

Introduction to Linux

 canvas.instructure.com/courses/5883171/pages/introduction-to-linux

Note: Before you begin this section, navigate to the [Intro to Linux](#) → [Links to an external site.](#) folder which is in your [section one](#) → [Links to an external site.](#) folder. You will use this folder and its contents to learn and practice this section.

Linux is an open-source operating system (OS) developed based on the kernel created by Linus Torvalds. In the last two decades, Linux has gained so much popularity and now it is used on many platforms. Nowadays, most of the high-end services to mobile phones (Android OS or iOS) run on different variants of Linux/Unix.

The Terminal

A computer is made up of hardware and software components. The kernel is part of the operating system that allows interaction between the hardware and the software, but it only understands machine language (binary code). Shell is a program that performs the job of converting our/user commands into binary code for the kernel. The terminal is the program that provides a graphical interface where users can enter commands that are converted into binary code by the shell to be executed by the computer via kernel. We use a terminal (AKA command line interface) to interact with the operating system. The terminal by default runs one of the “shells”. Shell is a program that sits between the user and the kernel (allows interaction between the hardware and the software in a computer), and translates user commands into machine code. The advantages of using the command line are greater control and flexibility over the system or software. Moreover, multiple commands can be saved in a file and executed as a script.

The most common shells are:

- Bourne Shell
- Bourne Again Shell
- Z Shell
- C Shell (variant is T Shell)
- K Shell

Among these, Bourne Again Shell (BASH) is the most popular one. This is the default shell on most Linux systems and we will be using it throughout our course.

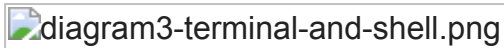


Image obtained [here](#) → [Links to an external site.](#)

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.

```
jm61@bioinformatics-server-1:~$
```

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.

Example: ls -l

```
jm61@mib116462s Intro_to_linux % ls -l
```

```
total 4112
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:47 GPS
```

```
-rwx-----@ 1 jm61  staff 2095972 23 Apr 2022 GPSC33_reference.fasta
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:43 JUNO
```

```
-rwx-----@ 1 jm61  staff   7123 18 Aug 17:01 serotypes_GPS_resources.txt
```

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

Note

In this course, commands that you need to enter into the Terminal window (i.e., command line) are presented in a black box. Sometimes a command is long and doesn't fit on a single line on a page, but it should still be entered as one single line on the computer terminal. Different commands are separated from each other in these tables using blank lines.

A few tips to remember:

- *Use the Tap button to automatically complete filenames - especially long ones*
- *Use the Up Arrow to scroll through your previous commands, it enables you to easily re-run or re-use/change/correct old commands*
- *Case Matters, following file names are all different*

Myfile.txt

MyFile.txt

MYFILE.txt

myfile.txt

my file.txt

My_file.txt

Watch out for number 1s being confused with lowercase letters L's , and capital O's being confused with zeroes

l = lowercase letter L

1 = number one

O = Capital letter O

0 = Zero

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 unauthorised 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.

```
jm61@mib116462s Intro_to_linux % ls -l
```

```
total 4112
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:47 GPS
```

```
-rwx-----@ 1 jm61  staff 2095972 23 Apr 2022 GPSC33_reference.fasta
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:43 JUNO
```

```
-rwx-----@ 1 jm61  staff  7123 18 Aug 17:01 serotypes_GPS_resources.txt
```


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 (**drwxr-xr-x**) represent the permissions that have been assigned to the owners of the file. The owner jm61 can read, write, and execute to the folders GPS and JUNO.

Moving on to the next three characters (**drwxr-xr-x**), which is **r-x**, represents the group permissions. The users from users group can access the file according to the group permissions, which specify they can read and execute in the directory but cannot write into it. The hyphen signifies that the permission is not granted.

The last three characters (**drwxr-xr-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.

The 11th character (**drwxr-xr-x 2 jm61**) is a number that represents the number of hard links for the file and is not related to permission for a file. The two columns next to this number (**drwxr-xr-x 3 jm61 staff**) represents the owner and group of the file.

Further reading on Linux permissions: <https://blog.ssdnodes.com/blog/linux-permissions/>  Links to an external site.

Quiz

What are the permissions for the file GPSC33_reference.fasta?

First Commands

ls

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

```
jm61@mib116462s Intro_to_linux % ls
```

```
GPS JUNO
```

```
GPSC33_reference.fasta serotypes_GPS_resources.txt
```

Commonly used options:

-l long list

```
jm61@mib116462s Intro_to_linux % ls -l
```

```
total 4112
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:47 GPS
```

```
-rwx-----@ 1 jm61  staff 2095972 23 Apr 2022 GPSC33_reference.fasta
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:43 JUNO
```

```
-rwx-----@ 1 jm61  staff   7123 18 Aug 17:01 serotypes_GPS_resources.txt
```

-a show all files (including hidden files)

```
jm61@mib116462s Intro_to_linux % ls -a
```

```
. GPSC33_reference.fasta
```

```
.. JUNO
```

```
.DS_Store serotypes_GPS_resources.txt
```

```
GPS
```

-t sort based on last modified time

```
jm61@mib116462s Intro_to_linux % ls -t
```

```
GPS serotypes_GPS_resources.txt
```

```
JUNO GPSC33_reference.fasta
```

-lt creates a long list ordered by time

```
jm61@mib116462s Intro_to_linux % ls -lt
```

```
total 4112
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:47 GPS
```

```
drwxr-xr-x  3 jm61  staff   96 22 Dec 03:43 JUNO
```

```
-rwx-----@ 1 jm61  staff   7123 18 Aug 17:01 serotypes_GPS_resources.txt
```

```
-rwx-----@ 1 jm61  staff 2095972 23 Apr 2022 GPSC33_reference.fasta
```

pwd

Will return current working directory's name

```
jm61@mib116462s Intro_to_linux % pwd  
/Users/jm61/bioinformatics/Intro_to_linux
```

cd

Change directory. It is used for changing the working directory

Example: **cd** GPS

```
jm61@mib116462s Intro_to_linux % cd GPS  
jm61@mib116462s GPS %
```

We are now in the “GPS” directory. Command “**cd ..**” takes you out from the current directory.

```
jm61@mib116462s Intro_to_linux % cd GPS  
jm61@mib116462s GPS % cd ..  
jm61@mib116462s Intro_to_linux %
```

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

mkdir

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

Example: **mkdir** Practice

ls (to print the directories in Intro_to_linux. Practice should now be listed)

```
jm61@mib116462s Intro_to_linux % mkdir practice  
jm61@mib116462s Intro_to_linux % ls  
GPS practice  
GPSC33_reference.fasta serotypes_GPS_resources.txt  
JUNO
```

rmdir

Remove directory. This command removed an empty directory

Example: **rmdir** Practice

ls (to print the files and directories in Intro_to_linux. Practice should be removed and therefore not listed)

```
jm61@mib116462s Intro_to_linux % rmdir practice
```

```
jm61@mib116462s Intro_to_linux % ls
```

```
GPS JUNO
```

```
GPSC33_reference.fasta serotypes_GPS_resources.txt
```

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

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** s.pneumo.txt

ls (to print the files and directories in Intro_to_linux. s.pneumo.txt should be included and therefore listed)

```
jm61@mib116462s Intro_to_linux % touch s.pneumo.txt
```

```
jm61@mib116462s Intro_to_linux % ls
```

```
GPS s.pneumo.txt
```

```
GPSC33_reference.fasta serotypes_GPS_resources.txt
```

```
JUNO
```

rm

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

Example: **rm** s.pneumo.txt

ls (to print the files and directories in Intro_to_linux. s.pneumo.txt should be removed and therefore not listed)

```
jm61@mib116462s Intro_to_linux % rm s.pneumo.txt
```

```
jm61@mib116462s Intro_to_linux % ls
```

GPS JUNO

GPSC33_reference.fasta serotypes_GPS_resources.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:

```
touch s.pneumo.txt
```

```
cp s.pneumo.txt s.pneumo2.txt
```

ls (to print the files and directories in Intro_to_linux. s.pneumo.txt and s.pneumo2.txt should be listed)

```
jm61@mib116462s Intro_to_linux % touch s.pneumo.txt
```

```
jm61@mib116462s Intro_to_linux % cp s.pneumo.txt s.pneumo2.txt
```

```
jm61@mib116462s Intro_to_linux % ls
```

GPS s.pneumo.txt

GPSC33_reference.fasta s.pneumo2.txt

JUNO serotypes_GPS_resources.txt

To copy directories, use the “-r” option.

mv

To move file or directory or to rename a file

Example:

```
mv s.pneumo.txt GPS/ (moves s.pneumo-file to GPS directory.)
```

ls (to print the files and directories in Intro_to_linux. s.pneumo.txt should not be there as it's been moved to GPS)

```
jm61@mib116462s Intro_to_linux % mv s.pneumo.txt GPS/
```

```
jm61@mib116462s Intro_to_linux % ls
```

```
GPS s.pneumo2.txt
```

```
GPSC33_reference.fasta serotypes_GPS_resources.txt
```

```
JUNO
```

cd to GPS folder and **ls**

```
jm61@mib116462s Intro_to_linux % cd GPS
```

```
jm61@mib116462s GPS % ls
```

```
s.pneumo.txt
```

mv s.pneumo.txt s.agalactiae.txt (renames s.pneumo.txt to s.agalactiae.txt)

ls (to print the files and directories in GPS. s.agalactiae.txt should have replaced s.pneumo.txt)

```
jm61@mib116462s GPS % mv s.pneumo.txt s.agalactiae.txt
```

```
jm61@mib116462s GPS % ls
```

```
s.agalactiae.txt
```

In

Link. To make links to files/directories.

There are two types of links:

- **Symbolic links** (also known as symlink or soft link) - Refer to a symbolic path indicating the abstract location of another file.
- **Hard link** - Refer to the specific location of physical data.

Soft links are created with the **ln -s** command. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).

Command to create a Soft link is:

```
ln -s [original filename] [link name]
```

For example, the following would create a soft link named **practice** to the file named **GPSC33_reference.fasta** in the **'/Users/jm61/bioinformatics/Intro_to_linux'** directory, both in the current directory.

Example: **ln -s /Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta practice**

ls -l (to print the files and directories in GPS. practice should be listed in this directory)

```
jm61@mib116462s GPS % ln -s
/Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta practice

jm61@mib116462s GPS % ls -l

total 0

lrwxr-xr-x  1 jm61  staff  64 22 Dec 03:56 practice ->
/Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta

-rw-r--r--  1 jm61  staff   0 22 Dec 03:53 s.agalactiae.txt
```

It is recommended to create links instead of copying data into various directories to save space.

Hard links are more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.

The command to create a hard link is

ln [original filename] [link name]

For example, the following would create a hard link named **practice2** to the file named **GPSC33_reference.fasta** in the **'/Users/jm61/bioinformatics/Intro_to_linux'** directory, both in the current directory.

Example: **ln /Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta practice2**

ls -l (to print the files and directories in GPS. practice should be listed in this directory)

```
jm61@mib116462s GPS % ln
/Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta practice2

jm61@mib116462s GPS % ls -l

total 4096

lrwxr-xr-x  1 jm61  staff   64 22 Dec 03:56 practice ->
/Users/jm61/bioinformatics/Intro_to_linux/GPSC33_reference.fasta

-rwx-----@ 2 jm61  staff 2095972 23 Apr 2022 practice2

-rw-r--r--  1 jm61  staff    0 22 Dec 03:53 s.agalactiae.txt
```

Further reading on symbolic links:

<https://www.cyberciti.biz/faq/creating-soft-link-or-symbolic-link/>  Links to an external site.

<https://www.geeksforgeeks.org/soft-hard-links-unixlinux/>  Links to an external site.

Viewing file content

Move to Intro_to_linux directory. We will use the GPSC33_reference.fasta file.

cat

Concatenate. Combine files and prints on the screen

Example:

cat GPSC33_reference.fasta

```
jm61@mib116462s Intro_to_linux % cat GPSC33_reference.fasta
>12837_1#32
gaagataactaagttatatctgcacctctccttgaaaatctactgaaaacagtggtgaaaa
catcgtttcagtagggcttcttagtgcacccttttttgaaatagtttcaaactttg
agcggggggcgcttgacattttaccactagactttttaacatattgtctaatttttt
ataaggaggaaatatgcaagaaagtaacaaacgcttaaaaacaaagcgaactattgaaaa
tgctatggtacaattactgatggaacagccatttgataaaatttctactgtcaaattagt
agaaaaagccggaattagtcgtccagcttctatactcactataaggataagtatgatat
gattgagcactatcaaagcaagctattccatacatttgaatatattttcaaaaacatgc
tcatcacaaaagagacgctattttagaagttttgaatatctagagtcagaaccacttct
ggctgcccttcttctgaaaatgggactaaagaaatccaaaatttcttacgaaataaact
tcatatcatgcttagtacagatttacaaaagcgatttatgcaactgaatctaaataccac
tgaattagaatatagtagcatctatctaactcatgcacttttgggtgtctgccaacttg
gattgcacatggaaaaaaagaaagtctcaagaaataacagacttccttatgaaaatgct
tggtgatacaacttggtgatacaaatgatacaaaaagaggaacacagctgtgttcctt
ttatctatactccgccagtaggactcgaacctacgacatcatgattaacagtcatgcgct
actaccaactgag
....
```

more/less

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

Example: **more** GPSC33_reference.fasta

```
jm61@mib116462s Intro_to_linux % more GPSC33_reference.fasta
```

```
>12837_1#32
```

```
gaagataactaagttatatctgcacctctccttgaaaatctactgaaaacagtgtagaaa  
catcgtttcagtagggcttcttagtgaccccttttttgaaatagtttcaaactttg  
agcggggggcgcttgacattttaccactagactttttaaacatattgtctaattttt  
ataaggaggaaatatgcaagaaagtaacaaacgcttaaaaacaaagcgaactattgaaaa  
tgctatggtacaattactgatggaacagccatttgataaaatttctactgtcaaattag  
agaaaaagccggaattagtcgttccagcttctatactcactataaggataagtatgatat  
gattgagcactatcaaagcaagctattccatacatttgaatatattttcaaaaacatgc  
tcatcacaaaagagacgctattttagaagttttgaatatctagagtcagaaccacttct  
ggctgcccttcttctgaaaatgggactaaagaaatccaaaatttcttacgaaataaact  
tcatatcatgcttagtacagatttacaaaagcgatttatgcaactgaatctaaataccac  
tgaattagaatatagtagcatctatctaactcatgcacttttgggtgtctgccaaacttg  
gattgcacatggaaaaaaagaaagtctcaagaaataacagacttccttatgaaaatgct  
tggtgatacaacttggtgatacaaattgatacaaaaagaggaacacagctgtgttcctt  
ttatctatactccgccagtaggactcgaacctacgacatcatgattaacagtcatgcgct  
actaccaactgagctatggcggataaaaatagtcggtacgggattcgaacccgtgttaccg  
GPSC33_reference.fasta
```

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

Example: **less** GPSC33_reference.fasta

>12837_1#32

```
gaagataactaagttatatctgcacctctccttgaaaatctactgaaaacagtggtgaaa
catcgtttcagtagggcttcttagtgacccttttttgaaatagtttcaaactttg
agcggggggcgcttgacattttaccactagactttttaacatattgtctaattttt
ataaggaggaaatatgcaagaaagtaacaaacgcttaaaaacaaagcgaactattgaaa
tgctatggtacaattactgatggaacagccatttgataaaatttctactgtcaaattag
agaaaaagccggaattagtcgttccagcttctatactcactataaggataagtatgat
gattgagcactatcaaagcaagctattccatacatttgaatatattttcaaaaacatgc
tcatcacaaaagagacgctattttagaagttttgaatatctagagtcagaaccacttct
ggctgcccttcttctgaaaatgggactaaagaaatccaaaatttctacgaaataaact
tcatatcatgcttagtacagatttcaaaaagcgatttatgcaactgaatctaaataccac
tgaattagaatatagtagcatctatctaactcatgcacttttgggtgtctgccaaactg
gattgcacatggaaaaaaagaaagtctcaagaaataacagacttccttatgaaaatgct
tggtgatacaacttggtgatacaaattgatacaaaaagaggaacacagctgtgttcctt
ttatctatactccgccagtaggactcgaacctacgacatcatgattaacagtcatgcgct
actaccaactgagctatggcggataaaaatagtcggtacgggattcgaacccgtgttaccg
ccgtgaaaaggcgggtgtcttaacccttgaccaacggaccttctatctgtagcagatata
accattatatcaatttcttgctaattgtcaatcacttttgagatttttctctaaaatat
:
```

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** GPSC33_reference.fasta


```
jm61@mib116462s Intro_to_linux % head GPSC33_reference.fasta
>12837_1#32
gaagataactaagttatatctgcacctctccttgaaaatctactgaaaacagtggtgaaaa
catcgtttcagtagggcttcttagtgcacccttttttgaaatagtttcaaactttg
agcggggggcgcttgacattttaccactagactttttaaacatattgtctaatttttt
ataaggaggaaatatgcaagaaagtaacaaacgcttaaaaacaaagcgaactattgaaaa
tgctatggtacaattactgatggaacagccatttgataaaatttctactgtcaaattagt
agaaaaagccggaattagtcgttccagcttctatactcactataaggataagtatgatat
```

Example: **tail** GPSC33_reference.fasta

```
jm61@mib116462s Intro_to_linux % tail GPSC33_reference.fasta
tccaacaagattcagtcattgtcattctcggaacttctggctcgggaaattcgaaacctgg
aaaaaatgatcaaggaaattgataaggccattgaagatatggtggaaccattcctgaat
atcaatgcctgacctccatacctggggtcggttaagggtctacgctgctggtatcattgccg
aaattggacaaatcgaacgctttaagaccatcctcaggtcgcaaaatttgctggtctga
attggagagaaaagcaatctggttaactcaaactcacaacatacttcgcttgtaataggg
gcaatcggtatctgcgctactacctggtgaagccgccaactcggttaaggcgctatgatg
acgaatacaaggatttctataagaagaaataccatgaagtacctaacaaccaacacaaac
```

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 non-graphical file editors, “nano”

nano (earlier called pico) is very much like any graphical editor without a mouse. All commands are executed through the use of keyboards, using the <CTRL> key modifier. It can be used to edit virtually any kind of text file from the command line.

nano without a filename gives you a standard (blank) nano window.

At the bottom of the screen, there are commands with a symbol in front. The symbol tells you that you need to hold down the Control (Ctrl) key, and then press the corresponding letter of the command you wish to use.

For example:

Ctrl+X will exit nano and return you to the command line.

Nano Quick Reference

Ctrl+X: Exit the editor. If you've edited text without saving, you'll be prompted as to whether you really want to exit.

Ctrl+O: Write (output) the current contents of the text buffer to a file. A filename prompt will appear; press Ctrl+T to open the file navigator shown above.

Ctrl+R: Read a text file into the current editing session. At the filename prompt, hit Ctrl+T: for the file navigator.

Ctrl+K: Cut a line into the clipboard. You can press this repeatedly to copy multiple lines, which are then stored as one chunk.

Ctrl+J: Justify (fill out) a paragraph of text. By default, this reflows text to match the width of the editing window.

Ctrl+U: Uncut text, or rather, paste it from the clipboard. Note that after a justify operation, this turns into unjustify.

Ctrl+T: Check spelling.

Ctrl+W: Find a word or a phrase. At the prompt, use the cursor keys to go through previous search terms, or hit Ctrl+R to move into replace mode. Alternatively you can hit Ctrl+T to go to a specific line.

Ctrl+C: Show current line number and file information.

Ctrl+G: Get help; this provides information on navigating through files and common keyboard commands.

Getting help in Linux

Most of the Linux commands have manual pages. To access them, use “**man**” or “**info**” command. The manual page gives a detailed explanation of the command, all available options and sometimes, also provides examples. For example, to get the manual page for ls command type “**man ls**”

Please explore manual pages of all the above commands for available options.

Some useful commands

ls: Lists the contents of the current directory

mkdir: Makes a new directory

mv: Moves or renames a file

cp: Copies a file

rm: Removes a file

cat: Concatenates files

more: Displays the contents of a file one page at a time

head: Displays the first ten lines of a file

tail: Displays the last ten lines of a file

cd: Changes current working directory

pwd: Prints working directory

find: Finds files matching an expression

grep: Searches a file for patterns

wc: Counts the lines, words, characters, and bytes in a file

kill: Stops a process

jobs: Lists the processes that are running

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** GPSC33_reference.fasta


```
jm61@mib116462s Intro_to_linux % cut -c 1-10 GPSC33_reference.fasta
>12837_1#3
gaagataact
catcgtttct
agcgggggggc
ataaggagga
tgctatggta
agaaaaagcc
gattgagcac
tcatcacaaa
...
```

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**: field number

We have a file named “serotypes_GPS_resources.txt” with all the count, serotype, Accession Number, Comments and Reference. These fields are separated by “,” symbol.

[serotype_GPS_resources.txt was copied from the web page
<https://www.pneumogen.net/gps/serotypes.html>  Links to an external site.]

head serotypes_GPS_resources.txt

```
jm61@mib116462s Intro_to_linux % head serotypes_GPS_resources.txt
```

```
Count,Serotype,Accession_Number,Comments,Reference
```

```
"1,1,CR931632,-,Bentley et al 2006 PloS Genet"
```

```
"2,2,CR931633,-,Bentley et al 2006 PloS Genet"
```

```
"3,3,CR931634,-,Bentley et al 2006 PloS Genet"
```

```
"4,4,CR931635,-,Bentley et al 2006 PloS Genet"
```

```
"5,5,CR931636,-,Bentley et al 2006 PloS Genet"
```

```
"6,6A,CR931638,-,Bentley et al 2006 PloS Genet"
```

```
"-,6E(6A),-,Genetic variant of 6A,Park et al 2017 Genome Announc?"
```

```
"7,6B,CR931639,-,Bentley et al 2006 PloS Genet"
```

```
"-,6E(6B),KU168827,Genetic variant of 6B,Burton et al 2016 J Clin Microbiol"
```

To get only Accession Number

```
cut -d “,” -f3 serotypes_GPS_resources.txt
```

```
jm61@mib116462s Intro_to_linux % cut -d "," -f3 serotypes_GPS_resources.txt
```

Accession_Number

CR931632

CR931633

CR931634

CR931635

CR931636

CR931638

-

CR931639

KU168827

EF538714 to EF538718

GQ848645 to GQ848646

KF597302

CR931643

CR931640

CR931641

CR931642

-

Here “,” is a delimiter and the third field is our choice of interest.

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

To sort serotypes_GPS_resources.txt based on their serotype

sort -t “,” -nk2 serotypes_GPS_resources.txt

```
jm61@mib116462s Intro_to_linux % sort -t “,” -nk2 serotypes_GPS_resources.txt
```

"Count,Serotype,Accession_Number,Comments,Reference"

"-,Group II nonencapsulated?,-,do not produce capsule due to novel genes in place of genes encoding capsule,Park et al 2012 mBio; Salter et al 2012 Microbiology"

"-,Group I nonencapsulated?,-,do not produce capsule due to mutations or deletion of genes encoding capsule?,Hathaway et al 2014 J Bacteriology"

"1,1,CR931632,-,Bentley et al 2006 PloS Genet"

"2,2,CR931633,-,Bentley et al 2006 PloS Genet"

"3,3,CR931634,-,Bentley et al 2006 PloS Genet"

"4,4,CR931635,-,Bentley et al 2006 PloS Genet"

"5,5,CR931636,-,Bentley et al 2006 PloS Genet"

"10,6H,KF597302,-,Park et al 2015 Clin Vaccine Immunol?"

"6,6A,CR931638,-,Bentley et al 2006 PloS Genet"

"-,6E(6A),-,Genetic variant of 6A,Park et al 2017 Genome Announc?"

"-,6E(6B),KU168827,Genetic variant of 6B,Burton et al 2016 J Clin Microbiol"

"7,6B,CR931639,-,Bentley et al 2006 PloS Genet"

"8,6C,EF538714 to EF538718,-,Park et al 2007 Carbohydr Res; Park et al 2007 Infect Immun."

"9,6D,GQ848645 to GQ848646,-,Bratcher et al 2010 Microbiology; Bratcher et al 2009 Microbiology?"

"11,7F,CR931643,-,Bentley et al 2006 PloS Genet"

"12,7A,CR931640,-,Bentley et al 2006 PloS Genet"

"13,7B,CR931641,-,Bentley et al 2006 PloS Genet"

"14,7C,CR931642,-,Bentley et al 2006 PloS Genet"

"15,7D,-,-,Kjeldsen et al 2018 Carbohydr Res"

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

To get all isolates from only “Bentley et al” from serotypes_GPS_resources.txt

grep Bentley serotypes_GPS_resources.txt


```
jm61@mib116462s Intro_to_linux % grep Bentley serotypes_GPS_resources.txt
```

```
"1,1,CR931632,-,Bentley et al 2006 PloS Genet"  
"2,2,CR931633,-,Bentley et al 2006 PloS Genet"  
"3,3,CR931634,-,Bentley et al 2006 PloS Genet"  
"4,4,CR931635,-,Bentley et al 2006 PloS Genet"  
"5,5,CR931636,-,Bentley et al 2006 PloS Genet"  
"6,6A,CR931638,-,Bentley et al 2006 PloS Genet"  
"7,6B,CR931639,-,Bentley et al 2006 PloS Genet"  
"11,7F,CR931643,-,Bentley et al 2006 PloS Genet"  
"12,7A,CR931640,-,Bentley et al 2006 PloS Genet"  
"13,7B,CR931641,-,Bentley et al 2006 PloS Genet"  
"14,7C,CR931642,-,Bentley et al 2006 PloS Genet"  
"16,8,CR931644,-,Bentley et al 2006 PloS Genet"  
"17,9A,CR931645,-,Bentley et al 2006 PloS Genet"  
"19,9N,CR931647,-,Bentley et al 2006 PloS Genet"  
"20,9V,CR931648,-,Bentley et al 2006 PloS Genet"  
"21,10F,CR931652,-,Bentley et al 2006 PloS Genet"
```

Other text processing commands worth looking at are: **tr**, **rev**, **sed**, **uniq**, **wc** and **paste**.

I/O control in Linux

When you run a command the output will be sent to standard out (stdout) i.e terminal. However, we can send the stdout to a file using ">" redirection. This will redirect the standard output to a file.

ls > list

ls

cat list

```
jm61@mib116462s Intro_to_linux % ls > list
```

```
jm61@mib116462s Intro_to_linux % ls
```

GPS list

GPSC33_reference.fasta s.pneumo2.txt

JUNO serotypes_GPS_resources.txt

```
jm61@mib116462s Intro_to_linux % cat list
```

GPS

GPSC33_reference.fasta

JUNO

list

s.pneumo2.txt

serotypes_GPS_resources.txt

This will create a new file called list with all the file names in the directory. Remember, if there is a file that exists with a name “**list**”, its content will be overwritten by the stdout. Instead, we can append to a file using “>>” redirection that will add new lines to the end of the file.

Another kind of output that is generated by programs is standard error. We have to use “**2>**” to redirect it.

ls /foo 2> error

```
jm61@mib116462s Intro_to_linux % ls
```

GPS list

GPSC33_reference.fasta s.pneumo2.txt

JUNO serotypes_GPS_resources.txt

error

To redirect stdout and stderr to a file, use “**&>**”

Pipes

Piping in Linux is a very powerful and efficient way to combine commands. Pipes (|) 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
```

```
jm61@mib116462s Intro_to_linux % sort -t "," -nk2 serotypes_GPS_resources.txt | head
```

```
"-,Group I nonencapsulatedÊ,-,do not produce capsule due to mutations or deletion of genes encoding capsuleÊ,Hathaway et al 2014 J Bacteriology"
```

```
"-,Group II nonencapsulatedÊ,-,do not produce capsule due to novel genes in place of genes encoding capsule,Park et al 2012 mBio; Salter et al 2012 Microbiology"
```

```
"Count,Serotype,Accession_Number,Comments,Reference"
```

```
"1,1,CR931632,-,Bentley et al 2006 PloS Genet"
```

```
"2,2,CR931633,-,Bentley et al 2006 PloS Genet"
```

```
"3,3,CR931634,-,Bentley et al 2006 PloS Genet"
```

```
"4,4,CR931635,-,Bentley et al 2006 PloS Genet"
```

```
"5,5,CR931636,-,Bentley et al 2006 PloS Genet"
```

```
"-,6E(6A),-,Genetic variant of 6A,Park et al 2017 Genome AnnouncÊ"
```

```
"-,6E(6B),KU168827,Genetic variant of 6B,Burton et al 2016 J Clin Microbiol"
```

We will be working on other examples during the course, where we use pipes to combine more than two commands.

Process control

Some commands take time to complete the job. For example, if you want to compress a huge file with the **gzip** command it might take a few minutes to finish the job. Until it finishes you cannot run any command. In such situations, it is better to run this command in the background by appending with "&" (we can also suspend a running job by **Ctrl+z** and type **bg**).

```
gzip serotypes_GPS_resources.txt &
```

Once it completes the task we get a message saying "Done". We can get a list of current jobs in the terminal by typing the "**jobs**" command. This will give you all the background jobs running in the current terminal. If you want to see all the running processes in the system, use "**top**". You can get user specific details on top using the "**-u**" option.

```
top - u $jm61!
```

“top” command prints a 12 column dynamic output. These columns are:

PID: Process ID, this is a unique number used to identify the process.

USER: The username of whoever launched the process

PR: Priority - The priority of the process. Processes with higher priority will be favoured by the kernel and given more CPU time than processes with lower priority. Oddly enough, the lower this value, the higher the actual priority.

NI: nice value: nice is a way of setting your process' priority.

VIRT: Virtual Memory Size (KiB): The total amount of virtual memory used by the process.

RES: Resident Memory Size (KiB): The non-swapped physical memory a task has used.

SHR: Shared Memory Size (KiB): The amount of shared memory available to a task, not all of which is typically resident. It simply reflects memory that could be potentially shared with other processes.

S: Process Status: The status of the task which can be one of:

- **D** = uninterruptible sleep

- **R** = running

- **S** = sleeping

- **T** = traced or stopped

- **Z** = zombie

%CPU: CPU Usage: The percentage of your CPU that is being used by the process. By default, top displays this as a percentage of a single CPU. On multi-core systems, you can have percentages that are greater than 100%. For example, if 3 cores are at 60% use, the top will show a CPU use of 180.

%MEM: Memory Usage (RES): A task's currently used share of available physical memory (RAM)

TIME+: CPU Time, hundredths: Total CPU time the task has used since it started.

COMMAND: Command Name or Command Line: To see the full command line that launched the process, start top with the -c flag

If you want to stop a running background job use the **“kill”** command.

kill 1234

This will kill the job with the process id 1234. As a user you can kill only your jobs. You do not have permission to run this command on other users' process ids.

Command line shortcuts

Up/Down arrows: Previous commands

!!: Rerun previous command

Tab: Auto complete

Tab+Tab: All available options

Ctrl+a: Move cursor to start of line

Ctrl+e: Move cursor to end of line

Ctrl+: Alternates between terminals

Ctrl+l: Clear screen ((or Command + k on Mac)

Ctrl+c: Terminates the running program

Ctrl+z: Suspends the running program

Ctrl+w: Removes a previous word

Ctrl+d: Logout

Ctrl+d (in a command): Removes a character

Ctrl+u: Removes till the beginning

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

rm 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