UNIVERSITY OF OXFORD

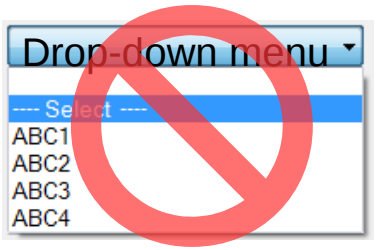
Command-line course part I

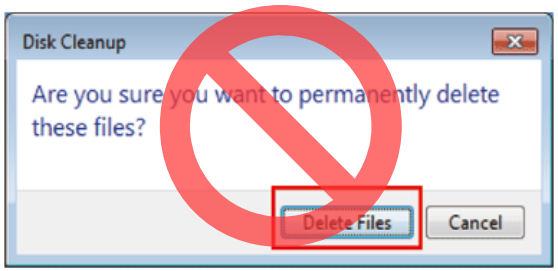
Silvia Salatino, PhD

14.10.2019 – Wellcome Centre for Human Genetics, Oxford

What is the command-line interface?

The ***command-line*** interface (sometimes also called "***command prompt***" or "***terminal***") is a way of interacting with the computer using only the keyboard.





Many of the programs and tools used in bioinformatics are designed to work only from command-line, so it's very important to get familiar with how the terminal works.

Although there are different types of terminals, all of them have an interface (called "***shell***") translating the text you type into meaningful commands that the computer can understand.

Today we'll focus on the most commonly used shell, ***BASH*** (developed in 1989!), which is the default one for Linux and MacOS systems. It can also work on Windows, but you have to manually install it.

BASH commands – introduction

BASH has hundreds of commands, but don't panic!

In most of the cases, you'll only use a handful of them in your day-to-day work (phew!)

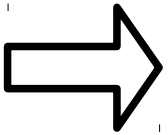
Most of them follow this simple ***general synthax***:

```
$ command option arguments
```

program you want to run ->
modify the program's behaviour ->
data passed to the program ->

If you're unsure about a specific command's synthax, you can type ***man*** followed by the command.

E.g.: `$ man ls`



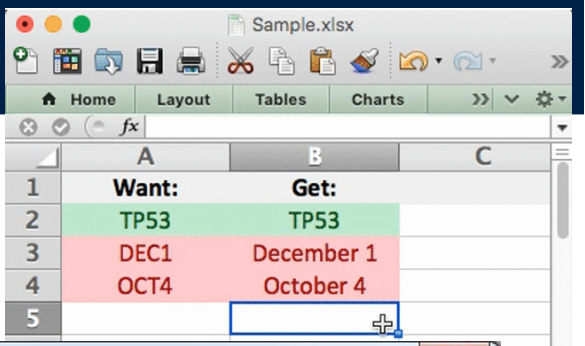
(use the arrows on your keyboard to scroll up and down the manual; then press ***q*** to exit when you're done)

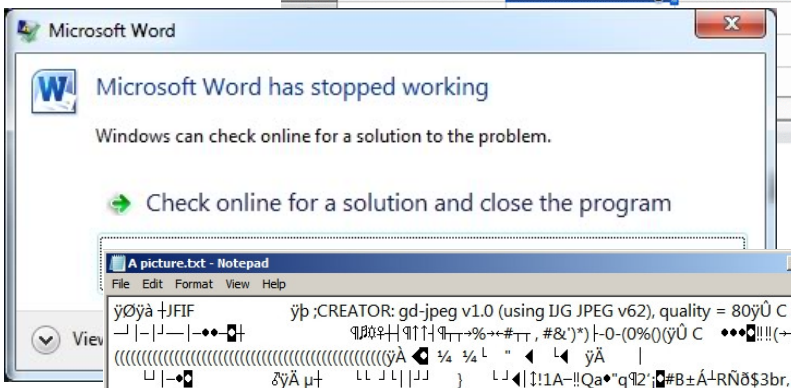
```
LS(1)                    User Commands                    LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the  FILEs  (the  current
       directory by default).  Sort entries alphabetically
       if none of -cftuvSUX nor --sort is specified.

       Mandatory  arguments  to long options are mandatory
       for short options too.

       -a, --all
              do not ignore entries starting with .

 Manual page ls(1) line 1 (press h for help or q to quit)
```

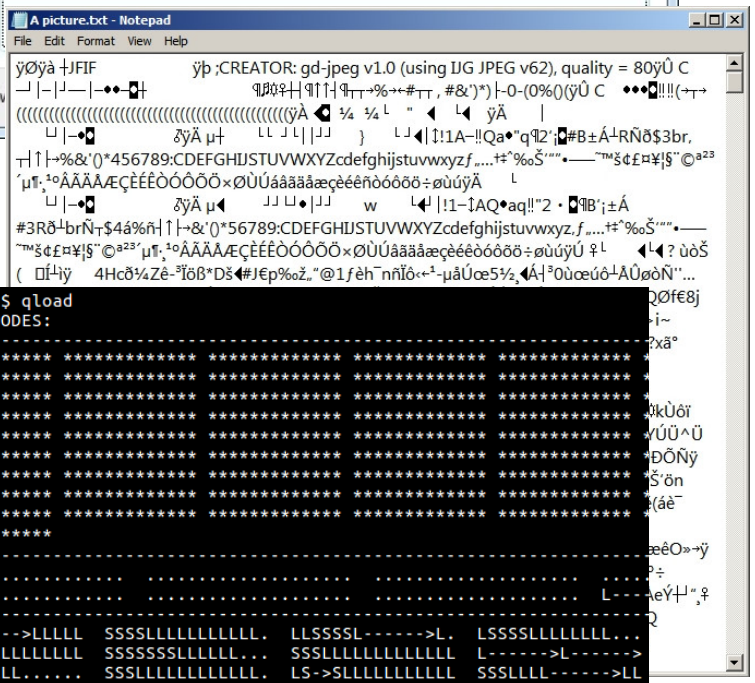
Why would you do that???

Here's some reasons:

- Genes that look like dates are automatically converted to dates by Excel
- Word or Notepad would crash if you try opening a big file (i.e. several MB or GB), like a FASTA or FASTQ
- Binary files (e.g. BAM or CRAM) can only be opened with dedicated command-line software like Samtools
- High-performance computing: the cluster doesn't have a GUI!
- Most bioinformatics tools are made for command-line users
- Command line has lots of powerful commands for parsing (even very large) text files







Moving between folders and checking their content: pwd, ls, cd (1)

Let's start by opening a terminal: where am I? Use the ***pwd*** (=print working directory) command:

```
silvia@zenbook:~$ pwd
/home/silvia
```

(If you run a command, this is where it will be executed)

What's inside the working directory I'm currently in? Use the ***ls*** command (***ls ** ** to check the content of all subfolders):

```
silvia@zenbook:~$ ls
config.txt  Documents  Music     Public  Templates
Desktop     Downloads  Pictures  R       Videos
```

The command ***ls -l*** can be used to check the difference between files and folders:

```
silvia@zenbook:~$ ls -l
total 40
-rw-rw-r-- 1 silvia silvia   12 Oct 12 11:04 config.txt
drwxr-xr-x 5 silvia silvia 4096 Oct 12 11:03 Desktop
drwxr-xr-x 2 silvia silvia 4096 Oct 12 11:01 Documents
drwxr-xr-x 3 silvia silvia 4096 Oct  4 14:53 Downloads
drwxr-xr-x 2 silvia silvia 4096 Jun 30  2016 Music
drwxr-xr-x 2 silvia silvia 4096 Oct  8 07:52 Pictures
drwxr-xr-x 2 silvia silvia 4096 Jun 30  2016 Public
drwxrwxr-x 3 silvia silvia 4096 Dec  3  2017 R
drwxr-xr-x 2 silvia silvia 4096 Jun 30  2016 Templates
drwxr-xr-x 2 silvia silvia 4096 May 27 11:09 Videos
silvia@zenbook:~$
```

(the "d" at the beginnning of the left-most column tells you that's a direcory, whereas files don't have that flag)

Moving between folders and checking their content: pwd, ls, cd (2)

The ***ls*** command can be used to check the content of other folders without changing your current directory:

```
silvia@zenbook:~$ ls Documents/
file1.txt  file2.txt  file3.txt
```

How can I change directory (for example “Desktop”)? Use the ***cd*** command:

```
silvia@zenbook:~$ cd Desktop/
silvia@zenbook:~/Desktop$ pwd
/home/silvia/Desktop
```

And if I want to return to my home folder? Use the ***cd ~*** command (or ***cd ..*** if it’s the parent directory):

```
silvia@zenbook:~/Desktop$ cd ~
silvia@zenbook:~$ pwd
/home/silvia
```

```
silvia@zenbook:~/Desktop$ cd ..
silvia@zenbook:~$ pwd
/home/silvia
```

Messy screen? Use the ***clear*** command to clear the screen:

```
silvia@zenbook:~$ clear
```

(the commands you’ve done so far are not gone, you’ll find them by just scrolling up, but the terminal is nicely cleaner now )

Creating and copying files and folders: mkdir, touch, cp

How to create a new folder? Use the ***mkdir*** command
How to create a new file? In several ways; one of them is the ***touch*** command

```
silvia@zenbook:~$ mkdir my_folder
silvia@zenbook:~$ mkdir my_folder/my_subfolder
silvia@zenbook:~$ touch my_folder/my_file.txt
silvia@zenbook:~$ ls -l my_folder/
total 4
-rw-rw-r-- 1 silvia silvia    0 Oct 12 14:01 my_file.txt
drwxrwxr-x 2 silvia silvia 4096 Oct 12 14:00 my_subfolder
silvia@zenbook:~$
```

What if I want to make a copy of a file? Use the ***cp*** command:

```
silvia@zenbook:~$ cp my_folder/my_file.txt my_folder/my_file_2.txt
silvia@zenbook:~$ ls -l my_folder/
total 4
-rw-rw-r-- 1 silvia silvia    0 Oct 12 14:09 my_file_2.txt
-rw-rw-r-- 1 silvia silvia    0 Oct 12 14:01 my_file.txt
drwxrwxr-x 2 silvia silvia 4096 Oct 12 14:00 my_subfolder
```

However, if you want to copy a folder, using cp alone will return an error. You need to add the ***-r*** option, which will copy the content of that folder recursively (if unsure, have a look at ***man cp***):

```
silvia@zenbook:~$ cp my_folder/ my_folder_2/
cp: omitting directory 'my_folder/'
silvia@zenbook:~$ cp -r my_folder/ my_folder_2/
silvia@zenbook:~$
```

Some important facts about file names...

It is worth remembering that:

- File names in Linux are ***case sensitive***. E.g., the names *“File.txt”* and *“file.txt”* refer to different files

- File names beginning with a period (*“.”*) character are ***hidden***, so if you type *“ls”* you won’t see them unless you also use the option *“-a”*. Some applications usually place their configuration/settings files in your home directory as hidden files.

- In Linux there is ***no concept of a "file extension"*** as in Windows, for instance. This means you can name files as you like (e.g. *“mickeymouse”*). However, some programs might require input files to have specific extensions. Also, pay attention not to name a file like a command! (e.g. *“man”*)

- Although Linux supports file names containing white spaces and punctuation characters, please limit the characters you use to period (*“.”*), dash (“-”), and underscore (“_”) and try to ***avoid using spaces***. You will thank yourself later for this!

- Please avoid using bash commands as filenames: although possible, this might mess up your commands!

Moving and removing files or folders: mv, rm

The ***mv*** command can be used both to move and to rename files or folders:

```
silvia@zenbook:~/Desktop$ ls
test_file.txt  test_folder
silvia@zenbook:~/Desktop$ mv test_file.txt my_file.txt
silvia@zenbook:~/Desktop$ ls
my_file.txt  test_folder
silvia@zenbook:~/Desktop$ mv test_folder/ my_folder/
silvia@zenbook:~/Desktop$ ls
my_file.txt  my_folder
silvia@zenbook:~/Desktop$ mv my_file.txt my_folder/
silvia@zenbook:~/Desktop$ ls
my_folder
silvia@zenbook:~/Desktop$ ls my_folder/
my_file.txt
```

To remove files, instead, you should use the ***rm*** command. If you need to remove a folder and, therefore, its content, you should add the -r option (=”recursive”).

```
silvia@zenbook:~/Desktop$ rm my_folder/my_file.txt
silvia@zenbook:~/Desktop$ ls my_folder/
silvia@zenbook:~/Desktop$ rm -r my_folder/
silvia@zenbook:~/Desktop$ ls
silvia@zenbook:~/Desktop$
```

IMPORTANT: when you delete something with “rm” they’re gone (it doesn’t ask for confirmation)!

TRICK: try first the same command but with “ls” instead of “rm”.

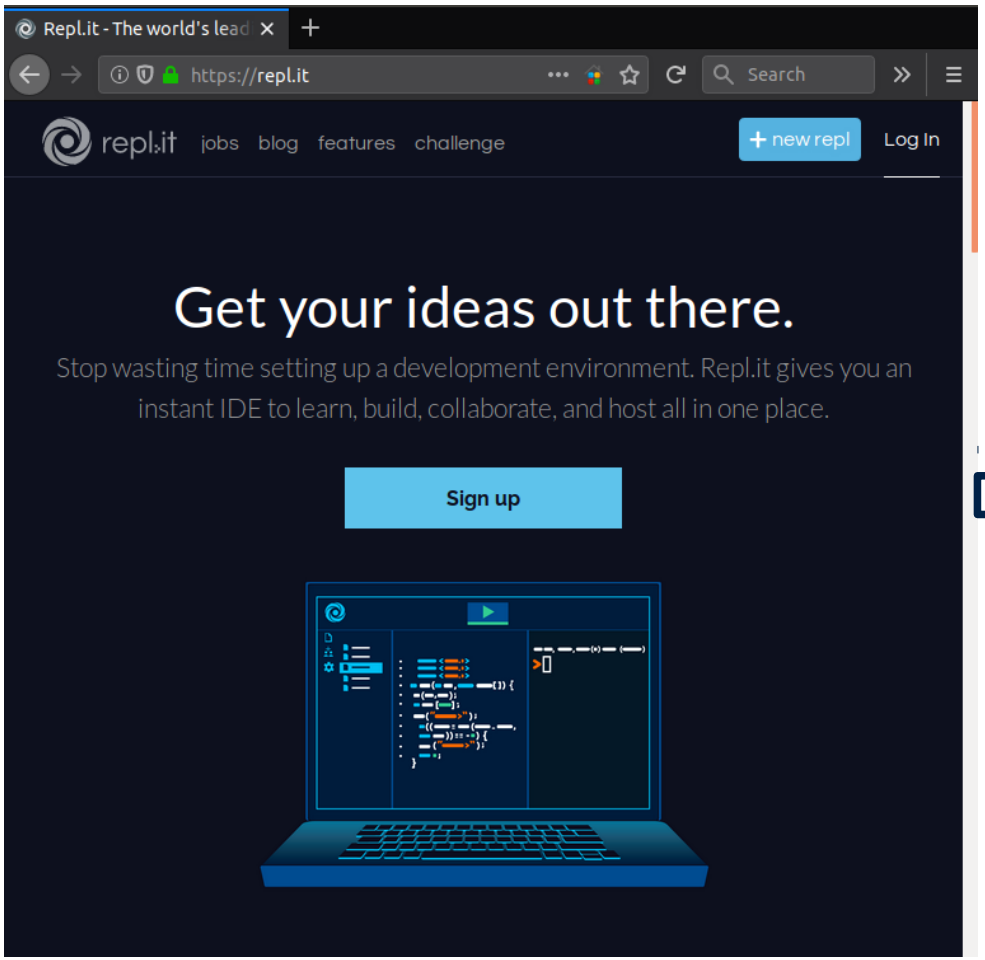
To remove multiple files, for example all those ending with “.png”, you can use the wildcard “*****” (e.g. “rm my_folder/*.png”). However, be careful with that: for example, the command “rm *” will delete everything in your current folder!

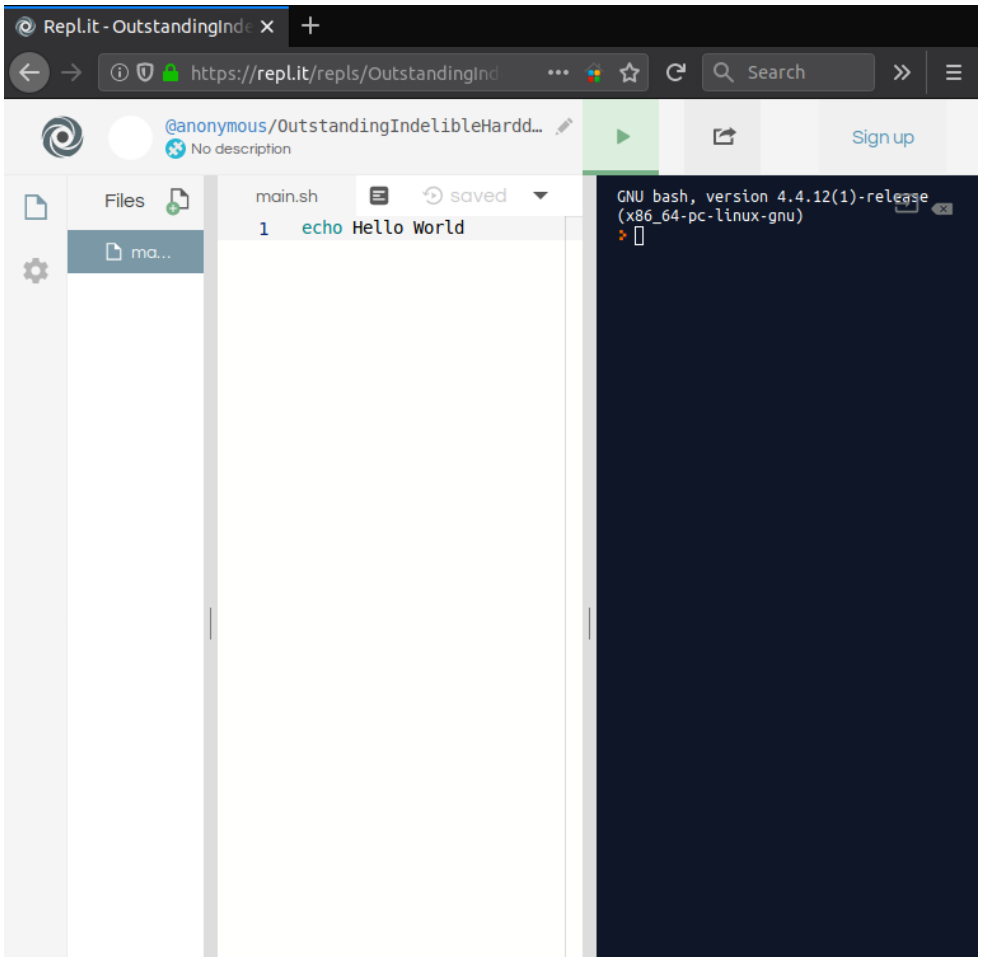
Practical session 

Setup

Open a browser and go to this URL **https://repl.it** Then, click on "**+ new repl**" and select "**bash**" from the drop-down menu. Et voilà!

You now have a terminal on the right-hand side, a text editor in the central panel, and a file navigation panel on the left-hand side. Let's start playing with it!





Practical session 1 – exercises

1) Create a new directory named **`folder_A/`**

2) Move into the folder you just created and create a new file named **`file_A.txt`**

3) Go back to your home folder, make a copy of **`folder_A/`** and call it **`folder_B/`**

4) Rename the text file in **`folder_B/`** as **`file_B.txt`**

5) Check the content of both folders

Practical session 1 – solutions

1) **`mkdir folder_A/`**

2) **`cd folder_A/`** , followed by **`touch file_A.txt`**

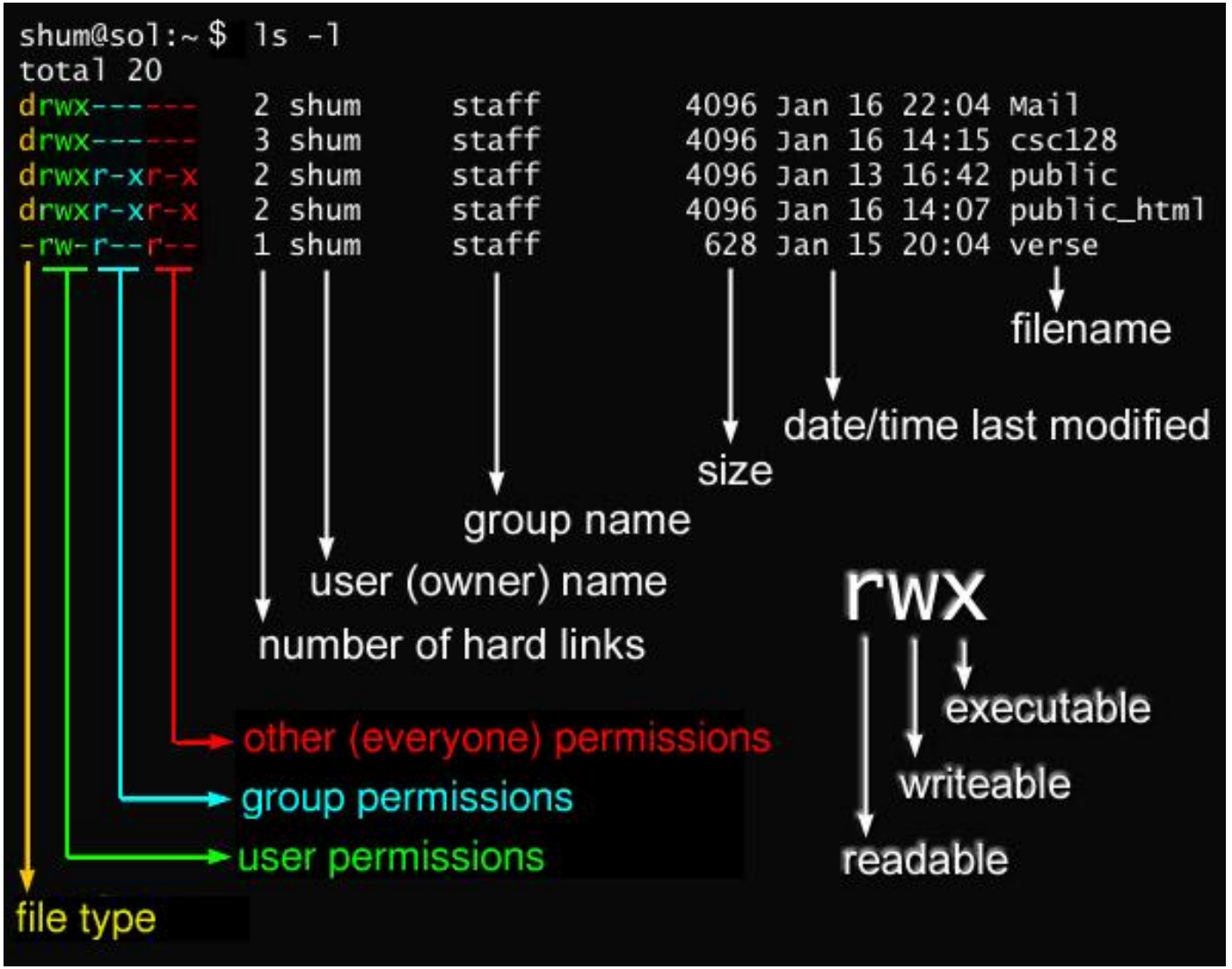
3) **`cd ~`** , followed by **`cp -r folder_A/ folder_B/`**

4) **`mv folder_B/file_A.txt folder_B/file_A.txt`**

5) **`ls folder_*`**

Changing permissions: chmod, chown, chgrp (1)

As we saw earlier, the ls -l command gives you a lot of information, including size, date/time, number of hard links, owner, group name and ***file permissions***:



Changing permissions: chmod, chown, chgrp (2)

Sometimes it might be necessary to change the default file permissions (e.g. when sharing files with other users, or making a file executable, protecting files against malicious tampering, etc.).

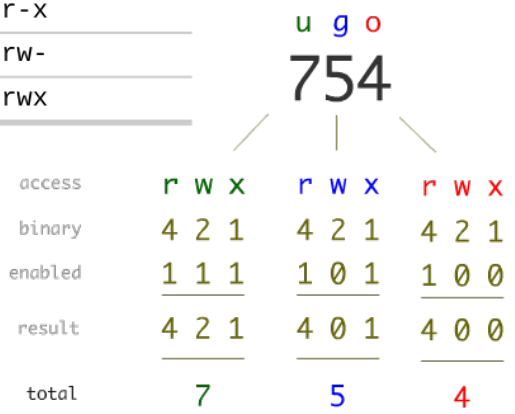
All files and directories are "owned" by the person who created them.

Only the ***owner*** and ***root*** (super user) are allowed to the change the permission of a file or directory, that is, they can set the read (r), write (w) and execute (x) permissions.

The ***chmod*** command is the key to do this. It can be used in two different ways:

octal mode

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx



symbolic mode

u	User
g	Group
o	Other
a	All three

+	Add
-	Remove
=	Equals

r	read
w	write
x	execute or search
s	setuid/setgid
t	sticky

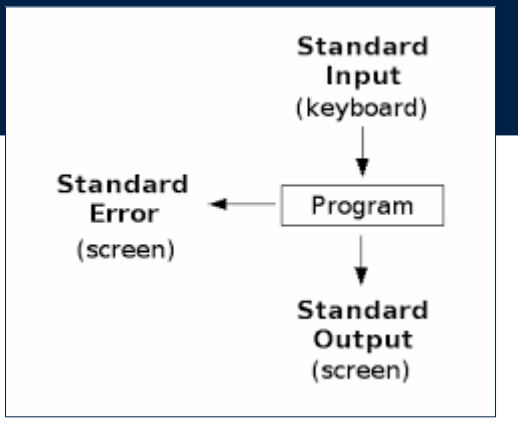
Changing the ownership (user/group) of files and directories with the commands ***chown*** / ***chgrp*** is only allowed to root.

Viewing the content of a file: cat, less, more, file, head, tail, wc

There are several ways to see the content of a file without having to use a text editor:

- ***cat*** will print the whole file content on the screen
- ***more*** *shows you the content of the file one page at a time. Press the space bar to see the next page*
- ***less*** will let you scroll through your file using the arrows and the PageUp / PageDown buttons on your keyboard; to exit from this view mode, just type *“q”*
- ***file*** can recognise most types of files without the need to open them, such as ASCII-text based files, bash files, executable binaries, web pages, compressed archives, etc.
- ***head*** will show you the first 10 lines of a file (default, it can be customised)
- ***tail*** will show you the first 10 lines of a file (default, it can be customised)
- ***wc*** will tell you the number of lines, words and bites of a given file

I/O redirection and piping (1)



In computer programming, there are three standardised streams of communication: standard input (***stdin***), standard output (***stdout***), and standard error (***stderr***).

Standard Output:

Many commands (e.g. “ls”) write their output on the display. However, sometimes you might need instead to have this output written to a file, or device instead. There are some special notations to do this:

- the “**>**” symbol ***writes*** the output of a command to a new file, so no results are shown on the screen. If the file doesn’t exist, it will be created; otherwise, it will be overwritten!

```
silvia@zenbook:~/Documents$ ls
a_file.png  another_file.png
silvia@zenbook:~/Documents$ ls > ../Desktop/file_list.txt
silvia@zenbook:~/Documents$ cat ../Desktop/file_list.txt
a_file.png
another_file.png
```

- the “**>>**” symbol ***appends*** the output of a command to a new file; if the file does not exist, it will be created; otherwise, the output will be added to the end of the file (the ***echo*** command used here displays a line of text to the standard output):

```
silvia@zenbook:~/Documents$ echo "something" >> ../Desktop/file_list.txt
silvia@zenbook:~/Documents$ cat ../Desktop/file_list.txt
a_file.png
another_file.png
something
```

I/O redirection and piping (2)

Standard Input:

Several commands can accept their input from a file or another command:

```
silvia@zenbook:~/Documents$ cat file_list.txt
file2.png
file1.png
file3.png
silvia@zenbook:~/Documents$ sort < file_list.txt
file1.png
file2.png
file3.png
silvia@zenbook:~/Documents$ cat file_list.txt
file2.png
file1.png
file3.png
```

A command can have both its input and output redirected:

```
silvia@zenbook:~/Documents$ sort < file_list.txt > sorted_file_list.txt
silvia@zenbook:~/Documents$ cat sorted_file_list.txt
file1.png
file2.png
file3.png
```

Pipelines:

Probably the most useful option for I/O redirection. It allows you to connect multiple commands by feeding the standard output of one command into the standard input of another command. Here's an example:

```
silvia@zenbook:~/Documents$ ls -l | less
```

Some filters: sort, uniq, cut, grep, tr, sed

Certain commands – often combined in pipelines – are used to take standard input, perform some operation on it, and then send the result to the standard output:

- ***sort*** sorts (numerically, alphabetically, or randomly) the standard input and outputs the sorted result to the standard output; we already saw an example in the previous slide
- ***uniq*** removes duplicate lines from the standard input (remember to sort it first!)
- ***cut*** lets you slice up lines based on particular criteria
- ***grep*** extracts the specified pattern of characters from the standard input

```
silvia@zenbook:~/Desktop$ grep AAA my_sequences.txt
TCGAAAG
AAGTCGAAACT
```

- ***tr*** translates characters into others (e.g. uppercase in lowercase)

```
silvia@zenbook:~/Desktop$ cat my_sequences.txt | tr [:upper:] [:lower:]
tcgaaag
aagtcgaaact
```

- ***sed*** can parse and transform text in a more sophisticated way than “tr”; it also works in-place

```
silvia@zenbook:~/Desktop$ sed 's/[AT]/N/gi' my_sequences.txt
NCGNNNG
NNGNCGNNNCN
```

Practical session 

Practical session 2 – exercises

1) Create a new empty file named **`test.txt`** , check its default file permissions, and then change them such that every user can read and modify the file

2) Using **`echo`** and redirection, write the following 5 strings, one per line (NB: use “\n” to write a new line) in the file **`test.txt`** :

```
5 oranges
3 bananas
2 apples
4 pears
1 pineapple
```

3) Display the content of your file on the stdout

4) Sort the file **`test.txt`** alphabetically (using the second column of fruit names) and, instead of having the output printed on the stdout, write it in a new file **`test_sorted.txt`**

5) Using **`tail`** to get the last 3 lines of the file **`test.txt`** , append them to the file **`test_sorted.txt`** , and then check how many lines are now in the file **`test_sorted.txt`** (there should be exactly 8 lines)

6) Sort in reverse numerical order the first column of the file **`test_sorted.txt`** , then pipe its result into a **`uniq`** command to get only unique lines, and finally use **`grep`** to extract only lines containing the word **`apple`**

Practical session 2 – solutions

1) **`touch test.txt`** , followed by **`ls -lh test.txt`** , followed by **`chmod a+rw test.txt`**

2) **`echo -e "5 oranges\n3 bananas\n2 apples\n4 pear\n1 pineapple" > test.txt`**

3) **`cat test.txt`** or **`more test.txt`**

4) **`sort -k 2 test.txt > test_sorted.txt`**

5) **`tail -3 test.txt >> test_sorted.txt`** , followed by **`wc -l test_sorted.txt`**

6) **`sort -k 1 -n -r test_sorted.txt | uniq | grep apple`**

Thank you for your attention!

Questions?

silvia@well.ox.ac.uk

bioinformatics@well.ox.ac.uk