# UNIX Exercise 2

Second exercise on UNIX deals with more complex commands with their useful options and using multiple commands at a time. Make sure you understand all the commands from the previous exercise as you will be using them frequently in this exercise**.**

## FASTA format:

FASTA format is nothing but a simple text file containing either nucleotide sequences or protein sequences. An individual sequence always starts with a single line description of the sequence, followed by lines of sequence data. Description can be just an identification number or even blank (not recommended) but should always begin with a greater-than (>) symbol. The sequence is considered to be complete if another line starting with a > is encountered. The simplicity of FASTA format makes it easy to manipulate and parse sequences using text-processing tools and any scripting languages like Python, Perl or Ruby.

Some example FASTA format protein sequences are given below:

>gi|18403023|ref|NP\_565747.1| splicing factor 3A subunit 2 [Arabidopsis thaliana]

MDREWGSKPGSGGAASGQNEAIDRRERLRRLALETIDLAKDPYFMRNHLGSYECKLCLTLHNNEGNYLAH

TQGKRHQTNLAKRAAREAKDAPTKPQPLKRNVSVRRTVKIGRPGYRVTKQYDPELQQRSLLFQIEYPEIE

DNIKPRHRFMSSYEQKVQPYDKSYQYLLFAAEPYEIIAFKVPSTEVDKSTPKFFSHWDPDSKMFTLQVYF

KPTKPEPNKPQSAVGANGLPPPPPPPPHQAQPPPPPPSGLFPPPPPPMANNGFRPMPPAGGFGHPNM

>gi|224140247|ref|XP\_002323495.1| predicted protein [Populus trichocarpa]

MDREWGSKPGSGGAASAQNEAIDRRERLRRLALETIDLAKDPYFMRNHLGSYECKLCLTLHNNEGNYLAH

TQGKRHQTNLAKRAAREAKDAPALPQPNKRKVNIRKTVKIGRPGYRVTKQFDPETKQRSLLFQIEYPEIE

DNTKPRHRFMSSYEQRIEANDKRFQYLLFSAEPYEIIAFKVPSTEIDKSTPKFFSHWDPDSKMFTLQLYF

KLKPPEANKPQSVAAANSTVPSQPPPPLPPQGLPAGSRPPPPPMPASLPPPPPPAMANGPRPMPPGGAPP

APPPPPGGSGAMVNFTPGTQAGRPSSMLPPHGFLGQQMQGQTIRPPLLPPNMGQ

## **Pipes and REDIRECTS**

Many UNIX commands use some input file/data and display the output on the screen. This is feasible when the data being displayed is small enough to fit the screen or if it is the endpoint of your analysis. But for large data outputs, it is efficient to redirect to a file instead of screen. This can be done very easily in UNIX using > (greater than) or < (lesser than) or >> signs.

* < redirects the data to the command for processing
* > redirects the data from the command's output to a file. The file will be created if it is non-existing and if present it will overwrite the contents with the new output data (you will lose the original file).
* >> unlike > this redirection lets user append the data to an already existing file or a new file
* Another special operator | (called pipe) is used sometimes to pass the output from a command to another command (as input) before sending it to an output file or display.

Some *eg.*

cat FILE1 > FILE2

Creates a new file, file2 with same contents as old file, file1

cat FILE1 >> FILE2

Appends the contents for file1 to file2, equivalent to opening file1, copying all the contents, pasting the copied contents to the end of the file2 and saving it!

cat FILE1 | less

Here, cat command displays the contents of the file1, but instead of sending it to standard output (screen) it sends it through the pipe to the next command 'less' so that contents of the file are now displayed on the screen with line scrolling.

**Task 2.1: The Sequences directory contains a number of files and each of these files contain a single FASTA formatted nucleotide sequence. Combine them all together to make a single file sequences.fasta using redirects.**

cat \*.fa >> sequences.fasta

this command will combine all .fa files into one.

## Regular Expressions

When working with the sequences (protein or DNA) we are often interested to see if a particular feature is present or not. This could be various things like a start codon, restriction site or even a motif. In UNIX all strings of text that follow some pattern can be searched using some formula called regular expressions. *eg.* If you are looking for a particular motif in large number of sequences, then you can create a regular expression in UNIX and search all the sequences having that motif relatively easily. Regular expression consists of normal and metacharacters. Commonly used characters include

|  |  |
| --- | --- |
| Expression | Function |
| . | matches any single character |
| $ | matches the end of a line |
| ^ | matches the beginning of a line |
| \* | matches one or more character |
| \ | quoting character, treat the next character followed by this as an ordinary character. |
| [] | matches one or more characters between the brackets |
| [range] | match any character in the range |
| [^range] | match any character except those in the range |
| \{N\} | match N occurrences of the character preceding (sometimes simply +N) where N is a number. |
| \{N1,N2\} | match at least N1 occurrences of the character preceding but not more than N1 |
| ? | match 1 occurrence of the character preceding |
| | | match 2 conditions together, \(this**\**|that)\ *matches both this or that in the text* |

**For complete list, type info** regex **on your terminal.**

**Some examples related to nucleotide/protein sequences:**

|  |  |
| --- | --- |
| **Patterns** | **Matches** |
| ^ATG | **Find a pattern starting with ATG** |
| TAG$ | **Find a pattern ending with TAG** |
| ^A[TGC]G | **Find patterns matching either ATG, AGG or ACG** |
| TA[GA]$ | **Find patterns matching either TAG or TAA** |
| ^A[TGC]G\*TGTGAACT\*TA[GA]$ | **Find gene containing a specific motif** |
| [YXN][MPR]\_[0-9]\{4,9\} | **Find patterns matching NCBI RefSeq (*eg* XM\_012345)** |
| \(NP\|XP\)\_[0-9]\{4,9\} | **Find patterns matching NCBI RefSeq proteins** |

**Some common commands that can be used to manipulate text using regular expressions are** grep **(filters input against a pattern),** sed **(applies transformation after searching a pattern) and** awk **(manipulates data arranged in columns). We will discuss these commands in detail**

## **grep**

grep (*g*lobally search a *r*egular *e*xpression and *p*rint) is one of the most useful commands in UNIX and it is commonly used to filter a file/input, line by line, against a pattern e*g.,* to print each line of a file which contains a match for pattern.

grep PATTERN FILENAME

Like any other command there are various options available for this command. Most useful options include:

-v inverts the match or finds lines NOT containing the pattern.

--color colors the matched text for easy visualization

-F interprets the pattern as a literal string.

-H, -h print, don't print the matched filename

-i ignore case for the pattern matching.

-l lists the file names containing the pattern (instead of match).

-n prints the line number containing the pattern (instead of match).

-c counts the number of matches for a pattern

-w forces the pattern to match an entire word.

-x forces patterns to match the whole line.

With options, syntax is

grep [OPTIONS] PATTERN FILENAME

Some typical scenarios to use grep:

* Counting number of sequences in a multi-fasta sequence file
* Get the header lines of fasta sequence file
* Find a matching motif in a sequence file
* Find restriction sites in sequence(s)
* Get all the Gene IDs from a multi-fasta sequence files and many more.

Now let's use grep command to do some simple jobs with the sequences:

Counting sequences: By FASTA format definition, we know that number of sequences in a file should be equal to the number of description lines. So by counting > in file, you can count the number of sequences. This can be done using counting option of the grep (-c).

grep -c ">" FILENAME

**Task 2.3: Count the number of sequences** AT\_cDNA.fa **and** RefSeq.faa

grep -c ">" AT\_cDNA.fa \_\_\_\_\_\_\_\_\_

grep -c ">" RefSeq.faa \_\_\_\_\_\_\_\_\_

If you are looking for information about the sequences, you can list all the headers (description lines) for the sequences using grep. Simply search for ">" and grep will list all the description lines.

grep ">" FILENAME

grep ">" AT\_cDNA.fa

Alternatively, you can send it to a file if you want to use it later or you can just pipe it to less or more command to scroll through it line by line or page by page.

grep ">" FILENAME > HEADERFILE.txt

grep ">" FILENAME | less

grep ">" AT\_cDNA.fa | less

Use up/down arrow keys to move up and down, press q to exit

*See what kind of sequences are in* AT\_cDNA.fa file*. Do they all seem to belong to same organism? yes/no Which organism? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Using grep you can also locate all the lines that contain a specific term you are looking for. This is very useful especially to look for a specific gene among a large number of annotated sequences.

grep "word or phrase to search" FILENAME

**Task 2.4: Try searching for your favorite gene, to see if it is present in** AT\_cDNA.fa**(this file contains all annotated sequences for *Arabidopsis thaliana*). Unlike Google or any search engines, only exact search terms will be identified, but you can ask** grep **to ignore cases while searching using** -i **option. Try these:**

grep -i "transcription factor" AT\_cDNA.fa

grep -i "TFIIIA" AT\_cDNA.fa

You can also use this feature to see if your sequence of interest has a specific feature (restriction site, motif etc.,) or not. This can be performed better using --color option of the grep.

**Go to the sequences directory, search for *Eco*R1 (GAATTC) site in the** NT21.fa **file, and use the color option. Also, try looking for a C2H2 zinc finger motif in** RefSeq.faa **file (for simplicity let's assume zinc finger motif to be** CXXXCXXXXXXXXXXHXXXH**. Either you can use dots to represent any amino acids or use complex regular expressions to come up with a more representative pattern. Try these:**

grep --color "GAATTC" ./Sequences/NT21.fa

grep --color "C..C............H...H" RefSeq.faa

(2) (12) (3)

You can also use grep command to exclude the results containing your search term. Say if you want to look at genes that are not located in chromosome 1, you can exclude it form your search by specifying -v option.

grep -i "transcription factor" AT\_cDNA.fa| grep -v "chr1"

grep -i "transcription factor" AT\_cDNA.fa| grep "chr1"

*Notice the difference in output from the above two commands.*

Try to understand the following command lines (and record your results, where applicable):

grep -c -w "ATP" RefSeq.faa \_\_\_\_\_\_\_\_\_\_\_\_\_

grep -c CGT[CA]GTG AT\_cDNA.fa \_\_\_\_\_\_\_\_\_\_\_\_\_

grep -l "ATG" ./sequences/\*.fa

**You can also try some regular expressions related to nucleotide/protein sequences provided earlier to see how it works.**

## **SED**

The sed command is a *s*tream *ed*itor that reads one or more text files, makes changes or edits according to editing script, and writes the results to standard output. Most common editing script sed uses is to substitute text matching a pattern. The simple syntax for using sed is as follows

sed 'OPERATION/REGEXP/REPLACEMENT/FLAGS' FILENAME

Here, / is the delimiter (you can also use \_ (underscore), | (pipe) or : (colon) as delimiter as well)

OPERATION specifies the action to be performed (sometimes if a condition is satisfied). The most common and widely used operation is s which does the substitution operation (also useful operator is y which does transformation).

REGEXP and REPLACEMENT specify search term and the substitution term respectively for the operation that is being performed.

FLAGS are additional parameters that control the operation. Some common FLAGS include:

g replace all the instances of REGEXP with REPLACEMENT (globally)

n (n=any number) replace nth instance of the REGEXP with REPLACEMENT

p If substitution was made, then prints the new pattern space

i ignores case for matching REGEXP

w file If substitution was made, write out the result to the given file

d when specified without REPLACEMENT, deletes the found REGEXP

For brevity we only discuss sed command with respect to search and replace function. To do other things please refer to the man page of sed or the reference provided here <http://www.grymoire.com/Unix/Sed.html#uh-47>.

Some search and replace examples:

sed 's/chr/chromosome/g' FILENAME replaces ALL instances in a line

sed '/MTF1/s/chr/chromosome/g' FILENAME

replaces all instances in a line only if it contains 'MTF1'

Other common tasks that can be performed using sed

|  |  |
| --- | --- |
| sed -n '52p' FILENAME | Prints 52nd line |
| sed -n '8,12p' FILENAME | Prints line 8 through 12 |
| sed -n '1,2~2p' FILENAME.fastq | Prints 2nd and every 4th line (header and sequence from a FASTQ file) |
| sed "1d" FILENAME | Delete 1st line |
| sed "1,3d" FILENAME | Delelte line 1, 2 and 3 |
| sed 's/^$//g' FILENAME | Delete balnk lines |
| sed '2 i line to insert' FILENAME; | insert "line to insert" on second line |

**Task 2.5: Try using replace function on** AT\_genes.gff **file (to change** Chr **to** Chromosome**). View both files to see the difference.**

sed 's/^Chr/Chromosome\_/g' AT\_genes.gff > AT\_genes\_converted.gff

## AWK

Unlike other UNIX commands awk is a structured language by itself. awk stands for the names of its authors *A*ho, *W*einberger, and *K*ernighan. Many bioinformatics programs generate rows and columns of information. awk is an excellent tool for processing these rows and columns, and it is easier than most conventional programming languages.

The typical syntax for awk is:

awk 'PATTERN {ACTION}' FILENAME

awk then works by reading the input file one line at a time, matching the given PATTERN and performing the corresponding ACTION for the matches. If there is no PATTERN, then the ACTION will be performed on each line. But if there is no ACTION then the default ACTION (printing all lines) on the matching PATTERN will be performed (empty braces {} without any ACTION turns off default printing).

A simplest *eg*. would be

awk '{print}' FILENAME *try* awk '{print}' AT\_genes.gff

Here, since there is no PATTERN, the print ACTION will be performed on each line (equivalent to cat INFILE).

Some inbuilt variables of awk include:

|  |  |
| --- | --- |
| FS | Field Separator (default SPACE) |
| OFS | Output Field Separator (default SPACE) |
| NF | Number of Fields in the input |
| NR | Number of Records (lines) in the input |
| RS | Record Separator (default NEWLINE) |
| ORS | Output Record Separator (default NEWLINE) |
| FNR | File line number |
| N | Nth field of the line where N can be any number (eg. $0 = entire line, $1 = First field, so on) |

awk accepts all standard patterns (regular expression and expression) plus some special patterns

|  |  |
| --- | --- |
| BEGIN | Special PATTERN that is executed before the INPUT is read |
| END | Special PATTERN that is executed after the INPUT is read |
| empty | nonexistent PATTERN that matches every input record |

Some simple *eg.* using awk (you can try these commands with AT\_genes.gff FILE)

|  |  |
| --- | --- |
| awk NF FILE | Deletes all blank lines |
| awk 'NF > 0' FILE | Deletes all blank lines |
| awk 'NF > 4' FILE | Prints only lines with more than 4 fields |
| awk '$NF > 4' FILE | Prints only lines with value of the 4th filed > 4 |
| awk 'END { print $NF }' FILE | Prints value of the last field of the last line |
| awk 'NR==25,NR==100' FILE | Prints lines between 25 and 100 |
| awk 'NR==50' FILE | Prints 50th line of input |
| awk 'NR < 26' FILE | Prints first 25 lines |
| awk 'NR > 25' FILE | Prints file after 25th line |
| awk 'END { print NR }' FILE | Prints the last line of the file |
| awk '{ print NF ":" $0 }' FILE | Prints number of fields in front of every line |
| awk '{ print FNR ":" $0 }' FILE | Prints line number in front of every line |
| awk '$5 == "abc123"' FILE | Prints lines which have 'abc123' in 5th field |
| awk 'BEGIN { ORS="\n\n" }; 1' FILE | Double spaces the file |
| awk '{ print $1, $2 }' FILE | Prints only 1st and 2nd field |
| awk '{ print $2, $1 }' FILE | Prints only 2nd and 1st field (swapping columns) |
| awk '{ $2 = ""; print }' FILE | Prints the file without 2nd column |
| awk '/REGEX/' FILE | Prints all the lines having REGEX |
| awk '!/REGEX/' FILE | Prints all the lines not having the REGEX |
| awk '/AAA|BBB|CCC/' FILE | Prints all the lines having either AAA, BBB or CCC |
| awk 'length > 50' FILE | Prints line having more than 50 characters |
| awk '/POINTA/,/POINTB/' FILE | Prints everything between POINTA and POINTB |

Try to understand the following command lines (and record it):

|  |  |
| --- | --- |
| awk 'END { print $NF }' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk 'NR==30,NR==35' AT\_genes.gff. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk 'NR==25' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk 'NR<25' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk 'END { print NR }' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '{ print NF ":" $0 }' AT\_genes.gff > with\_fields.txt | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '{ print NR ":" $0 }' AT\_genes.gff > with\_Line\_num.txt | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '{ print $1, $3 }' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '{print $1"\t"$3"\t"$2}' AT\_genes.gff | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '{print $1,$2,$(NF-4),$(NF-3)} ' AT\_genes.gff. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| awk '/Chr1/' AT\_genes.gff. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

### TR

The tr (*tr*anslate) utility in UNIX can translate or transliterate the input to produce a modified output. It uses 2 sets of parameters, and replaces occurrences of the characters in the first set with the corresponding elements from the other set.

tr [OPTIONS] "STRING1" "STRING2" <INFILE >OUTFILE

Useful options are

-c complements the set of characters specified by string1

-d delete occurrences of string1 (string2 not needed)

-s squeeze repeats or multiple occurrences found in string1 will be replaced with one string2

Common uses of tr command are:

tr "a-z" "A-Z" or tr "[:lower:]" "[:upper:]"

Convert lower case to upper case (or upper to lower case)

tr -s '\n'

Convert each sequence of repeated newlines to a single newline

Files generated in both Mac and Windows OS will have a different newline character (to mark the end of the line) that is not recognized by the UNIX OS. Similarly files generated in UNIX will have a different newline that can't be read in Windows or Mac OS. The 'tr' command provides an easy way to convert these 'newlines' to different forms.

tr '\r' '\n' <MAC.TXT >UNIX.TXT

Convert Mac text file to UNIX text file

tr '\n' '\r' <UNIX.txt >MAC.TXT

Convert UNIX text file to MAC text file

tr -d '\015' <WIN.TXT >UNIX.TXT

Convert Windows text file to UNIX text file

tr '\n' '\015'<UNIX.txt >WIN.TXT

Convert UNIX text file to windows text file

NOTE: There are in built commands like dos2unix, mac2unix and unix2dos to do these conversions automatically in most recent versions of UNIX.

There are several utilities that can mask low complexity regions of the genomes such as repeats. They do that either by converting the bases/residues to lower case (soft masking) or converting them to N or X (hard masking). The public databases often store these soft masked genomes. When downloaded it might be useful to remove the masking, if your analysis doesn't require it (pattern searching etc.). It can be easily done by changes cases

tr "ATGC" "atgc" <AT\_cDNA.fa >AT\_cDNA\_tr.fa

Converts masking from the sequences and saves them in a new file

tr "ATGC" "AUGC" <AT\_cDNA.fa > AT\_rna.fa

Converts cDNA to mRNA sequence and saves them in a new file

### Word count

wc (*w*ord *c*ount) is another useful command that lets you count the number of words (and lines) in a file

wc FILENAME *try this* wc AT\_genes.gff

This outputs both number of words as well as lines in a file.

wc -l FILENAME *try this* wc -l AT\_genes.gff

Outputs only number of lines in file

Often these commands are "piped" with other commands to count certain things. *eg.*: Counting the number of files in a directory, counting the number of sequences etc.

**Task 2.6: Count how many files with .fa extensions are present in** sequences **directory.**

ls Sequences| wc -l

### Sort

sort command can be used to arrange things in a file. Simplest way to use this command is:

sort FILE1 > SORTED\_FILE1

Useful options include

-n numerical sort

-r reverse sort

-k N,N sort the Nth field (column), where N is a number. Sorting can also be done on the exact character on a particular field *eg.* -k 4.3,4.4 sorts based on 3rd and 4th character of the 4th field. Additionally you can supply additional -k for resolving ties.

-t specify the delimiters to be used to identify fields (default is TAB) *eg.* -t : to use ':' as delimiter

**Task 2.7: The 'Sequences' directory consists of numerically labeled files. UNIX can sort either alphabetically or numerically (not both) and hence they are arranged in** NT1.fa**,** NT10.fa**,** NT11.fa ***etc*. In order to sort them in an easy to read way, try using**

ls |sort -n -k 1.3,1.4

This command lists all the files in sequences directory and then passes it to sort command. Sort command then sorts it numerically but only using 3rd and 4th letters of the first field (file name)

Try using sort on AT\_genes.gff file

sort -r -k 1,1 AT\_genes.gff

sort -r -k 4,4 AT\_genes.gff

## Uniq

uniq (*uniq*ue) command removes duplicate lines from a sorted file, retaining only one instance of the running matching lines. Optionally, it can show only lines that appear exactly once, or lines that appear more than once. uniq requires sorted input since it compares only consecutive lines.

uniq [OPTIONS] INFILE OUTFILE

Useful options include

-c count; prints lines by the number of occurrences

-d only print duplicate lines

-u only print unique lines

-i ignore differences in case when comparing

-s N skip comparing the first N characters (N=number)

**Task 2.8: Number each lines based on number of occurrences:**

uniq -c ids.txt

**Print only duplicated lines.**

uniq -d ids.txt

**Print only unique lines.**

uniq -u ids.txt

### Comparing files

diff (*diff*erence) reports differences between files. A simple example for diff usage would be

diff [OPTIONS] FILE1 FILE2

Useful options include

-b ignore blanks

-w ignore white space (spaces and tabs)

-i ignore case

-r recursively compare all files (when comparing folders)

-s list all similar files (when comparing folders)

-y side by side comparison of files

The differences reported will be in the form of corrections that are required to change the first file to second file

Generate diffIDs.txt by comparing the differences between ids\_a.txt and ids\_b.txt

diff -y ids\_a.txt ids\_b.txt > diffIDs.txt

*Are these files different?*

comm (*comm*on) command compares two sorted files line by line.

comm [OPTIONS] FILE1 FILE2

-1 suppress lines unique to FILE1

-2 suppress lines unique to FILE2

-3 suppress lines that appear in both files

**Task 2.9: Compare the same files (ids\_a.txt and ids\_b.txt) again with 'comm' command and see how the outputs differ**

comm -1 ids\_a.txt ids\_b.txt

comm -2 ids\_a.txt ids\_b.txt

comm -3 ids\_a.txt ids\_b.txt

### DIVIDING FILES

cut divides the file into several parts and displays selected columns or fields from each line of a file. Normally cut command requires how the fields are separated and what fields needs to be displayed.

cut -d "," -f 2-4 FILE displays columns 2,3 and 4 of a file separated by ","

cut -d "|" -f 1,10 FILE displays 1st and 10th columns of a file separated by "|"

cut -f 1 FILE displays 1st column of a file, assumes TAB as delimiter

split generates output files of a fixed size (bytes or lines). Useful when huge file needs to be processed. *eg.,*

split -d -l 100 FILENAME SUFFIX

here -d specifies numeric suffix only (suffix00, sufix01, suffix02 *etc*.) while -l specifies number of lines in each file (100 in this case). If you want to split based on bytes, you can use -b option (*eg*. -b 1k or -b 1m for 1 KB and 1 MB respectively)

*From the commands that you have learned, can you combine the all the split files into a single file again?*

**Task 2.10: Display only first column of the** AT\_genes.gff **file using cut**

cut -f 1 AT\_genes.gff (press ctrl + c to exit) or

cut -f 1 AT\_genes.gff | less

**Similarly, display 1st, 4th and 5th column of the** AT\_genes.gff **file**

cut -f 1,4,5 AT\_genes.gff (press ctrl + c to exit) or

cut -f 1,4,5 AT\_genes.gff | less

**Verify if all the columns in** AT\_genes.gff **file has same number of entries in every field**

cut -f 1 AT\_genes.gff |wc -l

cut -f 2 AT\_genes.gff |wc -l

cut -f 3 AT\_genes.gff |wc -l

**Split the file** AT\_genes.gff **every 100,000 lines. Use** gff\_split **as suffix for the files and use numerical suffix.**

split -d -l 100000 AT\_genes.gff gff\_split

*How many split files are generated: \_\_\_\_\_\_\_\_\_*

ls gff\_split\* |wc -l

### COMBINING files

paste prints lines consisting of sequentially corresponding lines of each specified file. eg.,

paste FILE1 FILE2 > FILE3

Combines the contents of FILE1 and FILE2, side by side generating a new file, FILE3.

**Task 2.11: Combine columns of** ids\_a.txt **and** ids\_b.txt **files.**

paste ids\_a.txt ids\_b.txt

*How many columns do you see after combining? \_\_\_\_\_\_\_\_\_\_\_\_*

join combines two files based on the common field that is specified

join -t':' -1 N -2 N FILE1 FILE2

-t':' Specify field separator (here ":" but you can specify anything. Default is TAB)

-1 N Common field number (N) from the 1st file

-2 N Common field number (N) from the 2nd file

**Task 2.12: Join columns based on column 1 in** genes\_a.gff **and column 3 in** genes\_b.gff

join -1 1 -2 3 genes\_a.gff genes\_b.gff