① **Sequence count and average quality.** Write a command to count the total number of sequences in a FASTQ file and calculate the average length across all sequences.

② **Removing low-quality sequences.** Write a command to remove all sequences from the FASTQ file that have an average quality below 20.

③ **Conversion to FASTA Format.** Convert the FASTQ file to FASTA format, keeping only the sequence and discarding quality information.

④ **Extraction of Sequences with Specific Motif.** Extract all sequences from the FASTQ file that contain a specific nucleotide motif.

⑤ **Filtering by sequence length.** Remove all sequences from the FASTQ file that are less than 100 nucleotides long.

⑥ **Calculating average sequence length.** Calculate the average length of sequences in the FASTQ file.

⑦ **Nucleotide counting.** Count the number of each type of nucleotide (A, C, G, T) in the FASTQ file.

⑧ **Removing duplicate sequences.** Remove duplicate sequences from the FASTQ file.

3 CSV

3.1 The format

CSV stands for **C**omma **S**eparated **V**alues. CSV files are a way to store data in a very simple format. They are basically plain text files that contain a spreadsheet. Each row of the text file is a separate row in the spreadsheet. Each column is separated by a comma. Thus *comma separated values*.

The advantage to a CSV file is it is not proprietary. It is not a Microsoft Excel file that can only be open in Excel or specific to any spreadsheet software. It is a very basic format that can be opened in any spreadsheet like MS Excel or Google's Sheets. So when someone wants to provide some data in a very basic format that doesn't rely on the other person having a particular app, a CSV file is a great way to do that.

Examples

sample.csv

```
Store,Apples,Oranges,Pears
A,32,15,20
B,67,96,87
C,47,67,21
D,83,10,17
E,87,82,86
F,15,17,92
```

employees.csv

```
Name,Age,Occupation
John Doe,30,Software Developer
Jane Smith,28,Data Analyst
Alice Johnson,35,Project Manager
```

3.2 Visualization

```
1 csvlook employees.csv
```

```
| Name          | Age | Occupation         |
| ------------- | --- | ------------------ |
| John Doe      |  30 | Software Developer |
| Jane Smith    |  28 | Data Analyst       |
| Alice Johnson |  35 | Project Manager    |
```

3.3 Exercises

① **CSV data extraction** Write a command to extract specific columns (e.g., GeneID and Expression Level) from a CSV file containing gene expression data.

② **Data filtering based on conditions.** Filter the rows in a CSV file containing mutation data to only include deepeness greater than 50.

4 GFF

4.1 The format

GFF stands for **G**eneral **F**eature **F**ormat. Other options are **G**ene-**F**inding **F**ormat and **G**eneric **F**eature **F**ormat. It is a file format used for describing genes and other features of DNA, RNA and protein sequences.

GFF files are composed by nine columns:

① **seqid:** The name of the sequence where the feature is located.

② **source:** Keyword identifying the source of the feature, like a program (e.g., Augustus or RepeatMasker) or an organization (like TAIR).

③ **type:** The feature type name, like *gene* or *exon*. In a well structured GFF file, all the children features always follow their parents in a single block (so all exons of a transcript are put after their parent *transcript* feature line and before any other parent transcript line). In GFF3, all features and their relationships should be compatible with the standards released by the Sequence Ontology Project.

④ **start:** Genomic start of the feature, with a 1-based offset. This is in contrast with other 0-offset half-open sequence formats, like BED.

⑤ **end:** Genomic end of the feature, with a 1-base offset. This is the same end coordinate as it is in 0-offset half-open sequence formats, like BED.

⑥ **score:** Numeric value that generally indicates the confidence of the source in the annotated feature. A value of "." (a dot) is used to define a null value.

⑦ **strand:** Single character that indicates the strand of the feature; it can assume the values of "+" (positive, or 5' → 3'), "-" (negative, or 3' → 5') or "." (undetermined).

⑧ **phase:** Phase of CDS features; it can be either 0, 1 or 2 (for CDS features) or "." (for everything else).

⑨ **attributes:** All the other information pertaining to this feature. The format, structure and content of this field is the one which varies the most between the three competing file formats.

Examples

spikeDomains.gff

```
##gff-version 3
Sequence        .       polypeptide     1       1270    .       +       .
   ID=Sequence;md5=9d13bab97d5cfb8f3b918fa30280375c
Sequence        Phobius protein_match   1       13      .       +       .
  date=27-05-2024;Target=Sequence 1 13;ID=match$1_1_13;signature_desc=Signal
peptide region;Name=SIGNAL_PEPTIDE;status=T
Sequence        Pfam    protein_match   345     523     1.2E-34 +       .
  date=27-05-2024;Target=Sequence 345
523;ID=match$2_345_523;signature_desc=Betacoronavirus spike glycoprotein S1,
receptor
binding;Name=PF09408;status=T;Dbxref="InterPro:IPR018548","Reactome:R-HSA-
9678110","Reactome:R-HSA-9679509","Reactome:R-HSA-9683686","Reactome:R-HSA-
9683701","Reactome:R-HSA-9692916","Reactome:R-HSA-9694322","Reactome:R-HSA-
9694548","Reactome:R-HSA-9694614","Reactome:R-HSA-9694635","Reactome:R-HSA-
9705671","Reactome:R-HSA-9733458"
```

4.2 Visualization

4.3 Exercises

① **Total feature count.** Write a command to count the total number of features (e.g., genes, exons) in a GFF file.

② **Feature types.** List all unique feature types (e.g., gene, exon) present in the GFF file. How many of each feature are there?

5 BED

5.1 The format

BED stands for **B**rowser **E**xtensible **D**ata.

BED files have three required fields and nine additional optional fields.

The three required columns are:

① **chrom:** Chromosome (e.g., chr1, chrY, chr2_random) or scaffold (e.g., scaffold10, scaffold10671) name.

② **chromStart:** Starting position (coordinate) on the chromosome or scaffold for the sequence (feature) considered. The first base on the chromosome is numbered 0.

③ **chromEnd:** Ending position (coordinate) on the chromosome or scaffold for the sequence (feature) considered. This position is non-inclusive, unlike `chromStart`.

The nine additional optional columns are:

4. **name:** Name of the line in the BED file.

5. **score:** Score between 0 and 1000.

6. **strand:** DNA strand orientation. It can be "+" (positive), "-" (negative) or "." (if the strand doesn't matter).
7. **thickStart:** Starting coordinate from which the annotation is displayed in a thicker way on a graphical representation (e.g,: the start codon of a gene).
8. **thickEnd:** End coordinate from which the annotation is no longer displayed in a thicker way on a graphical representation (e.g.: the stop codon of a gene).
9. **itemRgb:** RGB value in the form R,G,B (e.g.: 255,0,0) determining the display color of the annotation contained in the BED file.
10. **blockCount:** Number of blocks (e.g., exons) on the line of the BED file.
11. **blockSizes:** List of values separated by commas corresponding to the size of the blocks (the number of values must correspond to that of the `blockCount`).
12. **blockStarts:** List of values separated by commas corresponding to the starting coordinates of the blocks, coordinates calculated relative to those present in the `chromStart` column (the number of values must correspond to that of the `blockCount`).

The main reason a BED file is attractive for use is its simplicity in just giving you the start, the stop and the chromosome that each feature is encoded

0-base vs 1-base sequence coordinates

1-base	1	2	**3**	4	5	**6**	7	8	9
	C	C	**T**	**A**	**A**	**C**	C	C	T
0-base	0	1	**2**	3	4	5	**6**	7	8

1-base: Inclusive start and inclusive end.

0-base: Inclusive start and exclusive end.

Convert 0-base to 1-base:

- start1 = start0 + 1.
- end1 = end0.

BEDTools suite

Swiss-army knife of tools for a wide-range of genomic analysis tasks.

Toolkit that enables genomic arithmetic, i.e., set theory on the genome.

Can be used on multiple files in widely-used genomic file formats: BED, BAM, GFF/GTF and VCF.

Sintax: `bedtools sub-command [OPTIONS] <FILE>`

Sub-command examples: sort, intersect, closest, window, getfasta, random, shuffle, slop.

Sorting regions in a BED file: `bedtools sort [OPTIONS] -i <BED/GFF>`

```
$> cat A.bed
chr1 800 1000
chr1 80 180
chr1 1 10
chr1 750 10000

$> bedtools sort -i A.bed
chr1 1 10
chr1 80 180
chr1 750 10000
chr1 800 1000
```

Finding overlapping regions with BEDTools:

```
$> bedtools intersect [OPTIONS] -a <FILE> -b <FILE1, FILE2, ..., FILEN>
```

Identifying regions within a window:

```
$> bedtools window [OPTIONS] [-a | -abam] -b <BED/GFF/VCF>
```

Extension The `bed` extension is the most common. The number of columns sometimes is noted in the file extension: `bed3`, `bed4`, `bed6`, `bed12`.

Examples

dummy.bed3

```
chr1 14 200
chrY 140 2000
scaffold10 41 42
```

Types

BED3

BED5

BED6

BED9

BED12

BEDn+m

5.2 Visualization

5.3 Exercises