Linux command line
Part II

Input/Output redirections and text processing

Standard Input/Output

- Standard Input (*StdIn*) – something we print on screen with a keyboard
- Standard Output (*StdOut*) – something that the software prints to screen
- Standard error (*StdErr*) – a status message (usually an error message) that a program prints to screen

Change to ***sandbox/*** directory

echo command takes some keyboard input as an argument and prints it to screen

Let's use echo to print something to screen
$ echo Hello world! # will print "Hello world!" to *StdOut*

We can use **>** to redirect *StdOut* to file
$ echo Hello world! > hello_world.txt # ">" sign will redirect the output of the command to a file, it will create a new file if it does not exist
$ cat hello_world.txt # check the file content

Standard Input/Output

Redirecting output with **>** will completely overwrite the file with no warning and should be used with great care!

```
$ echo Good night world! > hello_world.txt
$ cat hello_world.txt # what is the content of the file now?
```

How can we add content to an existing file? Use **>>** !
Let's create a new file *movies.txt* and add 3 entries through redirect

```
$ echo Pulp fiction > movies.txt
$ echo Kill Bill >> movies.txt
$ echo Deathproof >> movies.txt
$ cat movies.txt
```

Using **>>** on non-existing file will result in a creation of a new file same with **>**.

```
$ echo Lord of the Rings >> books.txt
$ cat books.txt
```

Standard Input/Output

What happened is the software generates and error message with file a file re-direct?

$ ls -l ../docs > ls_out.txt

$ cat ls_out.txt # *ls_out.txt* file is empty, **ls** generates an error message since ***docs/*** directory does not exist. The error message is printed to screen through *StdErr* while standard input would be empty

How can we write *StdErr* to a file

$ ls -l ../doc 2> ls_out.txt # the file descriptor ***2*** is **>** to re-direct *StdErr*

$ cat ls_out.txt

We can append the error message to file in the same fashion as previously

$ ls -l ../doc 2>> ls_out.txt

$ cat ls_out.txt

We can also re-direct *StdOut* and *StdErr* to file at the same time

$ ls -l ../ ../docs > ls_out.txt 2>&1 # use ***2>&1*** construct

$ cat ls_out.txt

$ rm ls_out.txt

$ ls -l ../ ../docs &> ls_out.txt # we can also use **&>** on newer systems

Text processing

Key text processing commands:

- **cat** – concatenate files
- **sort** – sort lines of text
- **uniq** – report or omit supplicated files
- **grep** – print lines matching a pattern
- **wc** – count lines, words, characters, bytes in the file
- **cut** – select columns in the file
- **tr –** translates or deleted characters in text files
- **sed** – search, replace, insertion and deletion in text files. Can edit without opening the files

Using these commands provides a quick way to process and analyze data files used in bioinformatics.

Text processing: **cat**

Let's download some practice files containing M. tuberculosis protein sequences

We will do our practice in ***sandbox/*** directory

$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/mtub_prot1.fasta
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/mtub_prot2.fasta
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/mtub_prot3.fasta

Let's clear the clutter and remove all files in ***sandbox/*** apart from *.fasta* files we just download

Remove all of the files exept for *.fasta* using wildcards
$ ls -l !(*.fasta) # we will use **ls** test our wildcard before removing the files
$ rm -v !(*.fasta) # remove all files that are not *.fasta* with 'verbose' option
$ ls -l # check the results

We can use **cat** to print content of the file to screen
$ cat mtub_prot1.fasta
$ cat *.fasta # we can also print multiple files

Text processing: **cat**

Use re-direct to merge multiple sequence file into a single file.

This is a frequent task in day-to-day work.
$ cat mtub_prot*.fasta > merged.fasta
$ cat merged.fasta # we should have all 3 sequences in the same file

We can also use **cat** to create text files form keyboard input
$ cat # the command will wait for keyboard input
$ >seq1 # lets create tiny fasta sequence
$ AAACTG # press CTRL+D to tell that it reached the end of file (EOF)

Now, let's create a little fasta file with **cat**
$ cat > seq1.fasta # this will create a new file which we can fill from keyboard
$ >seq1
$ AAACTG # press CTRL+D combo
$ cat seq1.fasta # check the results

Text processing: **wc**

Count lines lines, words, characters using **wc**

```
$ wc merged.fasta
 12  30 741 merged.fasta
```

By default **wc** prints a number of lines, words and characters followed by path to file

Count lines only

```
$ wc -l merged.fasta
```

Count words

```
$ wc -w merged.fasta
```

Count characters

```
$ wc -c merged.fasta
```

Get the size of a file in bytes

```
$ wc --bytes merged.fasta
```

Text processing: **sort**

Let’s try numeric sorting

Create practice file *numerals.txt*
$ cat > numerals.txt
$ 10
$ 1
$ 75
$ 234
$ 400
Press CTRL+D

$ cat numerals.txt # check the results

Sort the file alphabetically
$ sort numerals.txt # not we would expect
$ sort -n numerals.txt # we must use **-n** option to enable numeric sorting
$ sort -rn numerals.txt # sort numerically and in reverse

Text processing: **sort**

Sorting a text by columns with **sort** and selecting columns with **cut**

Download practice comma separated file:
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/gene_exprs_result.csv
$ head head -n 3 gene_exprs_result.csv

Side note: most of the files used in NGS analysis are human readable text files and a few of those are organized into rows and columns – **delimited** files.

The most common delimiters are:
-> tab – "\t" in command line
-> comma – ","
-> space – " "
-> column – ":"
-> pipe – "|"

However, any character can be specified to separate the values

We need to know which separator is used analyze columns in the file

Text processing: **sort and cut**

Sorting a text by columns with **sort** and selecting columns with **cut**

$ head -n 3 gene_exprs_result.csv # examine the file, this a comma separated file

Select “padj” (adjusted p-values) with **cut**
$ cut -b 1,2,3 gene_exprs_result.csv # extract first 3 bytes from every line of the file. This will extract first 3 characters from the first column in gene expression results file.

You can specify bytes with ranges
$ cut -b 1-15 gene_exprs_result.csv # this will extract ensemble identifiers – ENSG followed by 11 stable and unique numbers

Ranges can be comma separated
$ cut -b 1-15,16-20 gene_exprs_result.csv # this will retrieve **gene ID** column and a part of **base mean** column

We can specify similar syntaxis in case of characters
$ cut -c 1-15 gene_exprs_result.csv # prints fixed length column – **gene ID**

Text processing: **sort and cut**

Sorting a text by columns with **sort** and selecting columns with **cut**

Print complete lines

```
$ cut -c 1- gene_exprs_result.csv
```

Print from starting position to a specified character

```
$ cut -c -15 gene_exprs_result.csv
```

Extract columns by a combination of **-d** (delimiter) and **-f** (field)

```
$ cut -d ',' -f 1 gene_exprs_result.csv # will print the first column
```

Extract columns by a combination of **-d** (delimiter) and **-f** (field)

```
$ cut -d ',' -f 7 gene_exprs_result.csv # will print column number 7 – adjusted p-values column
```

Extract columns by a combination of **-d** (delimiter) and **-f** (field)

```
$ cut -d ',' -f  1-7 gene_exprs_result.csv # will print a range of columns from 1 to 7
```

Text processing: **sort and cut**

Sorting a text by columns with **sort** and selecting columns with **cut**

Let's practice sorting using a small tab delimited file
$ cat sample_tab_delimited.txt # view the file first
$ sort sample_tab_delimited.txt # without any options it will sort alphabetically by line

Sort a file by a single column
$ sort -k 3,3 -n sample_tab_delimited.txt # **-k** specifies column number, by default -k accepts a range of columns **-k Start,End**, so **-k 3** means column 3 to the end of the line, and **-k 3,5** means columns 3 to 5. To sort by a single column, we must specify column number a range, i.e **-k 3,3**

Numeric sort by second column
$ sort -k2,2n sample_tab_delimited.txt
$ cat sample_tab_delimited.txt # sorting had no impact on column 1 that contains gene IDs

Spot and remove duplicated column entries
$ sort -k1,1 -u sample_tab_delimited.txt # **-u** option keeps unique entries only

Text processing: **uniq** and **pipe**

Using **uniq** to report and filter duplicated lines and starting with **pipes**

The power of the command line shines through the **pipe** (|)

'|' allows to use an output of a command as an input into another command

command 1 | command 2 | …. | command n

Let's download a practice file with duplicate entries
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/zoo.txt

We have a list of animals in *zoo.txt* some of them are listed more than once. Let's reduce the list of animals to unique entries only.
$ sort zoo.txt | uniq # **uniq** must be preceded by **sort**

Count a number of animals in the animal list
$ sort zoo.txt | uniq -c # **-c** option will count the number of times an item is observed in the list

Let's count and sort by occurrences
$ sort zoo.txt | uniq -c | sort -k1,1nr # here , we sorted the output by the first column

Text processing: **uniq** and **pipe**

Using **uniq** to report and filter duplicated lines and starting with **pipes**

- Using pipes brings us to the idea of “one-liner” approach

- Command line allows us parse, manipulate, and summarize files using tiny pipelines – scripts that fit into a single line

Let’s try to manipulate a more complex file
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/f523b8f2b8aaa071a9124107642b7fcfbeedff53/GCF_000195955.2_ASM19595v2_genomic.gff

Take a look at the file
$ head -n 10 GCF_000195955.2_ASM19595v2_genomic.gff # This is a tab delimited file containing gene annotations for M. tuberculosis genome. Lines starting with **#** do not belong to the body of the file

We want to skip first 7 lines that start with **#**
$ tail -n +7 GCF_000195955.2_ASM19595v2_genomic.gff | head –n 3 # check this command, it does almost what we need it to do, however, one lines starting with # is still remaining

Text processing: **uniq** and **pipe**

Using **uniq** to report and filter duplicated lines and starting with **pipes**

To skip first N lines, we need to specify N+1 as **tail** options

```
$ tail -n +8 GCF_000195955.2_ASM19595v2_genomic.gff | less
```
we can use **less** to test each step of the one-liner

The types of genomic features included in the *gff* annotation file are located in column 3

```
$ tail -n +8 GCF_000195955.2_ASM19595v2_genomic.gff | cut -f 3 | less
```

Summarize the types of genomic features contained in the *gff* file

```
$ tail -n +8 GCF_000195955.2_ASM19595v2_genomic.gff | cut -f 3 | sort | uniq -c
```

Let's also sort the output by the number of features

```
$ tail -n +8 GCF_000195955.2_ASM19595v2_genomic.gff | cut -f 3 | sort | uniq -c |  sort -k1,1rn
```

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

- **grep** searches a file for patterns and reports matching lines
- Finding matching patterns in text is very common task both in bioinformatics and in Linux administration
- **grep** uses regular expressions to match patterns in text, it can also match fixed strings

Let's practice **grep** on *divine_comedy.txt* file to current directory ***sandbox/***
$ cp ../divine_comedy.txt .

Search Divine comedy for a word "so"
$ grep so divine_comedy.txt # search "Divine comedy" for a so
$ grep --color=auto so divine_comedy.txt # it is much better to use **autocolor** option. This command will search for regexp pattern **so**, not a fixed string, and the search will be case-sensitive
$ grep -i --color=auto so divine_comedy.txt # **-i** makes the search case-insensitive

Count the number of lines where a match occurs
$ grep --color="auto" "heart" divine_comedy.txt | wc -l

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

Print number of a line where a match was encountered

```
$ grep -n --color="auto" "heart" divine_comedy.txt
```

Using **-c** option to count number of lines where a match occurred

```
$ grep -c --color="auto" “so” divine_comedy.txt
```

this will count a number of lines with a match, not a number of matches

Sometimes it is useful to get a number of matches as opposed to a number of lines with a match

```
$ grep -o --color="auto" "so" divine_comedy.txt
```

-o option will output matches

Now we can use **wc** to count matches

```
$ grep -o --color="auto" "so" divine_comedy.txt | wc -l
```

Figure out how many times the match occurs in each of the lines

```
$ grep -n -o "so" divine_comedy.txt | cut -d ':' -f 1 | sort -n | uniq -c
```

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

We can search multiple files by listing them one-by-one or using wild cards

```
$ grep --color="auto" "dolphin" animals.txt zoo.txt
$ grep --color="auto" "dolphin" *.txt
$ grep -r --color="auto" "dolphin" . # search a directory recursively
```

Match whole words as opposed to partial pattern matches, compare the results of the commands below

```
$ grep  --color="auto" "so" divine_comedy.txt
$ grep -w --color="auto" "so" divine_comedy.txt
```

Search for a sub-sequence in a genome

Let's download M. tuberculosis genome first

wget https://raw.githubusercontent.com/slavailn/bioinf_training/f523b8f2b8aaa071a9124107642b7fcfbeedff53/GCF_000195955.2_ASM19595v2_genomic.fna

```
$ grep --color="auto" "TATA" GCF_000195955.2_ASM19595v2_genomic.fna # note, how fast grep
```

completes the search!

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

We can use regular expressions, or *regexp,* to come up with complex matching patterns.

$ grep --color="auto" "C.TATA.G" GCF_000195955.2_ASM19595v2_genomic.fna # "." is a special symbol that will match any single character

$ grep --color="auto" "C..TATA..G" GCF_000195955.2_ASM19595v2_genomic.fna # This will match C followed by any 2 characters followed by any two characters and followed by G

$ grep -F --color="auto" "C..TATA..G" GCF_000195955.2_ASM19595v2_genomic.fna # **-F** disables regular expressions and forces a literal match

$ grep --color="auto" "^#" GCF_000195955.2_ASM19595v2_genomic.gff # **^** is an anchor that matches a pattern at the beginning of the line

$ grep -v --color="auto" "^#" GCF_000195955.2_ASM19595v2_genomic.gff # reverse the match to filter the file and keep only the lines with annotation in the *gff* files

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

Let's download another practice file
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/mirna_mm10.fasta # this file contains mouse miRNA sequences in fasta format
$ head mirna_mm10.fasta

Show all sequences ending with CC
$ grep -B 1 --color="auto" "CC$" mirna_mm10.fasta # **$** will match a pattern at the end of a line, **-B** option with print one line before the match

$ grep -B 1 --color="auto" "^A[ACTG]*CC$" mirna_mm10.fasta # This will match all lines start with A and is followed by any number A,C,T, or G characters. Finally, the line must end with CC

Let's unpack some new special characters
[] – matches any of the symbols put in brackets, for example [CG] with match either C or G; [A-Za-z] will match all upper-case or any lower-case letters

***** - match previous pattern zero or more times

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

Match more than one pattern with *POSIX Extended Regular Expressions* (ERE)
$ grep -E --color="auto" "AACCT|GGTGTG" GCF_000195955.2_ASM19595v2_genomic.fna # ERE use is invoked with **–E** option

Sometimes we need to match special characters. Use **** to *escape* special characters
$ grep --color="auto" "\." divine_comedy.txt # This will match a '.' literally as opposed to interpreting it as *any character*

We can match double-quotes buy enclosing them in single quotes
$ grep -E --color="auto" '"|"' divine_comedy.txt # matches double quotes symbols

We can use quantifiers to specify the exact number of pattern matches, this option is available with ERE
$ grep -E --color="auto" "C{3}$" mirna_mm10.fasta # Use **{}** to match a pattern a certain number of times. Here, we will match all lines that end with **CCC**

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

We can use **grep** to search for multiple patterns stored in a file

Let's create a file with multiple patterns

```
$ echo "miR-7" > multi_pattern.txt
$ echo "miR-46" >> multi_pattern.txt
$ echo "miR-23" >> multi_pattern.txt
$ grep -A 1 --color="auto" -f multi_pattern.txt mirna_mm10.fasta # This command with match patterns
```

stored in *multi_pattern.txt* file specified with **-f** option

We can use this functionality to exclude a list of matching lines from a file

The gene sannotation file GCF_000195955.2_ASM19595v2_genomic.gff contains pseudogenes and tRNA.

```
$ grep --color="auto" "tRNA"  GCF_000195955.2_ASM19595v2_genomic.gff
$ grep --color="auto" "pseudogene"  GCF_000195955.2_ASM19595v2_genomic.gff
$ echo tRNA > multi_pattern.txt
$ echo pseudogene >> multi_pattern.txt
```

Text processing: **grep**

Using **grep** to extract lines from the file that match (or do not match) a pattern

We can use **grep** to search for multiple patterns stored in a file

```
$ grep --color="auto" -f multi_pattern.txt GCF_000195955.2_ASM19595v2_genomic.gff |  grep -E --color="auto" "pseudogene|tRNA" # check if we are getting right matches

$ grep -v --color="auto" -f multi_pattern.txt GCF_000195955.2_ASM19595v2_genomic.gff | grep -E --color="auto" "pseudogene|tRNA" # check if we still observe those matches
```

Now we can save the results to a new file

```
$ grep -v --color="auto" -f multi_pattern.txt GCF_000195955.2_ASM19595v2_genomic.gff > annotation_filtered.gff
```

How many lines were removed

```
$ wc -l GCF_000195955.2_ASM19595v2_genomic.gff
$ wc -l annotation_filtered.gff
```

Text processing: **grep**

Regular expressions cheat sheet

Regular expression	Meaning
^	Anchor: start of the line
$	Anchor: end of the line
*	Matches 0 or more characters
.	Match any single character
[]	Matches any of the characters placed in brackets, accepts ranges, i.e. [A-Z] will match all upper-case alphabetical characters
[^]	Matches any characters except those included in [], for example [^C] will match any character except for C
\	Escapes special characters
pattern1|pattern2	Searches for pattern1 OR pattern2
{}	Matches an exact number of times stated in {}, for example C{3} will match CCC pattern

Learn more: https://caspar.bgsu.edu/~courses/Stats/Labs/Handouts/grepsearch.htm

Text processing: **tr**

Using **tr** to replace or delete characters

tr (translate) can find and replace pattern in text.

Syntax: tr [options] SET1 SET2