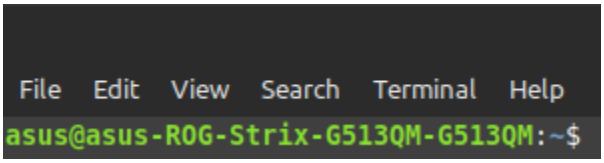
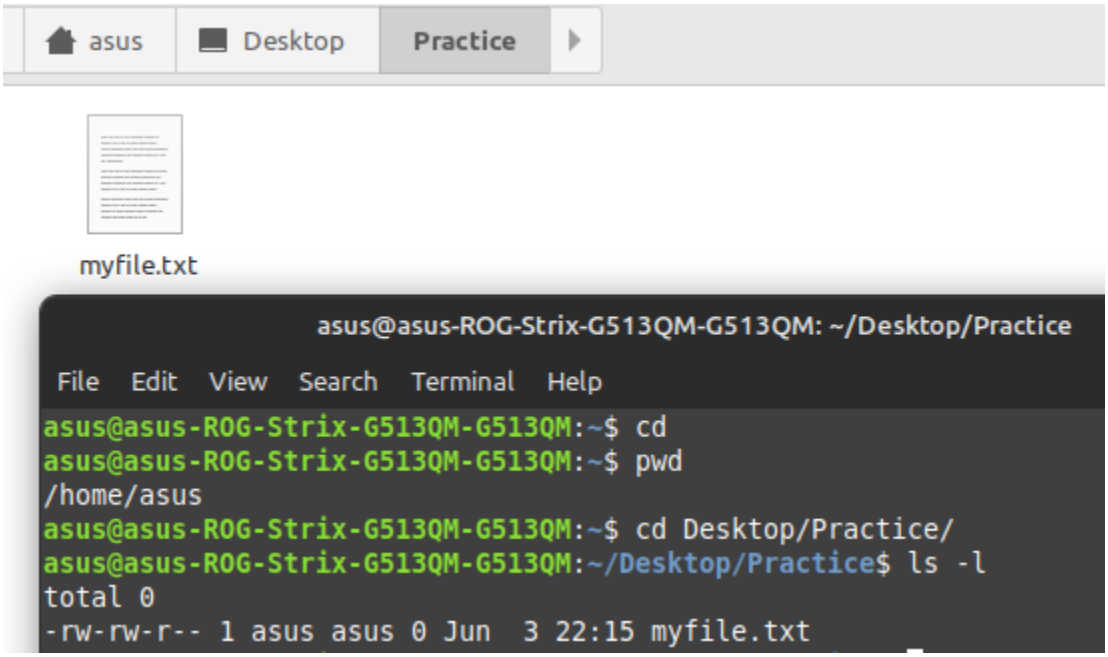
Linux command structure

When you open a terminal, you will see a command prompt ready to take commands. The default location on the terminal is your “home directory”. It is represented with the ~ (tilde) symbol.



All Linux commands are single words (can be alpha-numeric i.e. words consisting of letters and numbers). **For historical reasons, some of the early commands are only two letters long** and case sensitive. **Most of the command options (also called flags) are single letters**. They should be specified after the command before giving any input.



Here “**ls**” is the command to list the contents of the directory, “**-l**” is the option for long listing xx

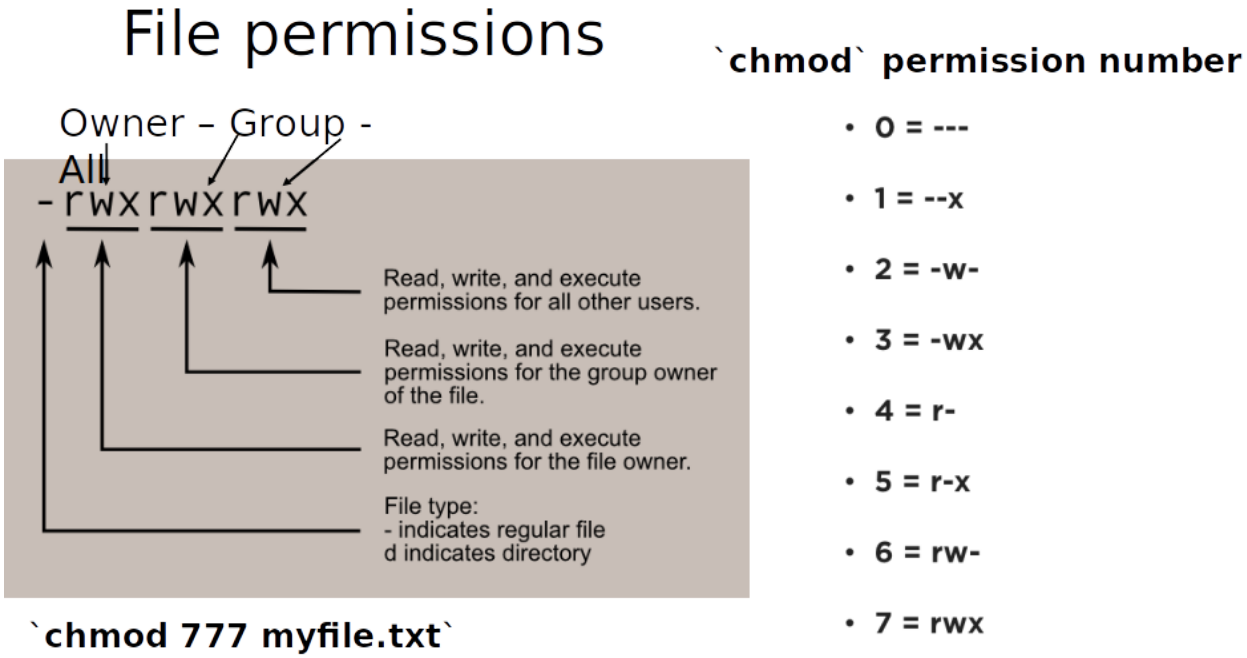
Please remember:

- Linux commands are case sensitive
- Single words

- Options have to follow the command
- Options can start with a single hyphen and a character or a double hyphen and word
- Single character options can be combined
- Arguments can be one or multiple inputs (ls -l Document Desktop)
- You can write more than one command separating with a semicolon;

You can use “tab” to auto-fill the command

File permissions



Linux is a multi-user operating system that can be accessed by many users simultaneously. This might make you think that a user can manipulate files and directories of another user, but all Linux operating systems protect file systems under two levels of authorisation (ownership and permission) to prevent unauthorized access to the filesystem in an effective and easy manner.

In Linux, there are two types of users: system users and regular users.

- System users are created by the operating system itself and are used to manage background processes.

- We generally create regular users to create and run processes interactively through a GUI or terminal.

Besides these two types of users, there is a superuser by the name root, which has access to the entire system to manage and override any settings in the system.

There are two levels of permissions assigned to the files, directories, and processes in Linux.

- The first one is permission groups, which is otherwise referred to as the ownership.
- The second one is permission types, which can be read, write, or execute.

Permission group

Owners: The user who creates a file, folder, or process is the owner.

Groups: Groups refers to anyone who is in the same group as the owner.

Others: Any user who is neither the owner of the file/directory and doesn't belong to the same group is assigned to others group.

Permission type

The operations each of the above three user groups can do is defined by permission types. There are three basic permission types that can be assigned to three groups of users and they are **read (r), write (w), and execute (x)**.

For files:

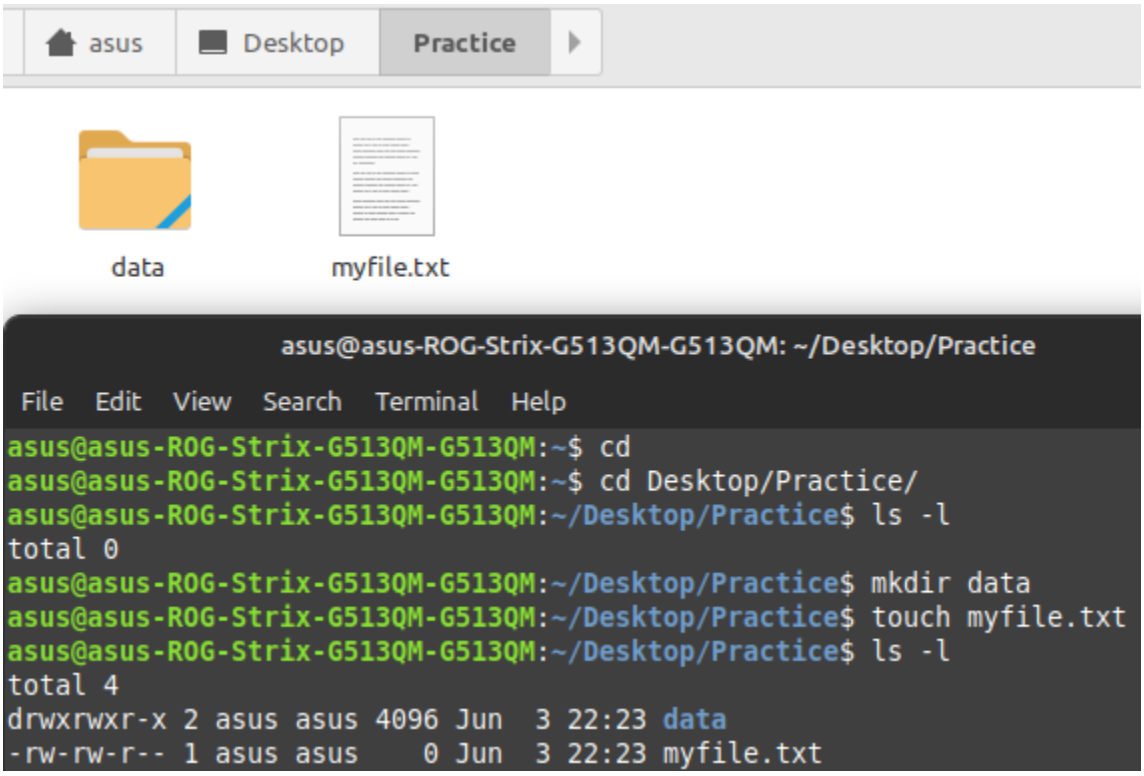
- **Read** is the ability to view the contents of a file.
- **Write** is the ability to edit or delete content of the file.
- **Execute** is the ability to run a file as an executable program.

For directories:

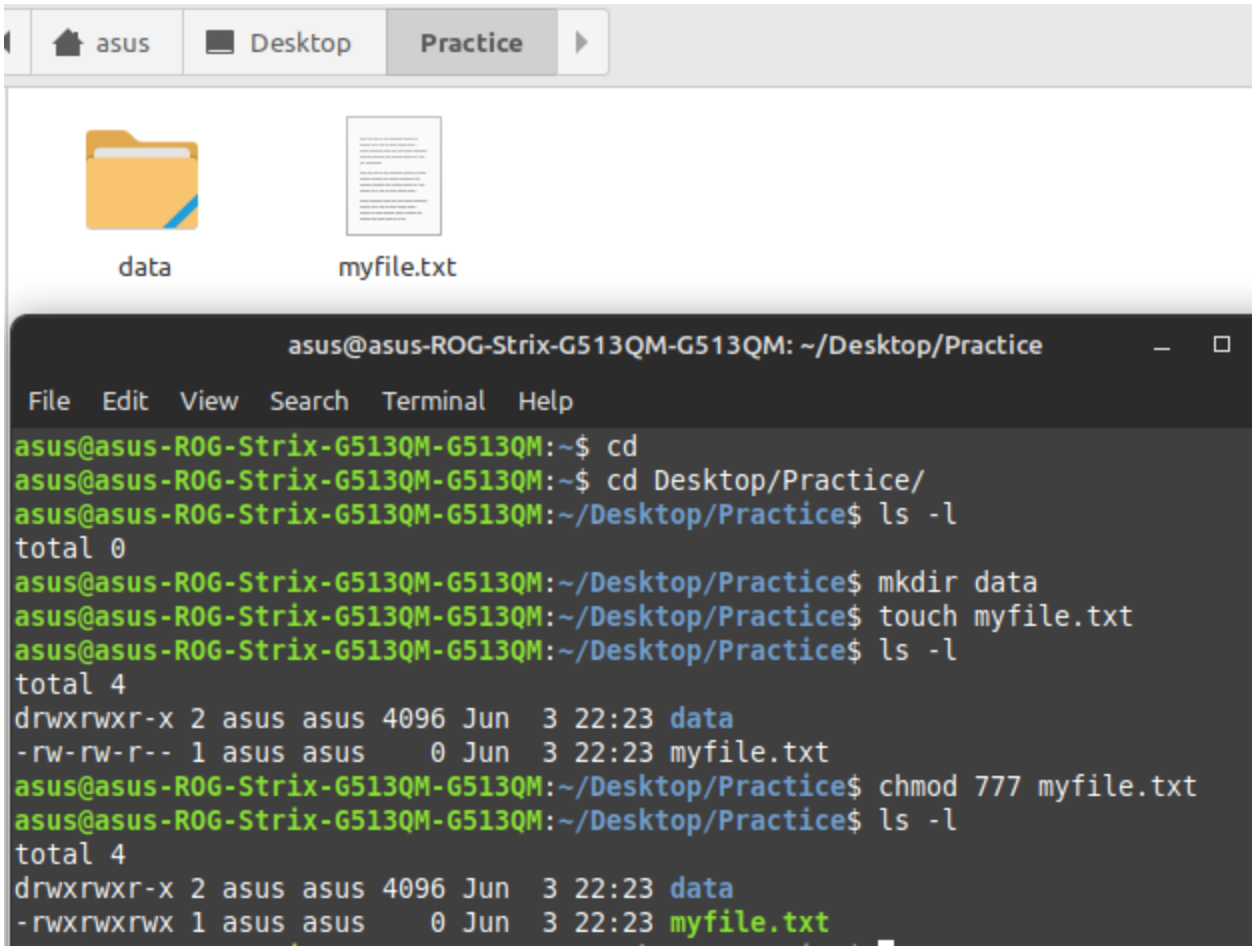
- **Read** is the ability to read the contents of a directory.
- **Write** is the ability to write into the directory, like creating files and sub-directories inside a directory.
- **Execute** is the ability to ***cd*** into the directory and to view the metadata of the files inside the directory using ***ls*** command.

Finding permission in a file/directory

To find the permissions that are assigned to files or directories, use **ls** command with **-l** switch.



- Ten characters in the format **drwxrwxrwx**, represents the permissions for all the three classes of users. The first character, **d**, signifies that the file is a directory.
- Then the next three characters (d**rwx**r-xr-x) represent the permissions that have been assigned to the owners of the file.
- Moving on to the next three characters (drwx**rwx**r-x), which is **rwx**, represents the group permissions. The users from users group can access the file according to the group permissions, which specify they can read, write and execute in the directory
- The last three characters (drwxr-x**r-x**) represent the permissions for other groups who are neither the owner nor a member of the group users and the permissions are set to read and execute only.



First Commands

mkdir

Make a directory. This command creates a directory if no files/directory exists with that name.

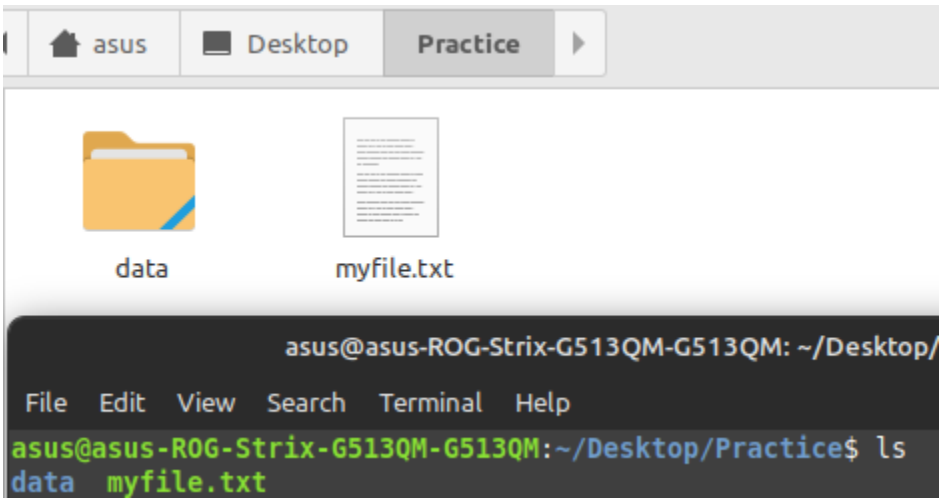
touch

It is the file's timestamp changing command. However, it can also be used for creating an empty file. This command is generally used for checking whether you have permission to write the file.

Example: **touch** myfile.txt

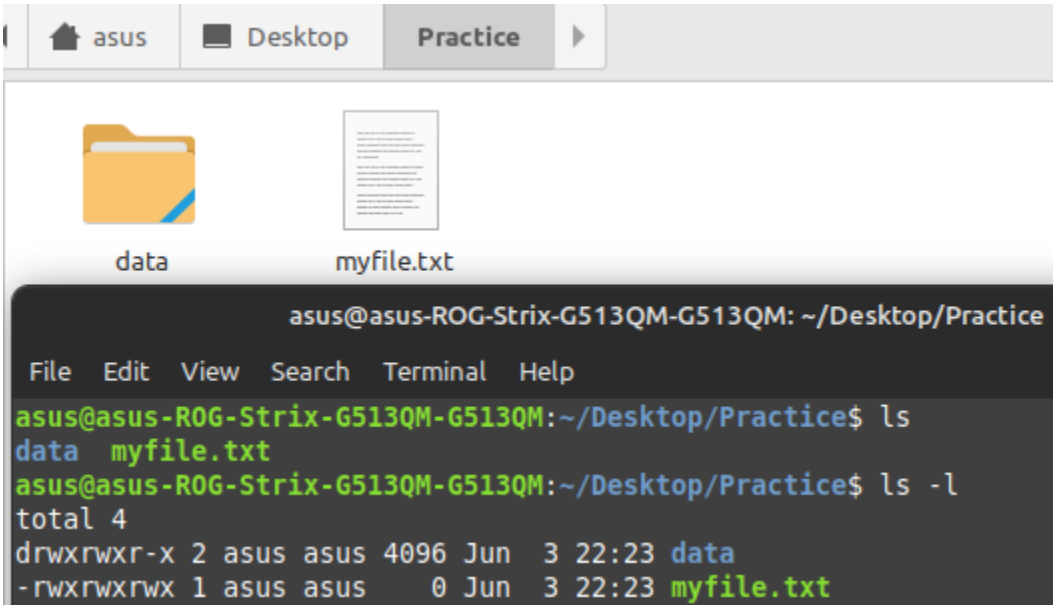
ls

Lists information about the files/directories. Default is the current directory. Sorts entries alphabetically.

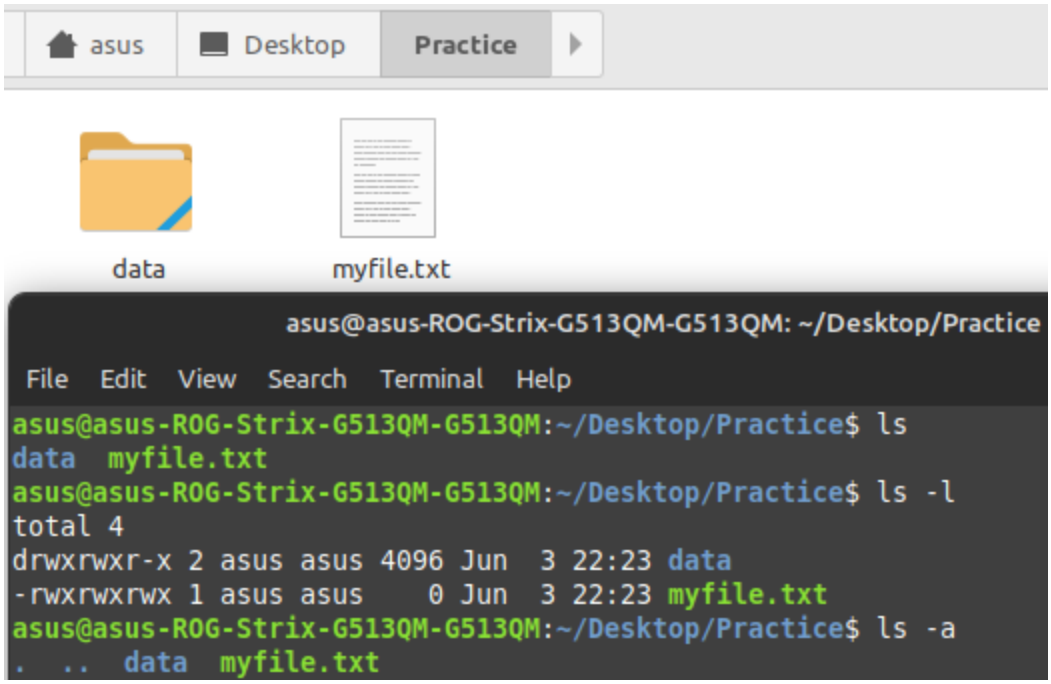


Commonly used options:

-l long list

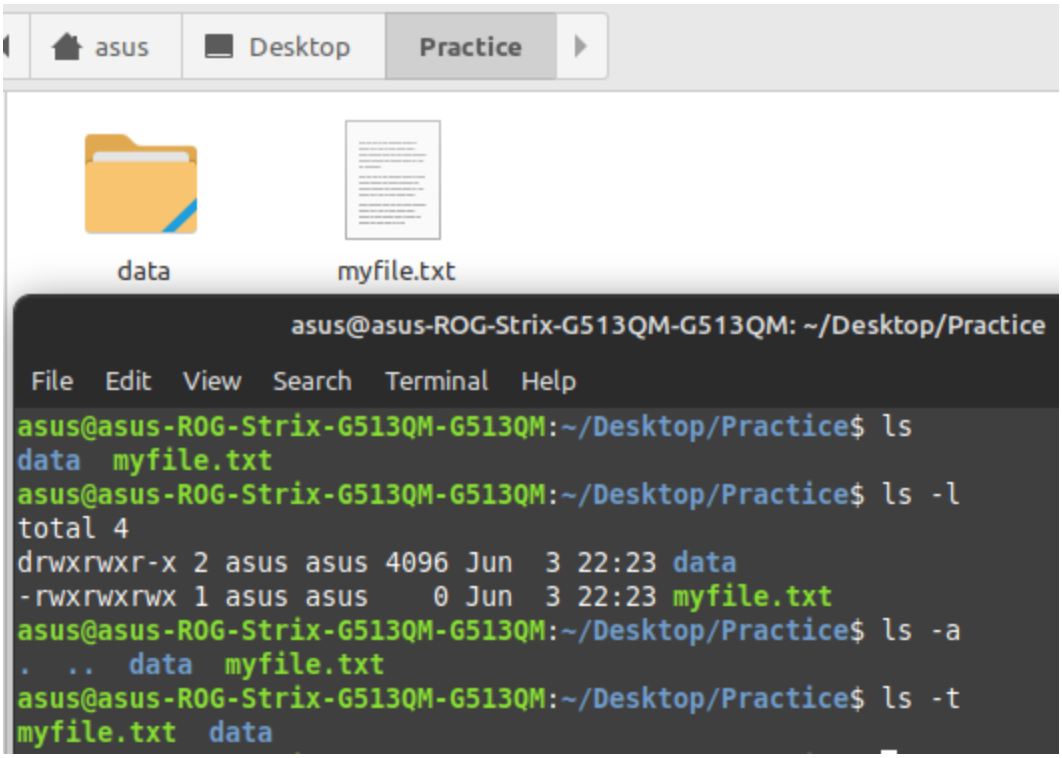


-a show all files (including hidden files)



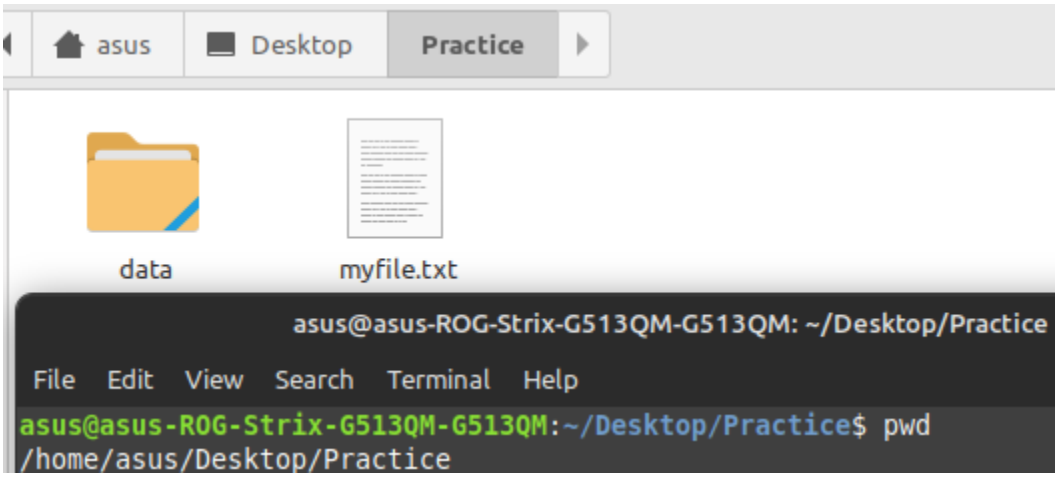
- `.`: This represents the current directory. It is a reference to the directory you are currently in.
- `..`: This represents the parent directory. It is a reference to the directory that contains the current directory.
- data: This is a directory. It is a subdirectory within the current directory.
- myfile.txt: This is a file. It is a regular file located in the current directory.

-t sort based on last modified time



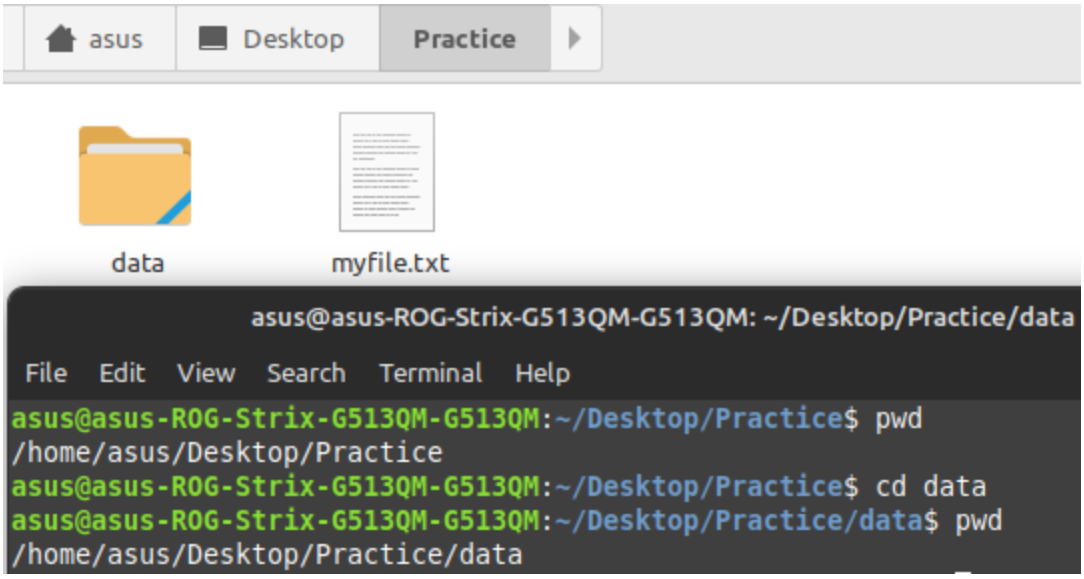
pwd

Will return current working directory's name



cd

Change directory. It is used for changing the working directory

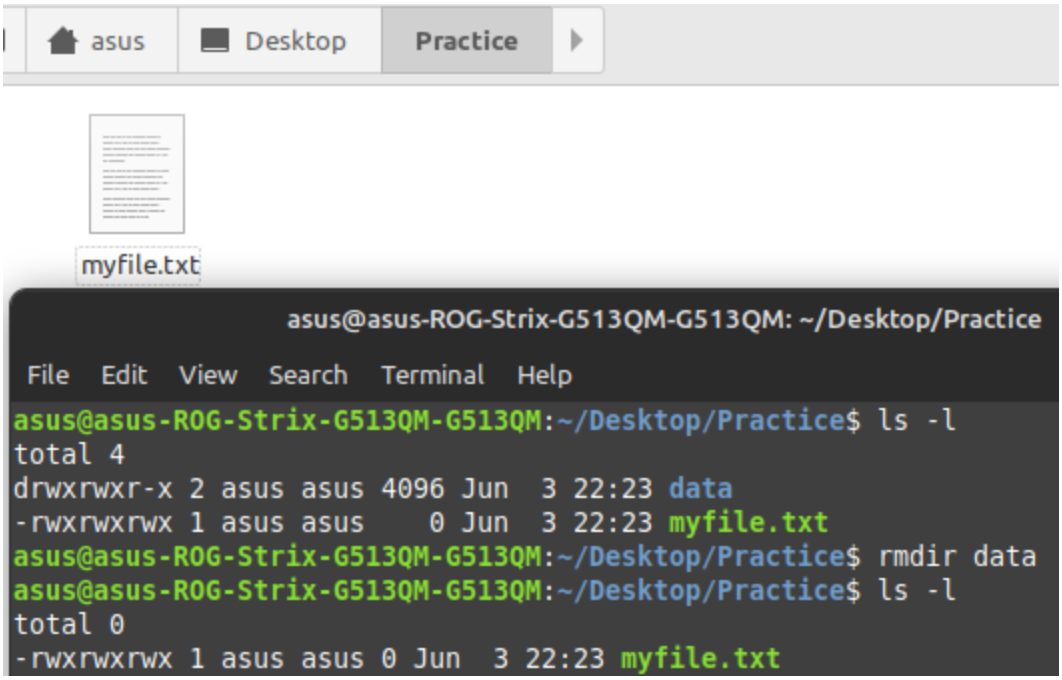


Example: **cd** data

Entering the "**cd**" command will bring you to the home directory.

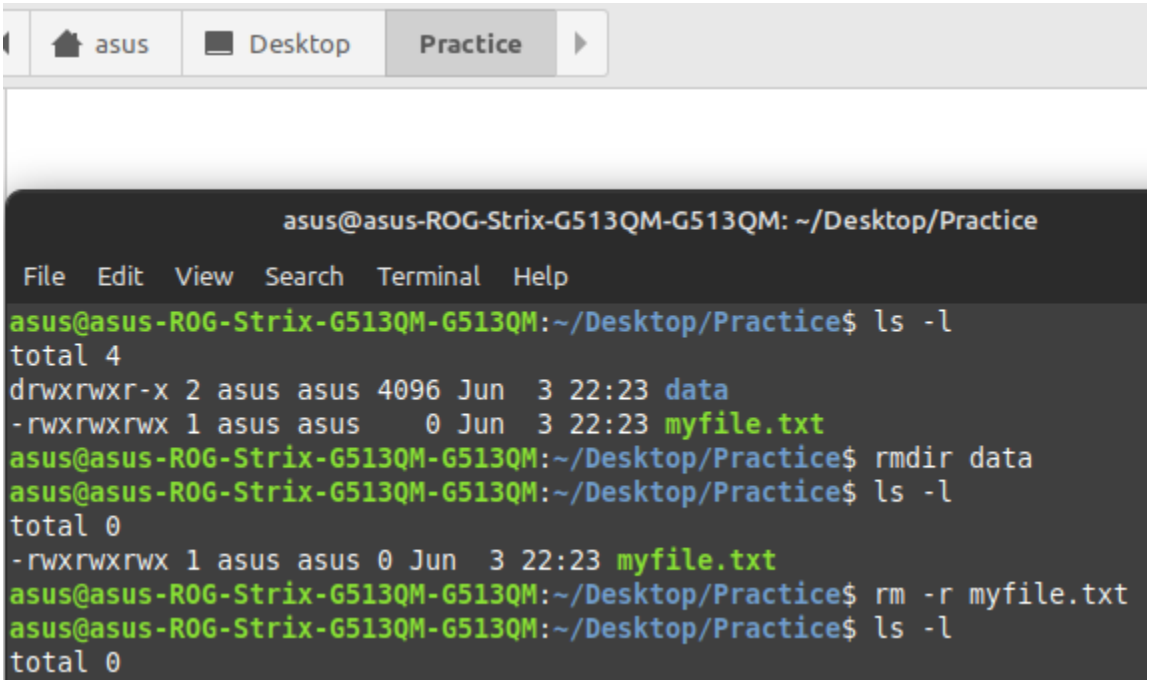
rmdir

Remove directory. This command removed an empty directory



Example: **rmdir** data

rm -r can also be used to remove directories but this removes directories that are not empty.



rm

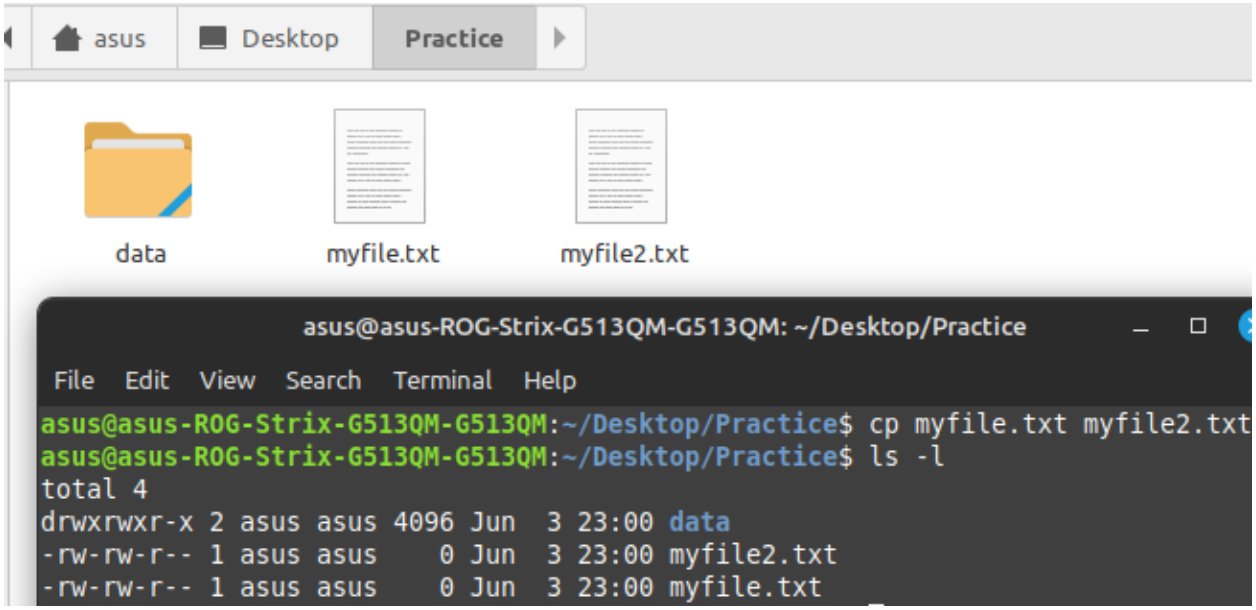
Means remove. **rm** is used for removing files and directories.

Example: **rm** myfile.txt

To remove directories, use the "**-r**" option. Please remember once a file or directory is deleted, it will not go to the "Recycle bin" in Linux and there is no way you can recover it.

cp

Copy files or directories.



Example:

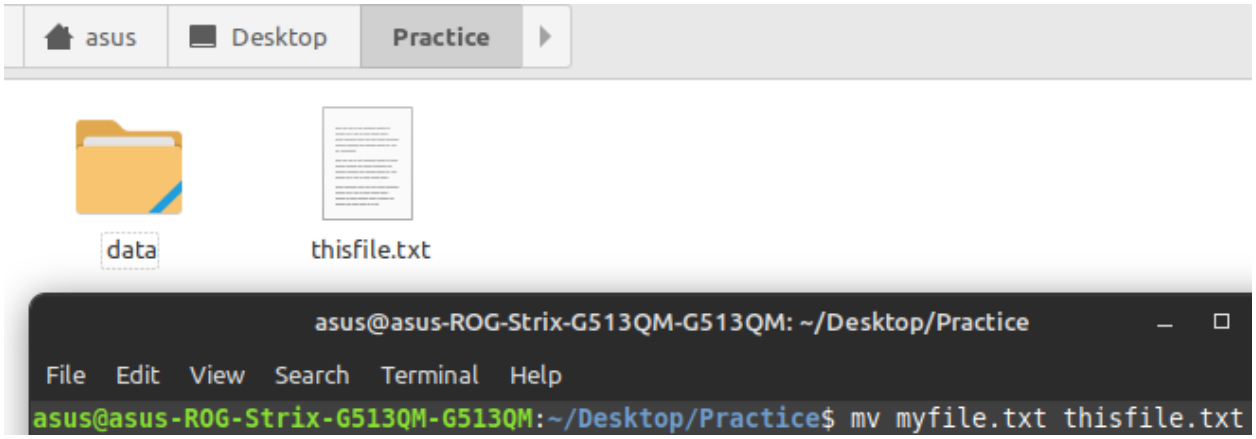
cp myfile.txt myfile2.txt

To copy directories, use the "**-r**" option.

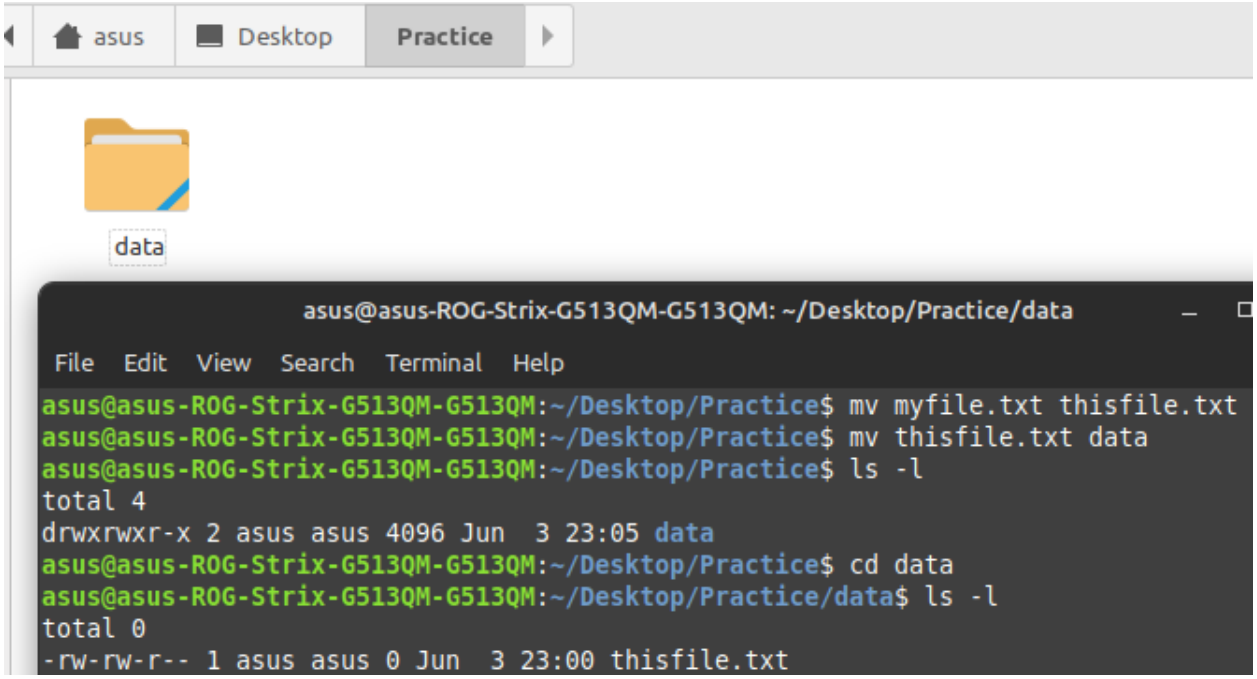
mv

To move file or directory or to rename a file

To move file or directory or to rename a file



mv thisfile.txt data/ (moves thisfile.txt -file to datadirectory.)

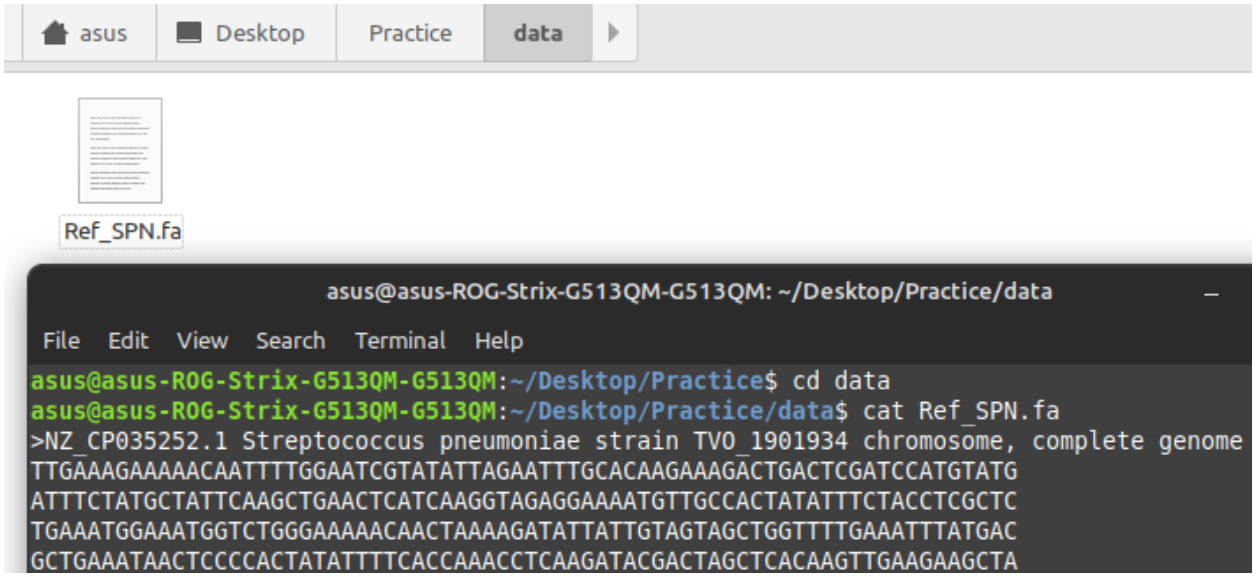


Viewing file content

Move to Intro_to_linux directory. We will use the Ref_SPN.fa file.

cat

Concatenate. Combine files and prints on the screen

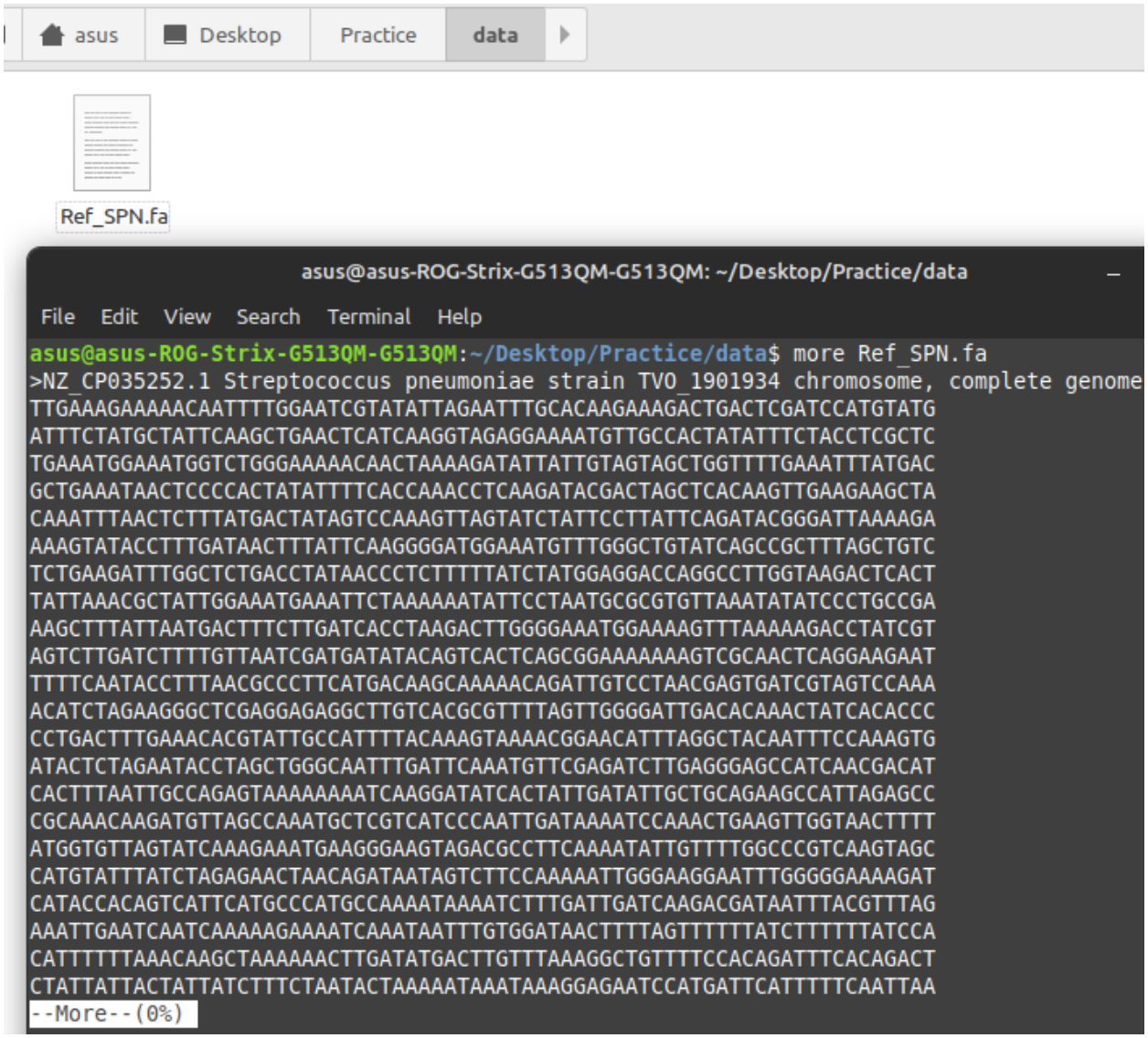


Example:

cat Ref_SPN.fa

more/less

These commands are used for viewing the files in the terminal. Its useful for scanning through large files.



Example: **more** Ref_SPN.fa

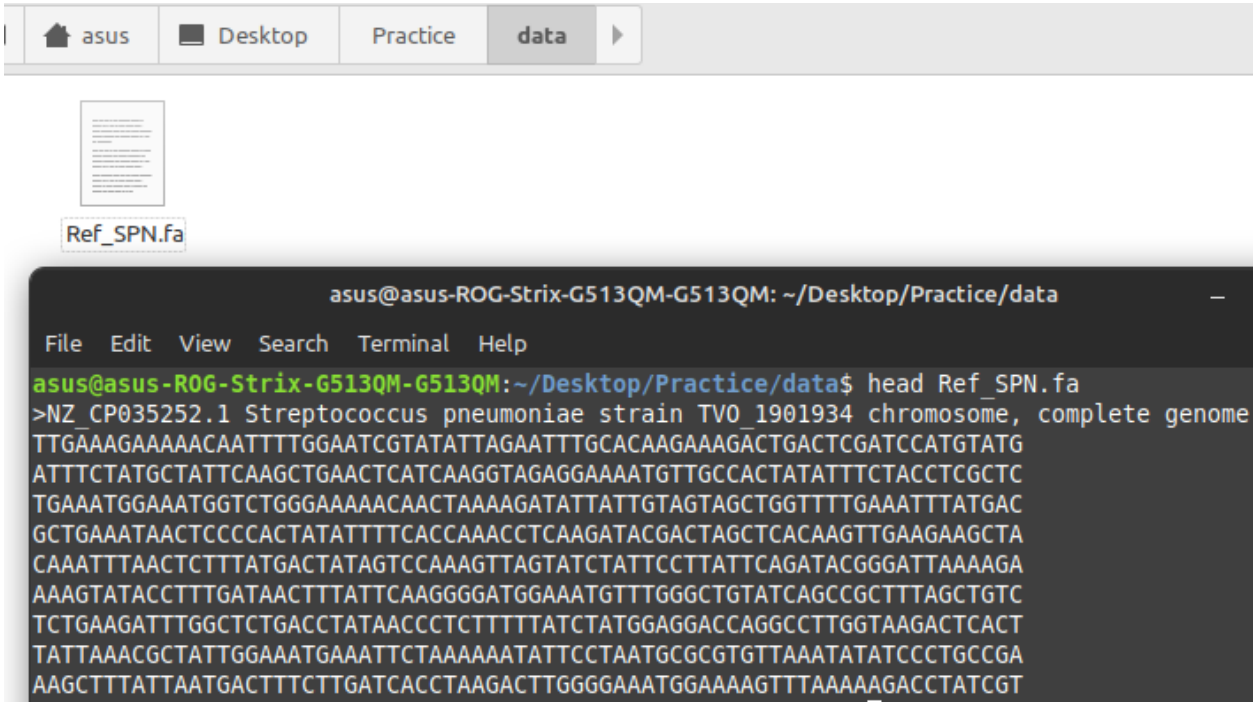
Press space to continue on to the next page. Press “q” to come out from the program

Example: **less** Ref_SPN.fa

Press space to continue on to the next page. Press “q” to come out from the program

head/tail

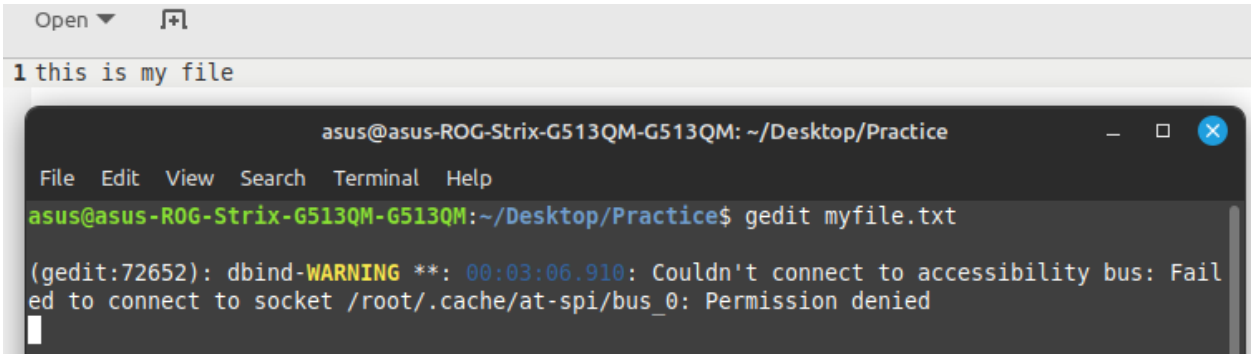
These commands show first and last lines (default is 10 lines), respectively, from a file



Example: **head** Ref_SPN.fa

File editors

File viewers show the content of the file without making any changes. To change the file content you have to use file editors. There are many non-graphical text editors like ed, emacs, vim and nano available on most of the Linux distributions. Some of them are very sophisticated (e.g., vi) and for advanced users. Here we will be learning about a “gedit”



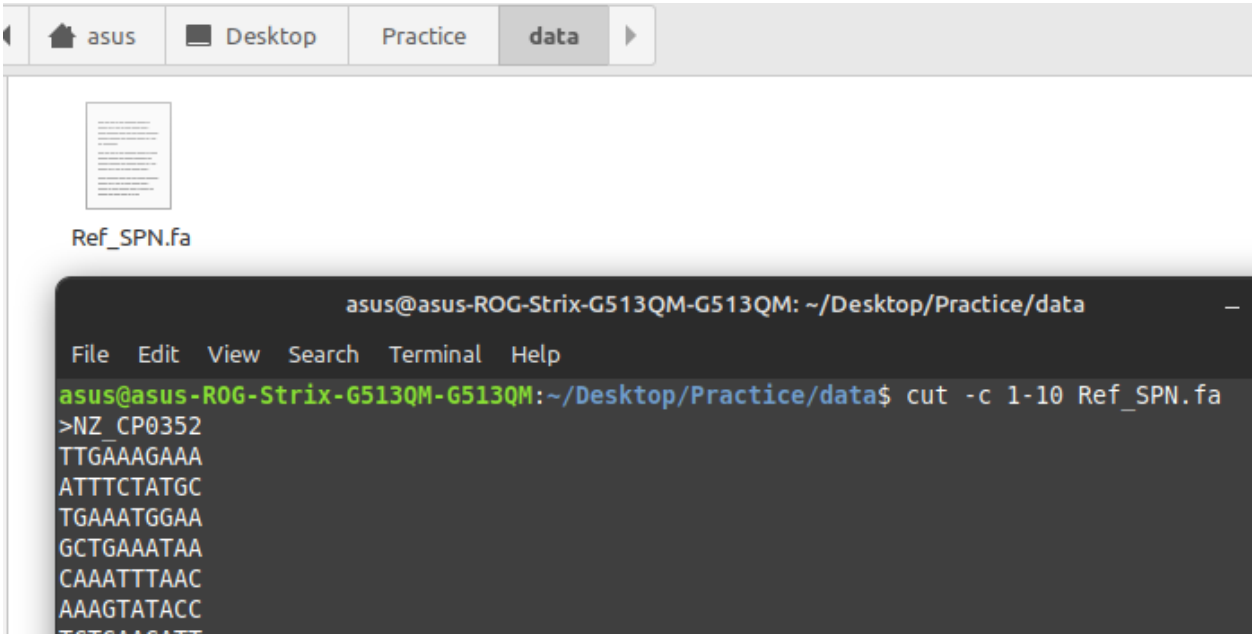
Example:gedit file name

Commands for text processing

cut

The cut command is a command line utility to cut a section from a file. Please see " **man cut**" for available options.

To cut a section of file use "**-c**" (characters)



Example: **cut -c 1-10** Ref_SPN.fa

The option "**-c 1-10**" will give you 1-10 characters from the input file.

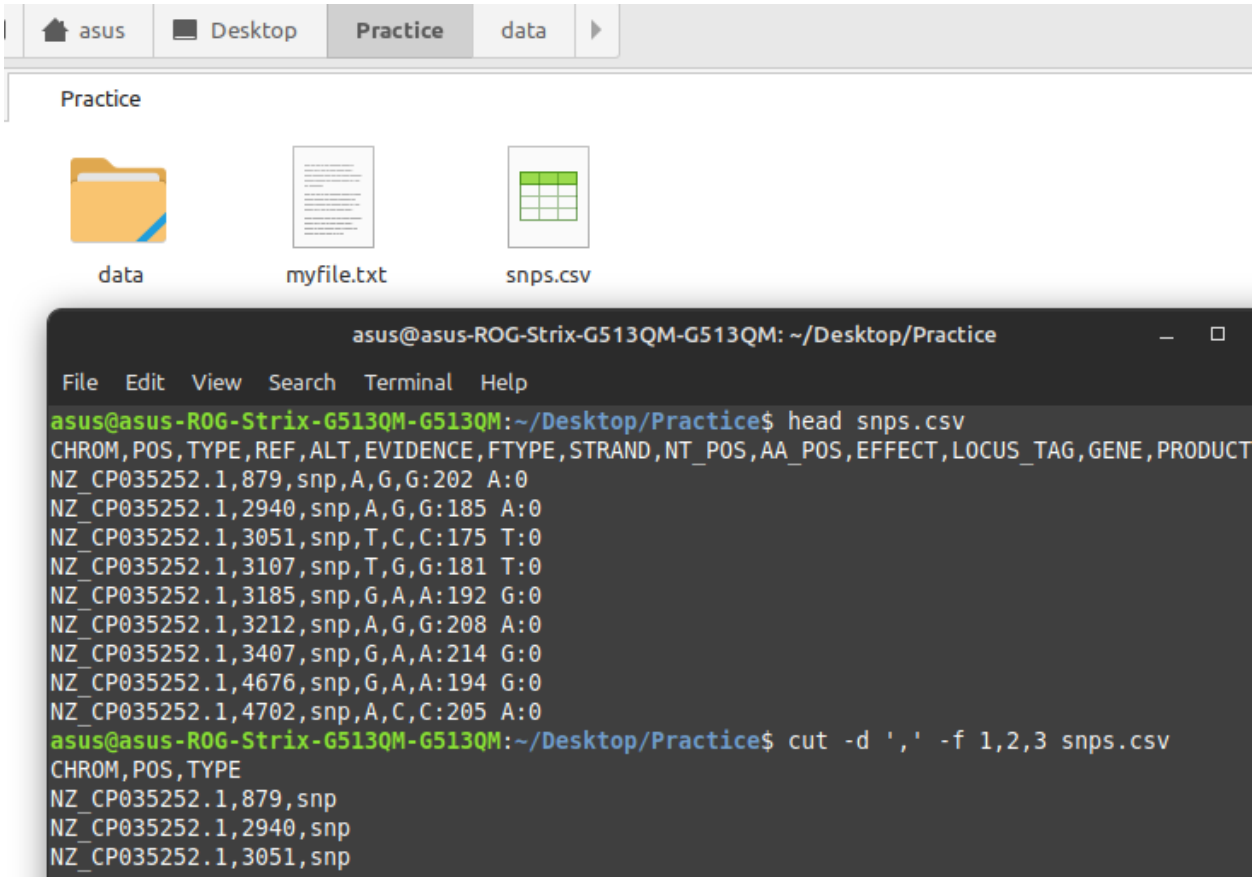
Here are some of the useful options:

-c: cut based on character position

-d: cut based on delimiter

-f: filed number

cut -d "," -f 1,2,3 snps.csv



sort

It is used for sorting input content

Some options are:

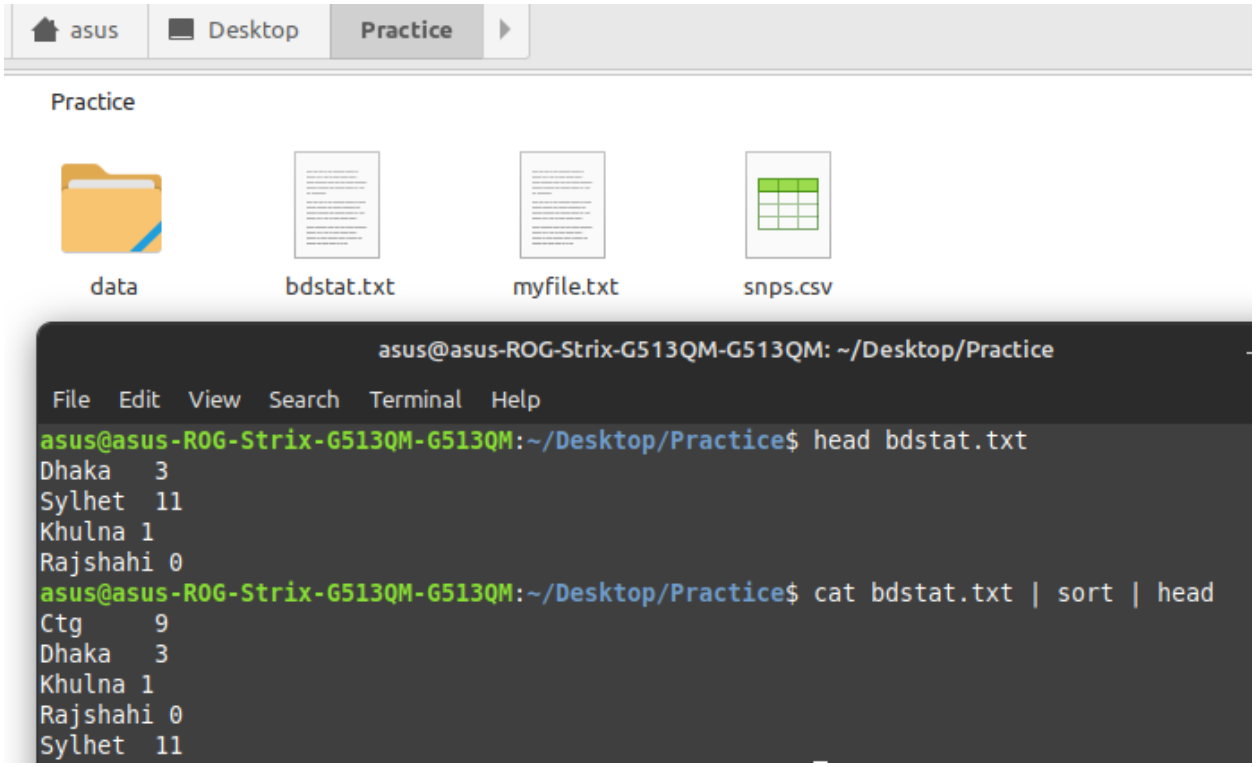
-t: field separator

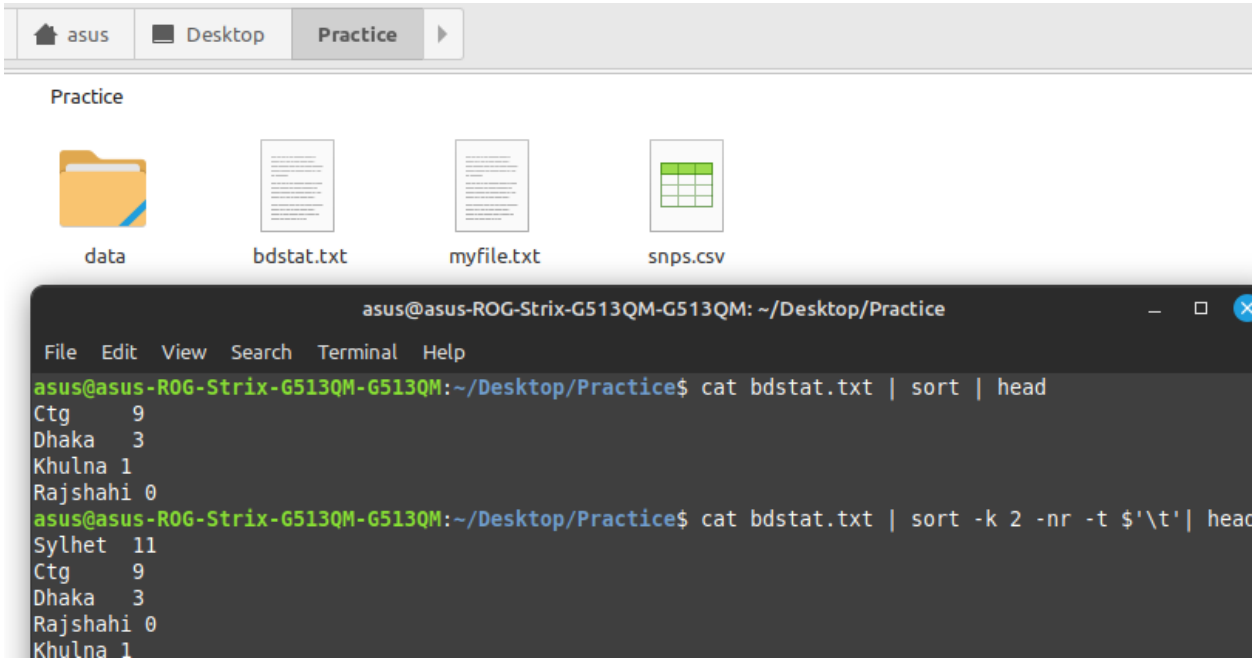
-n: numeric sort

-k: sort with a key (field)

-r: reverse sort

-u: print unique entries





grep

Searches input for a given pattern

Some options are:

-A: after context

-B: before context

-C: before and after context

-c: count

-l: file with match

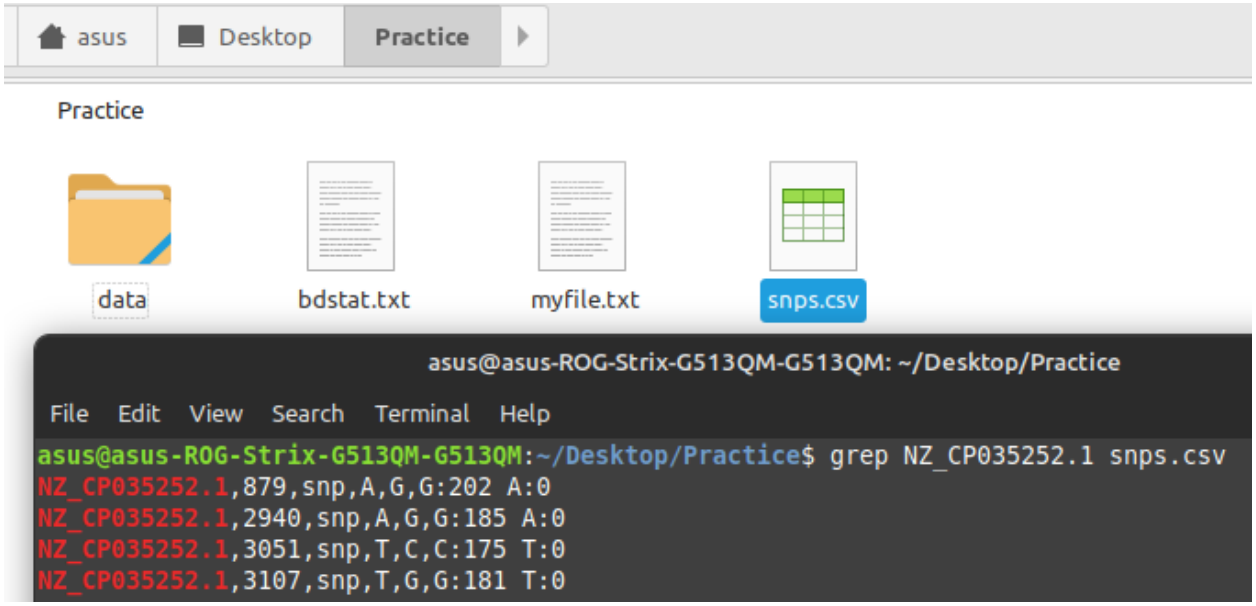
-i: ignore case

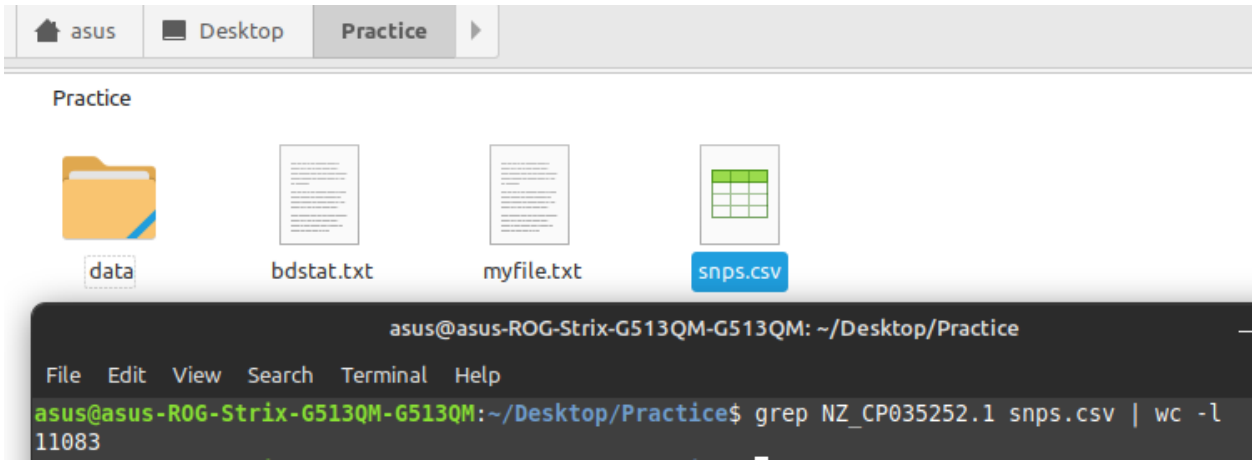
-o: only match

-v: invert match

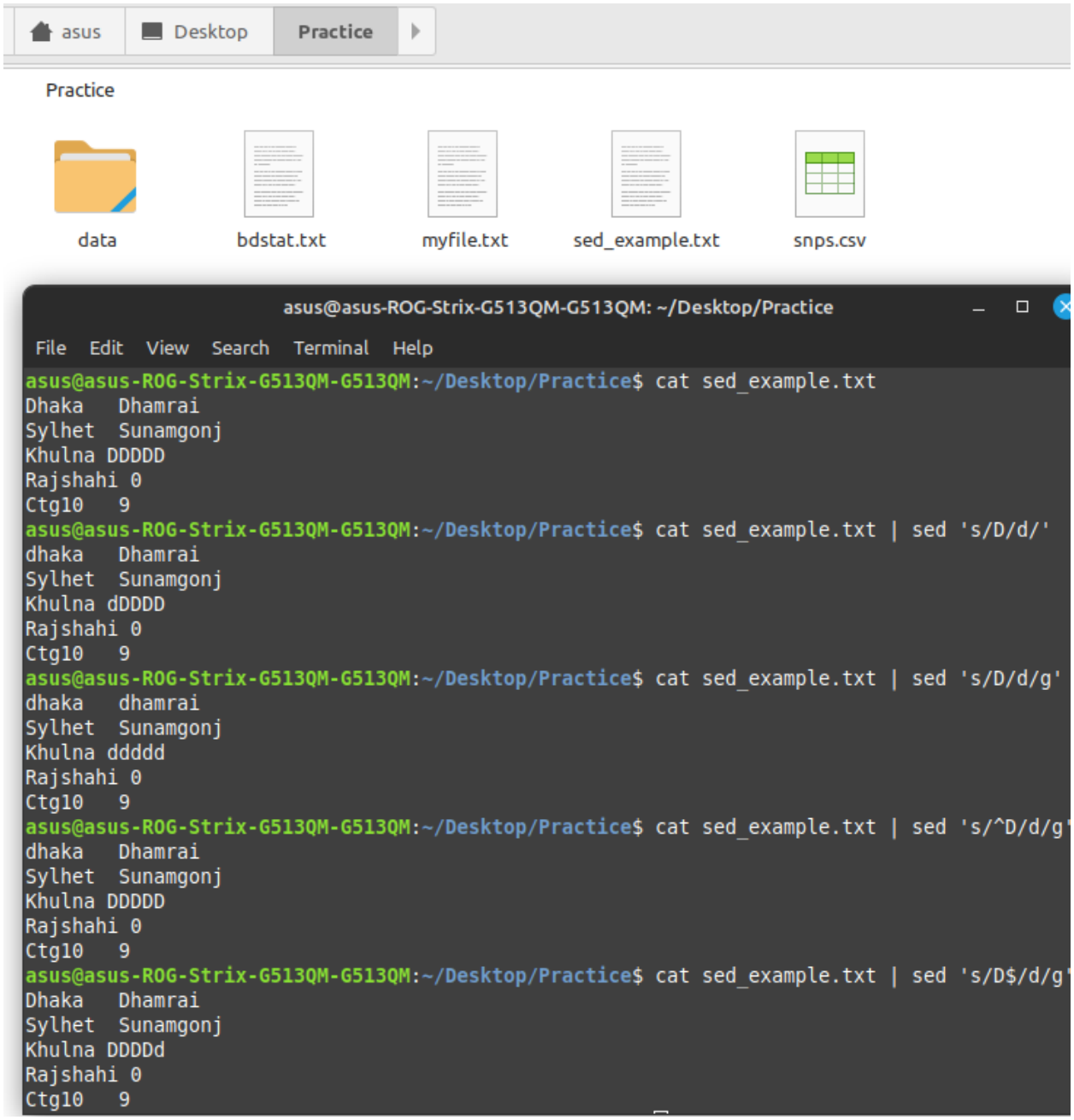
-**w**: word match

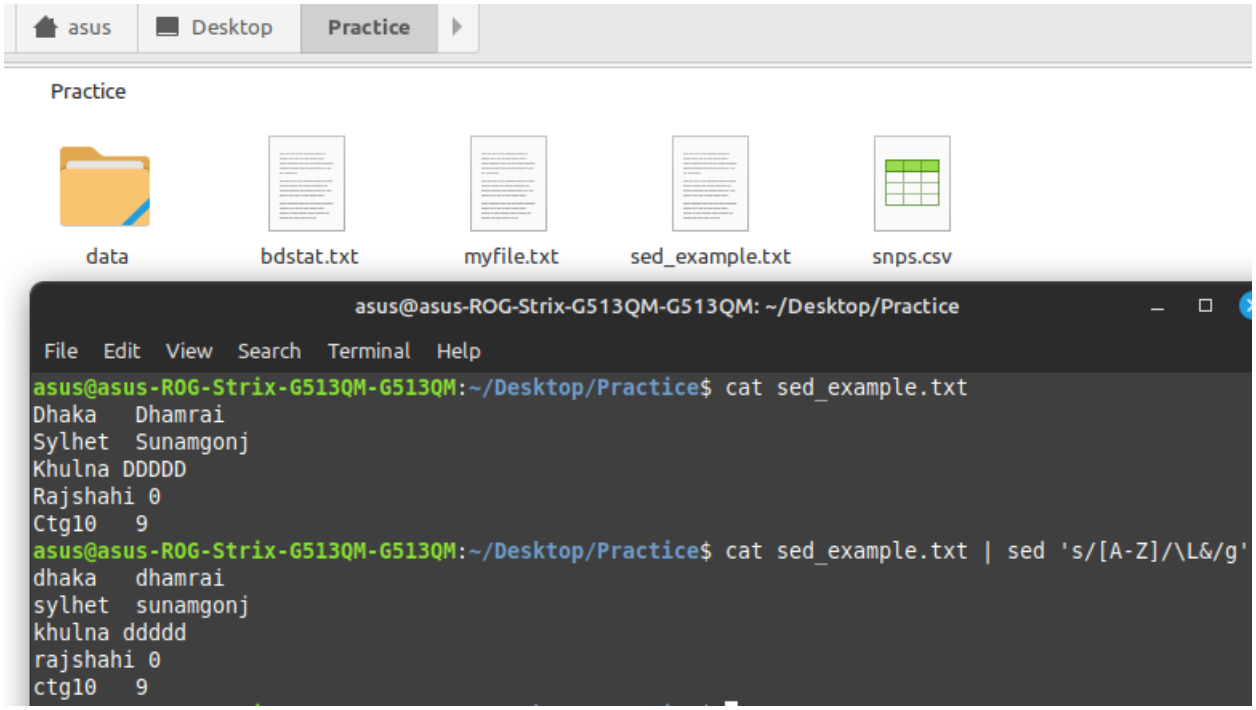
grep NZ_CP035252.1 snps.csv


asus
Desktop
Practice
Practice
data
bdstat.txt
myfile.txt
snps.csv
asus@asus-ROG-Strix-G513QM-G513QM: ~/Desktop/Practice
File Edit View Search Terminal Help
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ grep NZ_CP035252.1 snps.csv
NZ_CP035252.1,879,snp,A,G,G:202 A:0
NZ_CP035252.1,2940,snp,A,G,G:185 A:0
NZ_CP035252.1,3051,snp,T,C,C:175 T:0
NZ_CP035252.1,3107,snp,T,G,G:181 T:0


asus
Desktop
Practice
Practice
data
bdstat.txt
myfile.txt
snps.csv
asus@asus-ROG-Strix-G513QM-G513QM: ~/Desktop/Practice
File Edit View Search Terminal Help
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ grep NZ_CP035252.1 snps.csv | wc -l
11083

sed


asus
Desktop
Practice
Practice
data
bdstat.txt
myfile.txt
sed_example.txt
snps.csv
asus@asus-ROG-Strix-G513QM-G513QM: ~/Desktop/Practice
File Edit View Search Terminal Help
asus@asus-ROG-Strix-G513QM:~/Desktop/Practice$ cat sed_example.txt
Dhaka Dhamrai
Sylhet Sunamgonj
Khulna DDDDD
Rajshahi 0
Ctg10 9
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt | sed 's/D/d/'
dhaka Dhamrai
Sylhet Sunamgonj
Khulna dDDDD
Rajshahi 0
Ctg10 9
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt | sed 's/D/d/g'
dhaka dhamrai
Sylhet Sunamgonj
Khulna ddddd
Rajshahi 0
Ctg10 9
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt | sed 's/^D/d/g'
dhaka Dhamrai
Sylhet Sunamgonj
Khulna DDDDD
Rajshahi 0
Ctg10 9
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt | sed 's/D$/d/g'
Dhaka Dhamrai
Sylhet Sunamgonj
Khulna DDDDd
Rajshahi 0
Ctg10 9


asus
Desktop
Practice
Practice
data
bdstat.txt
myfile.txt
sed_example.txt
snps.csv
asus@asus-ROG-Strix-G513QM-G513QM: ~/Desktop/Practice
File Edit View Search Terminal Help
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt
Dhaka Dhamrai
Sylhet Sunamgonj
Khulna DDDDD
Rajshahi 0
Ctg10 9
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat sed_example.txt | sed 's/[A-Z]/\L&/g'
dhaka dhamrai
sylhet sunamgonj
khulna ddddd
rajshahi 0
ctg10 9

awk

```
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk {'print'}
a       2       4
a       5       10
b       4       7
a       3       2
c       9       4
d       1       0
c       7       6
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk {'print $1 $2'}
a2
a5
b4
a3
c9
d1
c7
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk {'print $1 "\t" $2'}
a       2
a       5
b       4
a       3
c       9
d       1
c       7
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk {'print $1 "\," $2'}
awk: cmd. line:1: warning: escape sequence `\,' treated as plain `,'
a,2
a,5
b,4
a,3
c,9
d,1
c,7
```

```
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk '$2 >=5 {print}'
a       5       10
c       9       4
c       7       6
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk '$2 >=5 && $1 =="a"{print}'
a       5       10
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk '$2 >=5 && $1 =="a"{print $3+5}'
15
```

```
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk ' $NF<=3 {print}'
a       3       2
d       1       0
asus@asus-ROG-Strix-G513QM-G513QM:~/Desktop/Practice$ cat awk_example.txt | awk ' $NR<=3 {print}'
a       3       2
c       9       4
d       1       0
c       7       6
```

Pipes

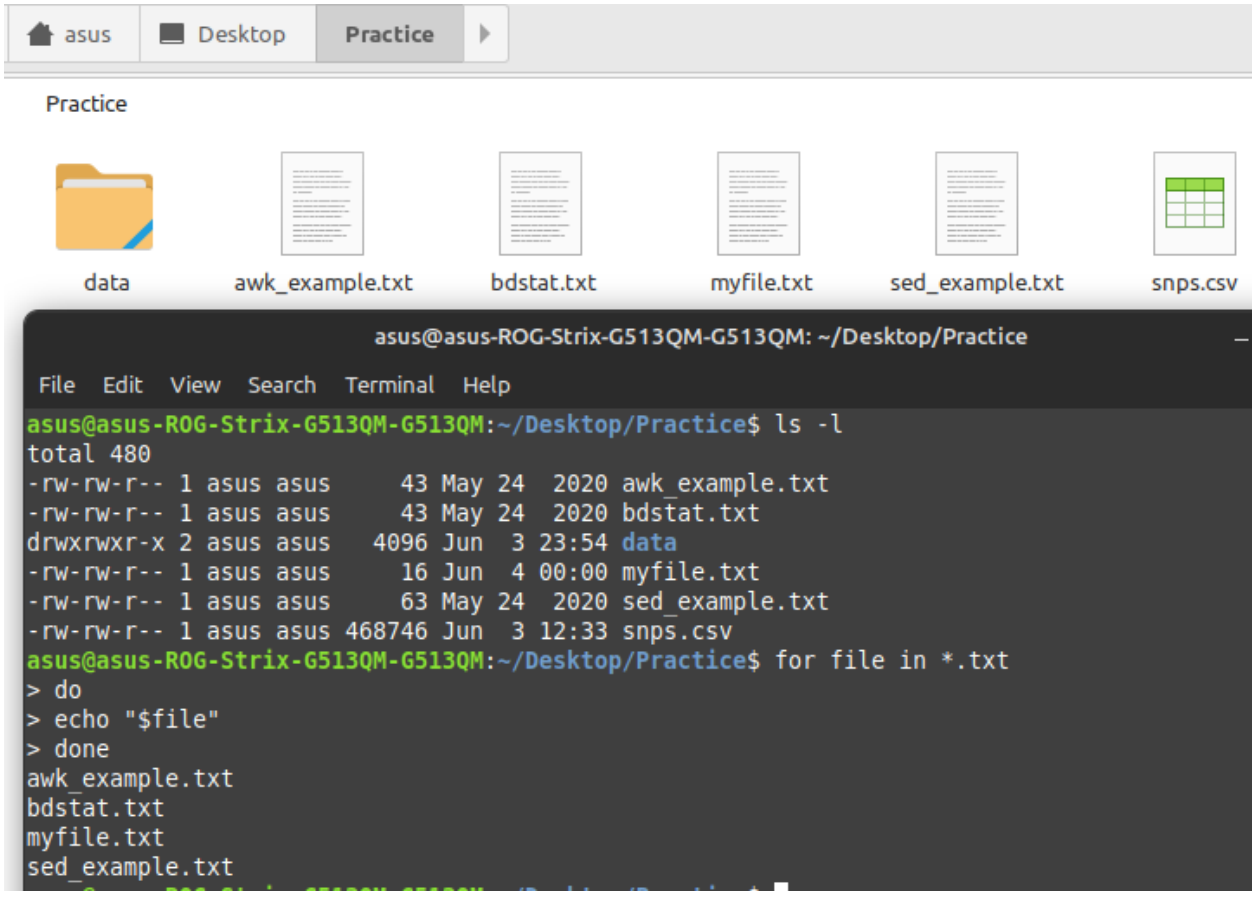
Piping in Linux is a very powerful and efficient way to combine commands. Pies (|) in Linux acts as connecting links between commands. Pipes make a previous command output as next commands input. We can nest as many commands as we want using

pipes. They play an important role in smooth running of the command flow and reducing the execution time.

To print isolates from Bentley et al

sort -t “,” -nk2 serotypes_GPS_resources.txt | head

for



##

Linux Cheatsheet

Directory/file commands

pwd: print working directory

ls -l -alh: list the contents of the current directory

cd dir: change directory to dir

mkdir dir: make directory named dir

touch file: make a file named file

m file: remove a file named file

rm -r dir: remove directory and contents named dir

rm -f dir: forcefully remove a file named file

rm -rf dir : forcefully remove a directory dir and contents (careful with this)

mv file1 file2 : move file1 to file2 (used for renaming files)

mv dir1 dir2 : move dir1 to dir2 (used for renaming dirs)

cp file1 file2: copy file1 to file2

cp -r dir1 dir2 : copy dir1 to dir2

cat file: display the contents of file to stdout

less file: display the contents of file fitting within the terminal screen

head -n 10 file: display the first 10 lines file

tail -n 10 file: display the last 10 lines of file

sort file: display the contents of the file with each line sorted

wc -l file: count the number of lines in file

ln -s target name: create a link to the target file with name

System commands

w: display who is logged in

whoami: display who you

man command: display info about command

df -h: display current disk usage

du -sh dir: display disk usage of dir

which app: display the path to the location of the app

whereis app : display all possible paths to the app

history: display all commands that have been run

clear: clear the terminal of text

File permission commands

chmod 777 file: set read(r) write(w) and execute(x) for all users

chmod 755 file: set owner to rwx and all other users to rx

chmod 766 file: set owner to rwx and all other users to rw

chmod 644 file: set owner to rw and all other users to r

chmod +x file: make file executable for all users

chown user file: change the owner of file to user

Compression commands

tar -cf file.tar files: create a tar named file.tar containing files

tar -xf file.tar: extract the files from file.tar

tar -czf file.tar.gz files: create a tar with Gzip compression

tar -xzf file.tar.gz: extract a tar using Gzip

tar -cjf file.tar.bz2 files: create a tar with Bzip2 compression

tar -xjf file.tar.bz2: extract a tar using Bzip2

gzip file: compresses file and renames it to file.gz

gzip -d file.gz: decompresses file.gz back to file

Process commands

ps -e: snapshot of processes

top: show processes in real time

kill pid: kill processes with id pid

pkill name: kill processes with name

killall name: kill all processes with the name

Searching commands

grep pattern files: search for pattern in files

grep -r pattern dir: search for pattern in dir

find dir -name “pattern“: find all files with pattern in name in dir

Piping commands

cmd > file: redirect the standard output (stdout) of cmd to file

cmd 2> file: redirect the standard error (stderr) of cmd to file

cmd &> file: redirect the stdout and stderr of a cmd to file

cmd >> file: redirect the stdout of cmd to file append to file if it exists

cmd > /dev/null: discard the stdout of cmd

cmd < file: redirect the contents of the file to the standard input (stdin) of cmd

cmd <(cmd1): redirect the stdout of cmd1 through a file to cmd (useful if cmd takes a file input)

cmd1 | cmd2: redirect the stdout of cmd1 to the stdin of cmd2

xargs cmd: reads data from stdin and executes cmd one or more times depending on the input

Other useful commands

count the number of unique lines in a file

cat file.txt | sort -u | wc -l

find all files with “assembly” in the name and copy them to a single assembly.txt file

find . -name “*assembly*” | xargs cat > assembly.txt

copy all “.fastq.gz” files from dir1 to dir2

cp `find dir1 -name “.fastq.gz”` dir2

split a multi fasta to individual fasta files

awk ‘/^>/{s=++d”.fa”} {print > s}’ multi.fa

convert a fastq file to fasta

sed -n ‘1~4s/^@/>/p;2~4p’ file.fq > file.fa

calculate the mean length of reads in a fastq file

awk ‘NR%4==2{sum+=length($0)}END{print sum/(NR/4)}’ input.fastq

create a backup of files here all .txt files are backedup as .bak

find . -name “*.txt” | sed “s/\.txt$//” | xargs -i echo mv {}.txt {}.bak | sh