Linux command line
Part I

File system, manipulating files and directories

The very basics

Command prompt:
$ slava@slava-HPE-480t:~/Projects

Let's enter a random command
$ slava@slava-HPE-480t:~/Projects$ fjgkhlh
fjgkhlh: command not found

!Command history
Press arrow up ↑ and down ↓ to navigate command history (previous 1000 commands are remembered)

Use left ← and right → arrows to move cursor
Use del and backspace to edit the prompt text
$ slava@slava-HPE-480t:~/Projects$ dog cat bird # try to move the cursor and edit this line

Let’s try some simple commands
$ slava@slava-HPE-480t:~/Projects$ date # prints current date and time
Thu Feb 9 16:46:48 MST 2023

The very basics

Show the hard drive usage

```
slava@slava-HPE-480t:~/Projects$ df
Filesystem      1K-blocks      Used  Available Use% Mounted on
udev              8183832         0    8183832   0% /dev
tmpfs             1641524      2288    1639236   1% /run
/dev/sdb1       944723660  708669060  188038576  80% /
tmpfs             8207620        20    8207600   1% /dev/shm
tmpfs                5120         4       5116   1% /run/lock
tmpfs             8207620         0    8207620   0% /sys/fs/cgroup
…
```

Who is logged in?

```
slava@slava-HPE-480t:~/Projects$ whoami
slava
```

What is my current directory?

```
slava@slava-HPE-480t:~/Projects$ pwd
/home/slava/Projects
```

The file system

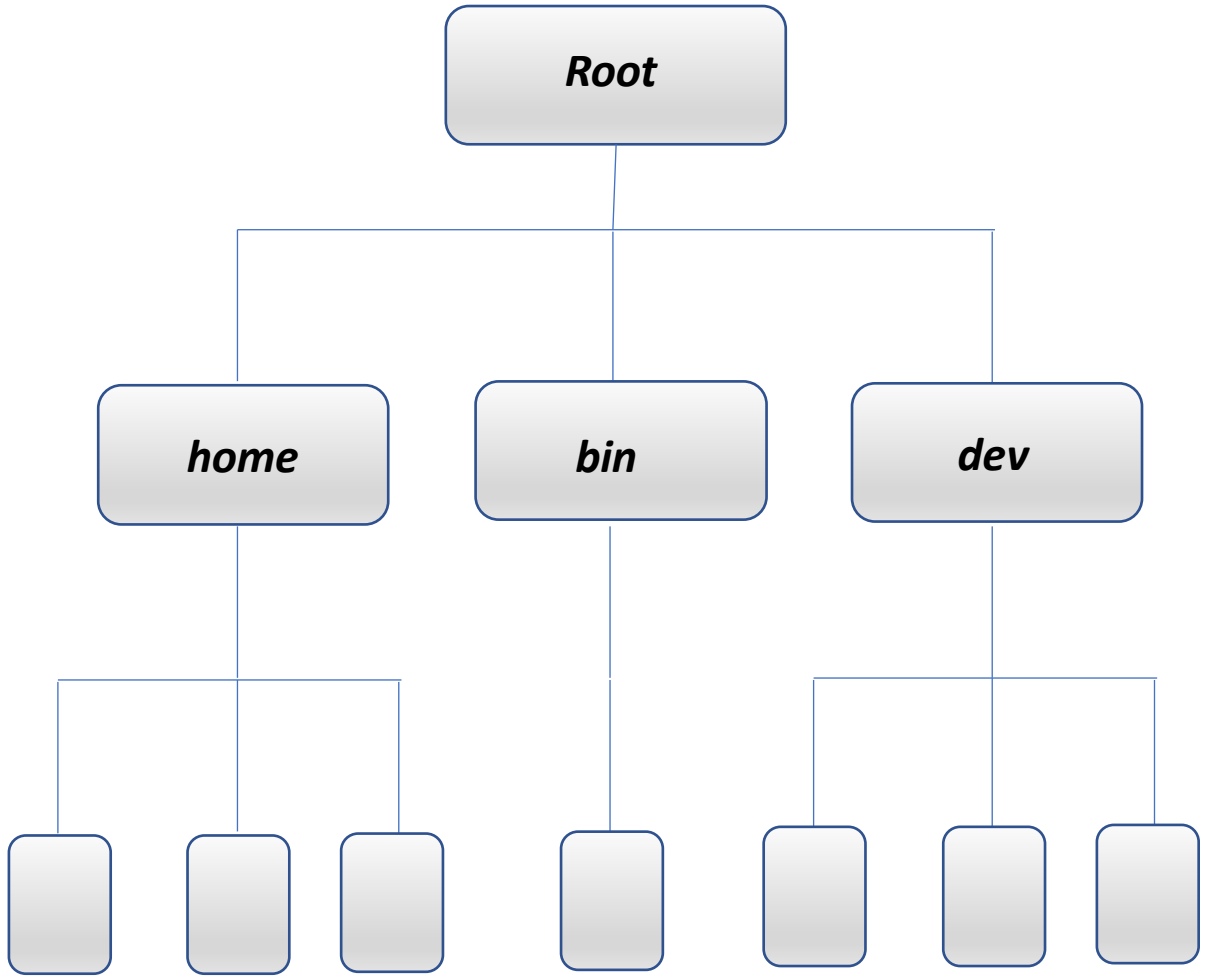
Some terminology

Current working directory – the directory we are in.

Parent directory is one level above the current directory

Root directory is the first directory at the root of the directory tree

Home directory is a directory that belongs to a particular user. It is located in **home/** folder and is a directory that user logs into. Personal files, software, and configuration files are located here.



Absolute and relative pathnames

To access a file or directory we need to tell the operational system (OS) their exact location.

The location can be expressed as **absolute** or **relative** paths.

1. Absolute path starts with the **root** directory shown as **/** and follows the full path down the directory tree. For example: **"/home/Project/TB_var_cal/".**
2. Relative path is relative to your current working directory (CWD). The special notation for CWD is **".".** The directory on level up from the CWD has a notation of **".."**.

Use **cd** command to move between directories

$ cd / # change to root directory

$ cd /home/<username> # change to your home directory using absolute path

$ cd /usr/bin/

$ cd ../ # change one level up from bin directory
$ **pwd** # to print current directory

Absolute and relative pathnames

Some examples of directory navigation with some helpful shortcuts:

$ cd . # you will stay in the same directory

$ cd ../../ # you will move two levels up

$ cd ~ # tilde is a shortcut to your home directory

$ cd /home/<username> # change to the home directory of the username stated in the command

$ cd - # change to the previous directory

NOTE: Use **TAB** to expand directory names, file names, and commads!

Always use **TAB**, no need type full names of files, folders or commands.

Linux directories explained

$ cd / # change to the root directory

$ ls -l # view the contents of the root directory

Use **ls** to explore Linux directories listed in a table below

Directory	Contents
/bin	Contains binaries (programs) of the essential commands necessary to load and run OS
/boot	Contains Linux kernel, drivers needed at boot times, the boot loader, configuration file for boot loader *grub/grub.conf*, and */boot/vmlinuz* configation file for linux kernel
/dev	A place where Linux keeps a list of all devices (device nodes)
/etc	Harbors a variety of system-wide configuration files and shell scripts that start system services at boot.
/home	Every user is given a directory in /home, without a root access a user can only make changes within the home directory, this is to protect the system from user activity
/lib	Contains files for the libraries shared between core system software
/lost+found	Every formatted partition or device using Linux system will have this folder to store data in the case of partial recovery from the corruption event. This directory should be empty unless the corruption event happened

Linux directories explained

Directory	Contents
/media	This directory contains mount points for external media: USBs, external hard drives, CD-ROM, etc.
/mnt	Present in older Linux systems and contains mount point for the devices that were mounted manually
/opt	Contains “optional” software, normally it holds commercial software installed on the system
/proc	Holds a virtual file system maintained by Linux kernel. Virtual files contained in /**proc** provide view into how kernel views the system. These files are human readable and provide a wealth of information o the system and processes
/root	A directory for the administrator (root) account
/sbin	This directory contains “system” binaries. The programs that perform vital system tasks reserved for the superuser
/tmp	Contains temporary files created by various programs
/usr	Contains all the programs and file used by regular users

Linux directories explained

Directory	Contents
/usr/bin	Contains executable programs installed by regular users
/usr/lib	Contains shared libraries for the programs in **/usr/bin**
/usr/local	This directory is intendent for the programs that are not included with the distribution but are meant for system-wide use. Programs compiled from the source code will be installed in **/usr/local/bin**. You will need administrator access to install programs here.
/usr/sbin	Contains additional system administration programs
/usr/share	Contains all shared files used by programs in **/usr/bin**. This includes default configuration files, icons, screen backgrounds, sound files, etc.
/usr/share/doc	Documentations for packages installed on the system
/var	This directory stores the data that is likely to change, such as various databases, spool files, user mail, etc.
/var/log	Contains *log* files that records system-wide activity. These files are important to monitor changes and processes in the system. On some systems you need to be a superuser to view these files

Explore files and folders

Use **ls** command to view the content of the directories.

ls is probably the most used command in Linux.

Let’s change to home directory
$ cd ~

$ ls # View the content of the current directory

You can list any directory by specifying absolute or relative path
$ ls /usr # view **usr/** directory

$ ls ../ # view a directory one level up

$ ls ../../ # view a directory 2 levels up

Explore files and folders

Using option with commands:
$ ls -l ~ # -l option will change the output to long format

The structure of Linux commands is composed of **options** and **arguments**.

Options, like **–l**, modify the behavior of the commands and arguments are items that serve as inputs.

ls – command
-l – option
~ – argument

Short options start with **‘-’** and are symbolized by a single letter, for example **‘-l’**

Long options start **‘--’** and are followed by a word, for example **‘--reverse’**

Commands can have multiple options and arguments:
$ ls -l -t ~ # print the contents of **~ (home)** directory in long format and sorted by the date of creation

$ ls -lt ~ # this will yield the same result as above as short options can be combined

Explore files and folders

Short and long can be used together:
$ ls –lt --reverse ~ # this command will list the contents of the **home** directory in long format and sorted by time of creation in reverse.

The **long format** of ls command explained:

```
-rwxr-xr-x  1 slava slava      504 Feb 21  2018  backup.sh
drwxr-xr-x  5 slava slava     4096 Mar 19  2020 'Calibre Library'
drwxrwxr-x  4 slava slava     4096 Nov 24  2017  clinEff
```

Field	Meaning
-rwxr-xr-x	Access rights to the file
1	File's number of hard links
slava	The username of file's owner
slava	The name of the user group that owns the file
504	Size of the file in bytes
Feb 21 2018	Date of file's creation
backup.sh	Name of the file

Viewing file contents

First, let's download a file to practice:

Change to home directory
$ cd ~

Create a new directory called "sandbox"
$ mkdir sandbox # **mkdir** command creates directories

Change to *sandbox/*
$ cd sandbox

Download a sample text file from our courses github page
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/divine_comedy.txt # **wget** is a command for non-interactive download of files from the web. **wget** works in the background and continues the download even after user had disconnected from the system.

Viewing file contents

Sometimes it is useful to check the file type. Note, that, unlike in Windows, file extensions are not necessary in Linux.

```
$ file divine_comedy.txt # file command will output information about the type of the file
divine_comedy.txt: UTF-8 Unicode text
```

There are many types of files encountered in bioinformatics.

They can be text, such as SAM, FASTA, FASTQ, VCF or binary, like BAM.

We can fully print a file to the screen with **cat** (concatenate):

```
$ cat divine_comedy.txt
```

We can use **cat** to print multiple files. Download a poem Darkness by Lord Byron from our github repository.

```
$ wget https://raw.githubusercontent.com/slavailn/bioinf_training/main/darkness_byron.txt
$ cat divine_comedy.txt darkness_byron.txt # This command will print both files
```

Use **head** command to print top lines of the file:

```
$ head divine_comedy.txt # print top 10 lines by default
```

Viewing file contents

Specify a number of lines to print:

```
$ head -n 5 divine_comedy.txt # print first 5 lines
```

We can also print all, but a certain number of lines at the end:

```
$ head -n -5 divine_comedy.txt # print all, but the last 5 lines
```

head command can be applied to multiple files:

```
$ head divine_comedy.txt darkness_byron.txt
```

Use **tail** to print last lines of the file:

```
$ tail divine_comedy.txt # print last 10 lines of the file by default
```

Print last N lines from the end:

```
$ tail –n 10 divine_comedy.txt
```

Print N lines till the end of the file

```
$ tail -n +20 divine_comedy.txt # print last 20 lines
```

Tail can be used on multiple files:

```
$tail divine_comedy.txt darkness_byron.txt
```

Viewing file contents

The contents of a text file can be explored in detail using **less** utility.

```
$ less divine_comedy.txt
```

Use ↑ and ↓ keys to scroll back and forward the text.

Page-Up or **b** – scroll back one page

Page-Up or **space** – scroll forward

G – move to the end of the file

g – move to the beginning of the file

/<characters> - search forward for the occurrence of characters

n – search for the next occurrence of the previous search

h – display help screen

q – quit **less**

Manipulating files and directories

Directory and file manipulation commands:

cp – copy file or directory

mv – move a file or a directory to another location

mkdir – create directory

rm – remove file or a directory

ln – create hard and symbolic links

touch – create an empty file

Manipulating files and directories

Wildcards:

Wildcards are absolutely vital for the work with the command line.

They provide a way to select and change multiple items that fit certain criteria.

Wildcard	Meaning
*	Match any characters
?	Match any single character
[characters]	Matches any character that is a member of a set of characters
[!characters]	Matches any character that is not a member of a set of characters
[[:class:]]	Matches any character that is a member of a specified class

Manipulating files and directories

Wildcards:

Character classes used in constructing the wild cards

Character class	Meaning
[:alnum:]	Matches any alphanumeric character
[:alpha:]	Matches any alphabetic character
[:digit:]	Matches numeral
[:lower:]	Matches any lowercase letter
[:upper:]	Matches any uppercase letter

Alphanumeric – numbers and letters

Alphabetic – only letters

Numeric – only digits

Manipulating files and directories

Wildcard examples

Pattern	Match
*	Select all files
*.fasta	Select all files that end with *.fasta*
Sample[[:digit:]]*.txt	Select all files that start with *Sample* followed by any number of digits and followed by *.txt*
??_??.txt	Select files with 2 digits separated by dash and followed by *.txt*
[abc]*	Select all files that begin with *a, b or c*
[[:upper:]]	Select files that begin with the uppercase letter
[![:digit:]]*	Select files that do not start with a numerical
*[[:lower:]123]	Any file that ends with a lowercase letter or *1,2,3* numerals

Manipulating files and directories

Create directories

Create a single directory
$ mkdir dir1

Create multiple directories
$ mkdir dir1 dir2 dir3

Copy file and directories

Copy *item1* to *item2*, where item is a file or a directory
$ cp item1 item2

Copy multiple items into a directory
$ cp item… directory

Manipulating files and directories

Useful options for **cp**

Copy file with all permissions and attributes
$ cp -a item1 item2 # Normally the item is copied with default attributes of the user performing copy

Interactive copy, ask a user before overwriting a file
$ cp -i item* dir # Default copy will silently overwrite the files

Recursively copy the directory
$ cp -r dir1 dir2 # Copy a directory with all of its contents, this option is required to copy directories

Update while copying
$ cp -u item* dir1 # This option is useful when you need to copy only the files that exist or were recently modified compared to the files in the target directory

Generate messages while copying
$ cp -v item* dir1 # verbose mode for copying

Manipulating files and directories

Let's run some examples:

Change to ***sandbox/*** directory.

In ***sandbox*** directory create ***doc scripts reads*** folders
$ mkdir doc scripts reads

Change to ***doc*** and create some empty sample file
$ cd doc
$ touch data_report.txt samples.txt workflow
$ ls -lh # always check the results of operations

Create a backup copy for *data_report.txt*
$ cp data_report.txt data_report_copy.txt

Copy all of the *.txt* files from ***doc*** to ***scripts*** directory
$ cp *.txt ../scripts
$ ls -lh ../scripts # check the results

Copy *.txt* files from ***scripts*** back to ***doc*** in interactive mode
$ cp -i ../scripts/*.txt .

Manipulating files and directories

Copy examples continued:

Assuming that ***sandbox/*** is the working directory

Copy ***doc*** directory to ***scripts***

```
$  cp -r doc scripts
$ ls -lh scripts
```

Copy ***doc*** and ***scripts*** directories to ***reads***

```
$ cp doc scripts reads # This will generate an error since we did not use -r option
$ cp -r doc scripts reads # Now the command should work
$ ls -lh reads # Check the results
```

Copy all files with underscores (‘_’) from ***reads/doc*** to the working directory

```
$ cp reads/doc/*_* .
$ ls -lh
```

Manipulating files and directories

Moving file and directories - **mv** command

Use **mv** command to move or rename files or directories.

mv will move or rename an item depending on how it is used.

Rename *file1* to *file2*
$ mv file1 file2

Interactive renaming, asks for permission before overwriting a file if it already exists
$ mv -i file1 file2 # **mv** silently overwrites files without the interactive option

Move files to a directory
$ mv file1 file2 dir1 # the directory must already exist

Rename **dir1** to **dir2** if **dir2** does not exist
$ mv dir1 dir2

Move dir1 to dir2 if dir2 already exists
$ mv dir1 dir2

Manipulating files and directories

mv command examples

Change to ***sandbox/*** directory

Create files *sample1.txt* and *sample2.txt*
$ touch sample1.txt sample2.txt

Rename sample1.txt to sample2.txt
$ mv sample1.txt sample2.txt # Note that this command will remove the original sample2.txt without any warning
$ ls -lh # check the results

Try the same in interactive mode
$ touch sample1.txt sample2.txt
$ mv -i sample1.txt sample2.txt # safer behavior – the command will ask for permission before overwriting the files

Move multiple file in verbose mode
$ touch sample1.txt sample2.txt
$ mv -v sample?.txt doc

Manipulating files and directories

mv command examples

mv has an update mode activated with -u option
$ touch sample1.txt sample2.txt
$ mv -u sample?.txt doc # -u option will move the files only if they do not exist in the target directory or if they are more recent than the files in the target destination

---

Removing files and directories with **rm** command

rm item… , where item is one or more files or directories

Remove file
$ rm file1

Remove multiple files
$ rm file1 file2

Remove multiple files in interactive mode
$ rm -i file1 file2

Manipulating files and directories

Removing files and directories with **rm** command

Remove multiple directories
$ rm -r dir1 dir2 # -r (recursive) option is needed to remove directory
$ rm -r file1 dir1 dir2 # it is possible to combine files and directories when -r option is provided

Ignore non-existent items with -f (force) option
$ rm -rf file1 dir1 dir2 # will continue removal even if one of these items does not exist

Use -v (verbose) option to print a message when each item is removed
$ rm -rv file1 dir1 dir2

rm command examples

Change to ***sandbox/*** folder

Remove *data_report.txt* and *data_report_copy.txt* files
$ rm data_report.txt data_report_copy.txt
$ cd doc # change to **doc** directory that contains more practice files
$ ls -lh
$ rm -i sample* # remove all files that start with “sample” in interactive mode

Manipulating files and directories

rm command examples

Remove ***reads/*** directory one level up
$ rm -r ../reads

Try to remove **scripts/** directory, but specify a non-existent item
$ rm -r file1 ../scripts # this will generate an error message
$ rm -rf file1 ../scripts/ # add -f option to ignore non-existent file

Move one level up and remove doc directory in verbose mode
$ cd ../
$ rm -rv doc

Understanding the dangers of **rm** and **mv** commands

All items that are removed and overwritten are gone permanently and without warning.

Vital data files and scripts can be easily deleted.

It is easy to damage Linux itself when using **rm** with root access, in fact it is possible to delete the entire file system.

Back up the data and take extreme care when using rm, especially with the wildcards. For example rm *html will remove all *html* files, however rm * html will remove all files and then warn than "*html*" file does not exist.

Manipulating files and directories

Hard and soft file links

- A link is a symbolic connection or a pointer to a file.
- A link allows an access to a file from more than one directory, for example you can set up a link to a file in a restricted directory without allowing access to the directory itself
- By default, every file has a single hard link which gives the file its name
- Hard or soft links are not file copies; they point to the same file.
- If we change the file content, we will get the same output, no matter which link is used to access the file. If we change a copy, the content of the original file will remain the same.
- When hard link is removed, the file is not deallocated (permitted for overwriting) until all the hard links are removed.

Manipulating files and directories

Hard and soft file links

Hard link	Soft link
Cannot cross the filesystem boundaries, i.e. it can only reference the file on the same disk partition	Can cross file system boundaries (point to files on a different disk partition)
Cannot point to directories	Can point to directories
Has the same inode and permissions as the original file	Different inode number and permissions from the original file
If the file properties (for example, permissions) are updated, the properties of the hard link are updated as well	The properties of a soft links are not undated
If the original file is removed, its contents is still accessible through the hard link	If the original file is removed the soft link is broken

Manipulating files and directories

Hard and soft file links examples:

Change to **sandbox/** directory

Create test directories
$ mkdir doc scripts

Create file *workflow* in **doc** directory
$ touch workflow
$ echo variant call workflow > workflow # add some content to the with *echo* command

Create hard link to *workflow* file in scripts directory
$ cd ../scripts
$ ln ../doc/workflow work-hard
$ cat work-hard

Manipulating files and directories

Hard and soft file links examples:

Now, create a soft link to *workflow* file in scripts directory - **-s** option

```
$ ln  -s ../doc/workflow work-soft
$ cat work-soft # check the content of the link
```

Modify the content of the original file and check both links

```
$ echo rna-seq workflow >  ../doc/workflow
$ cat work-hard
$ cat work-soft
```

Let's modify the content of the hard link and see how the original file behaves

```
$ echo dna-seq workflow >  work-hard # we changed rna-seq to dna-seq
$ cat ../doc/workflow # check the original file
```

Let's do the same with a soft link and check the results

```
$ echo rna-seq workflow >  work-soft # we changed dna-seq back to rna-seq
$ cat ../doc/workflow # check the original file
```

Manipulating files and directories

Hard and soft file links examples:

Now, let's see what happens if we remove the original file
$ rm ../doc/workflow
$ cat work-hard # check the content of the link
$ cat work-soft # what was the difference?

We are done, let's remove **doc/** and **scripts/** directories
$ rm -r ../doc ../scripts