UNIVERSITY OF OXFORD

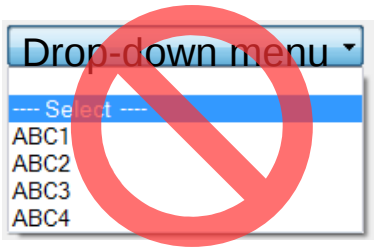
Command-line crash course

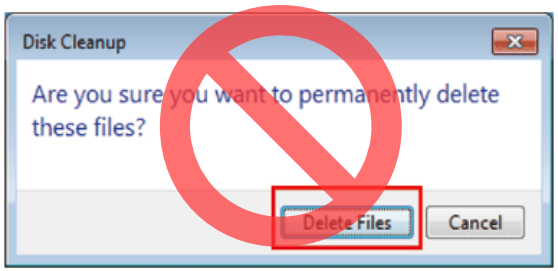
Silvia Salatino, PhD

29.11.2018 – 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!)

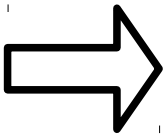
Most of them follow this simple ***general synthax***:

```
$ 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`



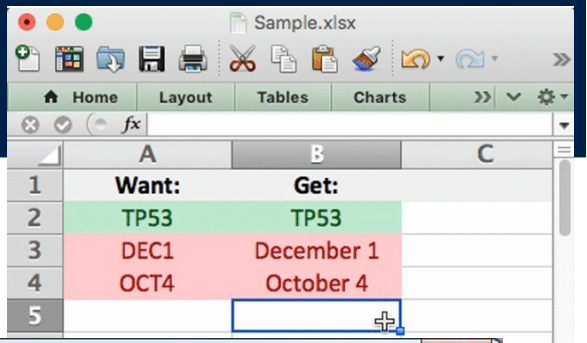
(use the arrows on your keyboard to scroll up and down the manual; then press ***q*** to exit when you're done)

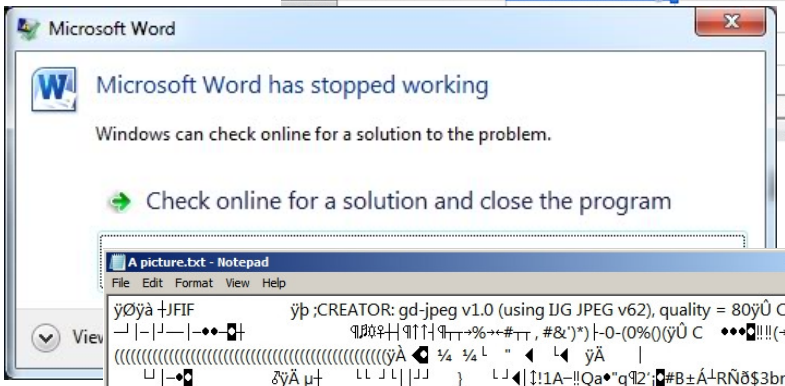
```
LS(1)                    User Commands                    LS(1)

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)
```

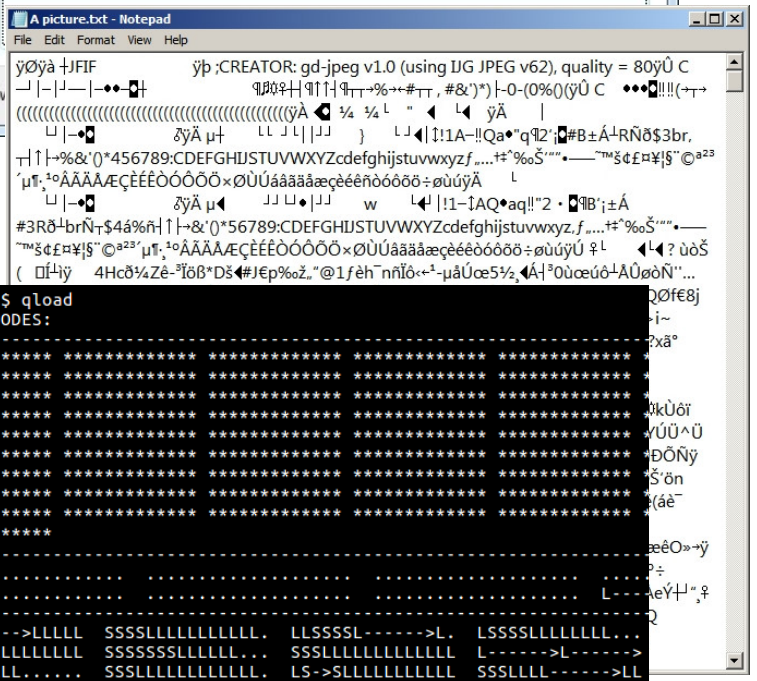
Why would you do that???

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 

How to setup your connection to the rescomp servers:

During this course, you'll be assigned a user name (e.g. workshop01) and a corresponding password.

Instructions for Windows users:

OPTION A)
1. Download SmarTTY from http://smartty.sysprogs.com/
2. Double-click on the downloaded ".msi" file and follow the steps in the setup wizard to install it
3. Launch the program and click on "New SSH connection..."
4. In the pop-up window, type:
  - Host Name: rescomp.well.ox.ac.uk
  - User Name: (your user name)
  - Password: (your password)
5. Click "Connect" and then "Start with a regular Terminal"

OPTION B)
1. Launch Xming, check only "Private networks" and click "Allow access"
2. Launch SSH Secure Shell
3. In the Secure Shell Client click on "Edit/Settings"
4. In the "Settings" window on the left-hand side click on the + next to expand "Profile Settings"
5. Click on "Tunneling"
6. On the right-hand side of the window, make sure there is a check next to the words "Tunnel X11 Connections"
7. On the left-hand side of the window, click on "Authentication". Make sure there is a check next to the words "Enable for SSH2 Connections"
8. Click "OK"; then click "File" and finally "Save Settings"; press "Enter"
10. In the pop-up window, type:
  - Host Name: rescomp.well.ox.ac.uk
  - User Name: (your user name)
  - Port Number: 22
  - Authentication Method: \<Profile Settings\>
11. Click "Connect" and, when prompted, type your password

Instructions for GNU/Linux or Mac users

1. Open an X11 terminal (XQuartz from Mac) and type `ssh -X <your_user_name>@rescomp.well.ox.ac.uk`
2. When prompted, type your password

Practical session 1 – exercises

1) Go to the “workshop folder” (that is, **`/well/workshop/workshopXX/`** where XX is your account number, for example 01) and 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 “workshop 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) **`cd ~`** , followed by **`mkdir folder_A/`**

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

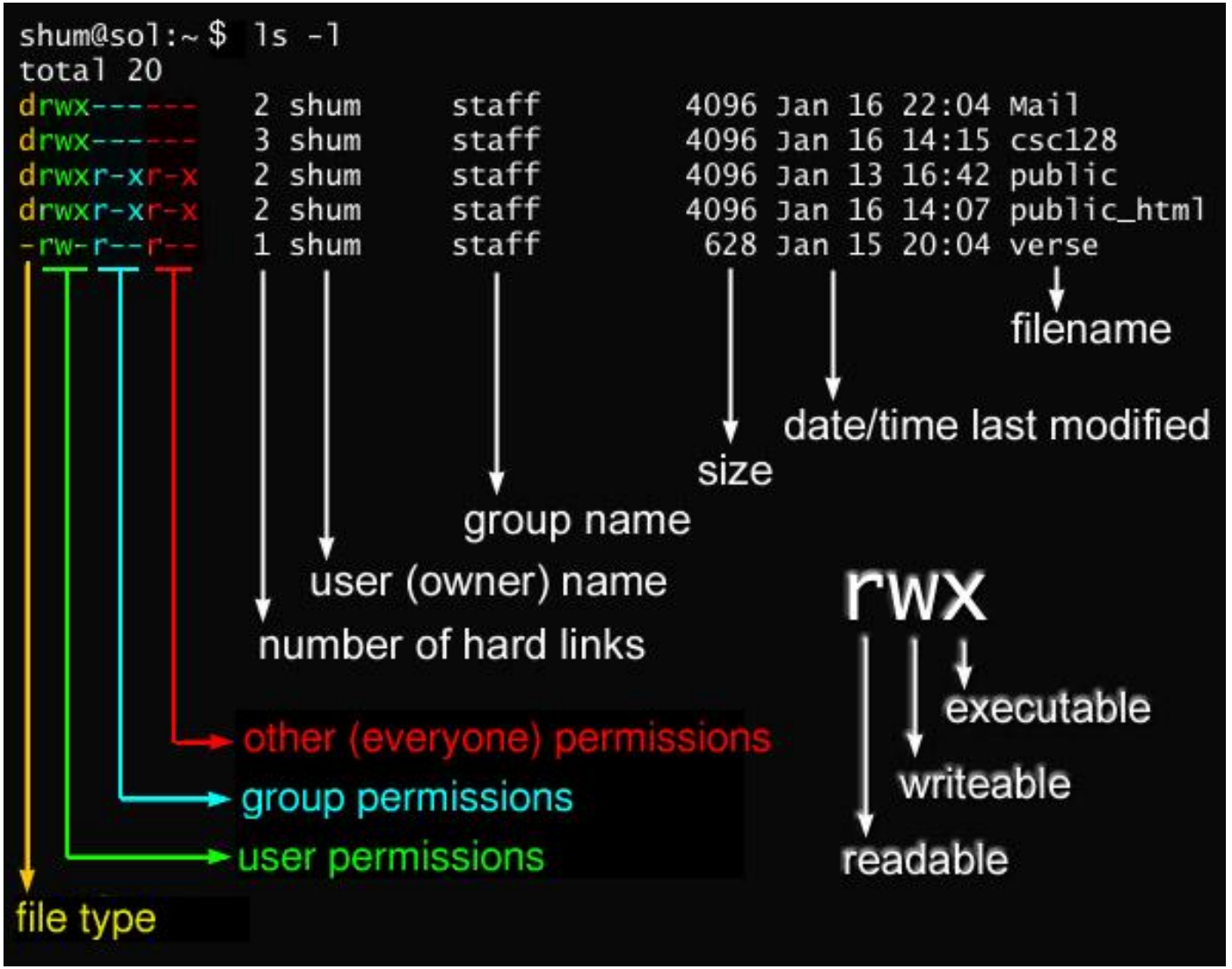
3) **`cd ~`** , followed by **`cp -r folder_A/ folder_B/`**

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

5) from your “workshop folder”: **`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.).

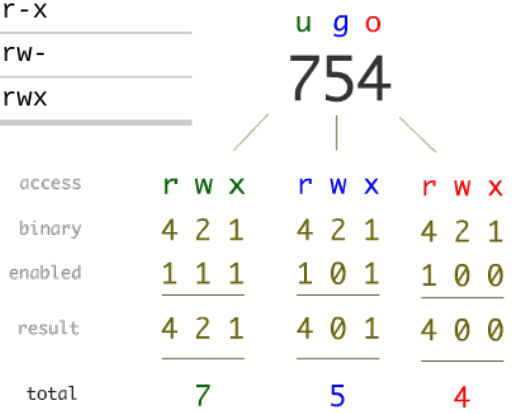
All files and directories are "owned" by the person who created them.

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

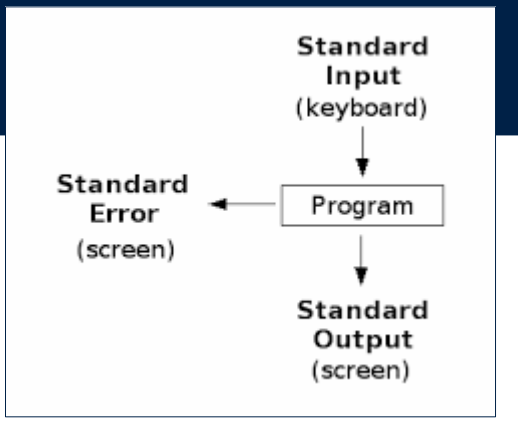
Changing the ownership (user/group) of files and directories with the commands ***chown*** / ***chgrp*** is only allowed to root.

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
```

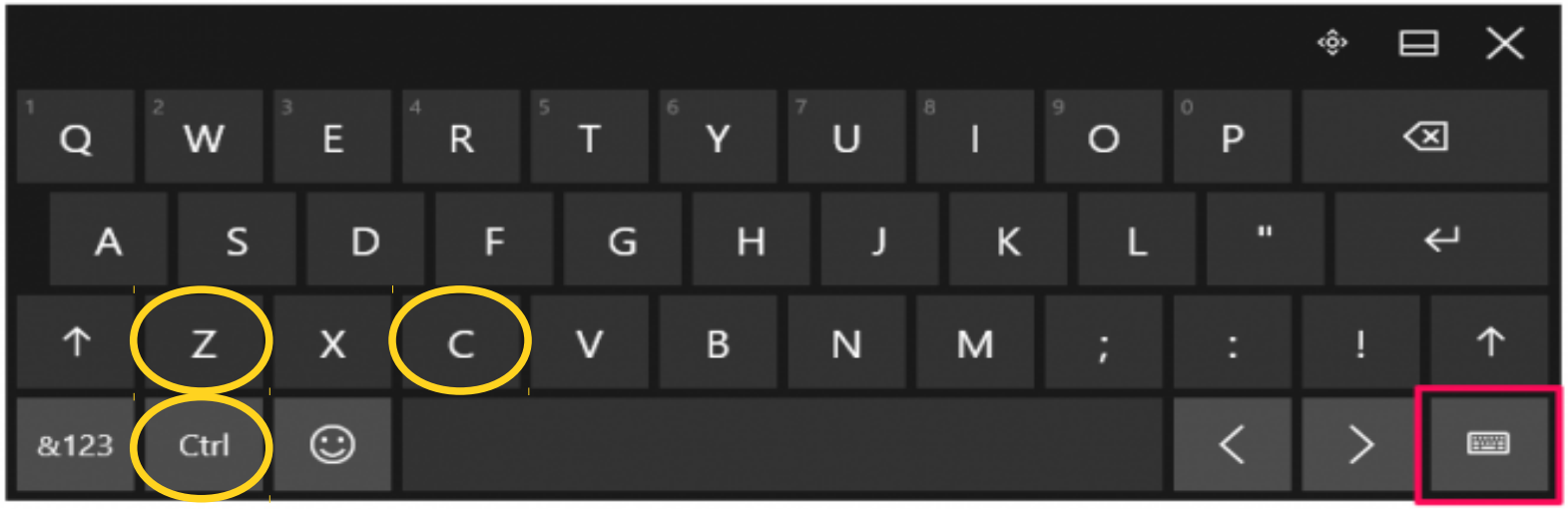
- ***tr*** translates characters into others (e.g. uppercase in lowercase)

```
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
```

A couple of very useful shortcuts to keep in mind



Control + C

- This key combination will ***interrupt a process***, usually causing it to ***abort***, but it is up to the application to decide.

Control + Z

- This key combination will ***send a foreground process to the background***, in a suspended state (= still alive but not running).

  To view the suspended processed, type "***jobs***":

```
[1]-  Stopped                 cat
[2]+  Stopped                 vi
```

  To go back to the application, type "***fg***" and it be resumed ("***bg***" does the opposite).

  To list the processes running, type "***ps***".

  To kill a suspended process in the background, type "***kill %n***", where n is the number displayed by the jobs command (e.g. kill %1 to terminate the cat process)

Looking for files: locate, find, quoting

Let's say you forgot where a file or a given program is and you need to look for it. What can you do?

- ***locate*** has one advantage over find: speed; it allows you to quickly find a particular file by name
- ***find*** has many advantages over locate: a rich expression synthax, allowing to select files not only by name, but also by date, size, owner, permissions, depth, etc.; it can search a subset of the filesystem; do actions on found files (e.g. -delete, -exec); runs in real time, so the output is always up-to-date (locate relies on a pre-built database)

When looking for files, keep in mind that if you place some text inside ***double quotes***, any special character used by the shell will be treated as an ordinary character, with the only exception of "**$**", "****", "**`**". This is useful if your file name contains white spaces, for example:

```
silvia@zenbook:~/Desktop$ ls -l my file.txt
ls: cannot access 'my': No such file or directory
ls: cannot access 'file.txt': No such file or directory
silvia@zenbook:~/Desktop$ ls -l "my file.txt"
-rw-rw-r-- 1 silvia silvia 0 Nov 12 11:20 my file.txt
```

Instead, ***single quotes*** suppress all expansions, while ***back quotes*** execute the content of variables:

```
silvia@zenbook:~/Desktop$ foo=who
silvia@zenbook:~/Desktop$ echo "$foo"
who
silvia@zenbook:~/Desktop$ echo `$foo`
silvia tty7 2018-11-12 11:20 (:0)
silvia@zenbook:~/Desktop$ echo '$foo'
$foo
```

File compression: gzip, bgzip, gunzip

In Bioinformatics, it is very common to use compressed file formats as files tend to be very big, particularly the raw data ones (like FASTA or FASTQ). Here's a couple useful commands to handle them:

- To compress and uncompress in .gz format use ***gzip*** and ***gunzip***, respectively. NOTE: these commands do overwrite the input file!

```
silvia@zenbook:~/Documents$ ls
sequences.fa
silvia@zenbook:~/Documents$ gzip sequences.fa
silvia@zenbook:~/Documents$ ls
sequences.fa.gz
silvia@zenbook:~/Documents$ gunzip sequences.fa.gz
silvia@zenbook:~/Documents$ ls
sequences.fa
```

- To compress and uncompress in .zip format use ***zip*** and ***unzip***, respectively. NOTE: these commands do not overwrite the input file

```
silvia@zenbook:~/Documents$ zip sequences.fa.zip sequences.fa
  adding: sequences.fa (stored 0%)
silvia@zenbook:~/Documents$ ls
sequences.fa  sequences.fa.zip
```

- To read and parse a compressed file w/o uncompressing it, there are the "Z commands": ***zcat***, ***zless***, ***zgrep***, ***zdiff***, ***zmore***, which behave exactly the same as their corresponding non-Z ones. Some of them uncompress the input file temporarily in the /tmp directory, others uncompress it on the fly. In either cases, they allow you to do operations on compressed files without having to worry about the overhead of uncompressing the file before performing a given operation.

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) 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`**

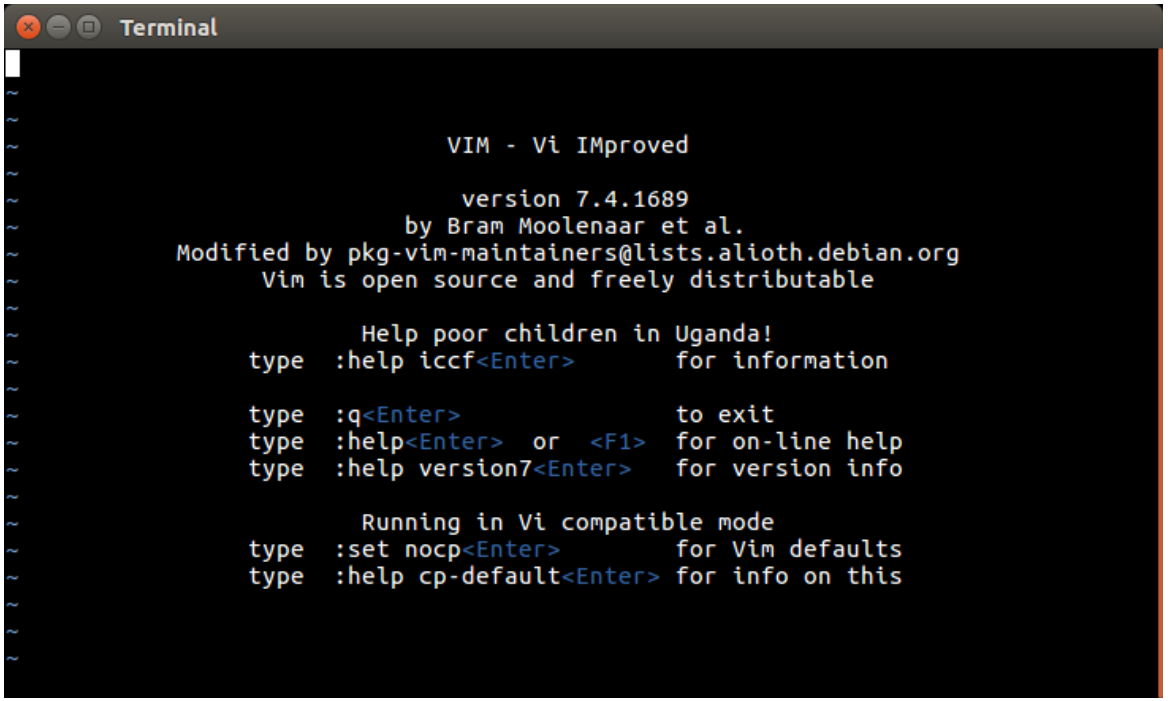
Shell scripts and text editors (1)

A shell script is a file containing a series of bash commands, executed in the command line in the order they've been written in the script.

To write a script, you need a ***text editor***. Some of them work from command-line (e.g. vi / vim, nano, emacs), whereas others have a graphical interface (e.g. gedit, sublime, atom). Most of them need to be installed (especially the newest ones). However, the text editor you'll always find already installed on Linux and Max is ***vi*** / ***vim*** (= vi improved); despite being infamous for its difficulty, it is lightweight and fast.

Let's go quickly through the basic commands:

- Enter "vi" or "vim" in the terminal
- To write some text, move the cursor to the position you want with the arrows, press "**i**" and start typing
- To exit press "**ESC**" to enter normal mode, then "**:**" and enter "**q!**"
- For help type ":help", to save type "**:w**", to save and quit type "**:wq**"
- Google "vi cheat sheet" for the full list of features/options (e.g. type "**:set number**" to display line #)



Now that you've done this rite of passage, feel free to use a more user-friendly editor! 

Shell scripts and text editors (2)

Whichever text editor you chose, it's time to start writing a short shell script!

Open a new file in your favourite editor, save it as "hello_world.sh" and start typing these lines:

```
#!/bin/bash
# My first script

echo "Hello World!"
```

The first line is very important. It tells the shell which program is used to interpret the script (in this case it's bash, but it might well be python, awk, etc.). The second line is just a comment; everything written after the "#" is ignored by bash. Comments are crucial to document your code, particularly if you have to share it with others.

Next, change permissions to make your script executable and run it:

```
silvia@zenbook:~/Documents$ ls -l
total 4
-rw-rw-r-- 1 silvia silvia 53 Nov 13 10:44 hello_world.sh
silvia@zenbook:~/Documents$ chmod u+x hello_world.sh
silvia@zenbook:~/Documents$ ls -l
total 4
-rwxrw-r-- 1 silvia silvia 53 Nov 13 10:44 hello_world.sh
silvia@zenbook:~/Documents$ ./hello_world.sh
Hello World!
```

Shell scripts and text editors (3)

Sometimes you might need to repeat the same operation multiple times. To do this, you can use a ***for loop***, which will apply the same operation to a given variable representing each element of an array, set, or list.

Here are a few examples:

script_1.sh (~/Desktop) - gedit

```
1 #! /bin/bash
2 # This script prints integer numbers from 1 to 3.
3 for i in 1 2 3
4 do
5   echo $i
6 done
7
```

```
silvia@zenbook:~/Desktop$ ./script_1.sh
1
2
3
```

script_2.sh (~/Desktop) - gedit

```
1 #! /bin/bash
2 # This script prints odd numbers between 1 and 10.
3 for (( i=1; i<=10; i+=2 ))
4 do
5   echo $i
6 done
7
```

```
silvia@zenbook:~/Desktop$ ./script_2.sh
1
3
5
7
9
```

script_3.sh (~/Desktop) - gedit

```
1 #! /bin/bash
2 # This script prints each element of the array "fruit"
3 fruit=("apple" "banana" "orange")
4 for i in ${fruit[@]}
5 do
6   echo $i
7 done
8
```

```
silvia@zenbook:~/Desktop$ ./script_3.sh
apple
banana
orange
```

Shell scripts and text editors (4)

...and here are some more examples, ***using as input the output of another command*** (ls in this case):

script_4.sh (~/Desktop) - gedit

```
#! /bin/bash
# This script prints all files in the current directory
for i in $(ls)
do
  echo $i
done
```

```
silvia@zenbook:~/Desktop$ ./script_4.sh
script_1.sh
script_2.sh
script_3.sh
script_4.sh
```

...***or user-defined arguments provided from command line*** (the first argument is assigned to the variable $1, the second to $2, etc.).

script_5.sh (~/Desktop) - gedit

```
#! /bin/bash
# This script prints first and third input arguments
echo $1
echo $3
```

```
silvia@zenbook:~/Desktop$ ./script_5.sh apple banana orange
apple
orange
```

If the number of input arguments can vary from run to run, it's better to use ***$***@ to catch them all (the variable ***$#*** is assigned the total number of input arguments, instead):

script_6.sh (~/Desktop) - gedit

```
#! /bin/bash
# This script prints all the input arguments
for i in $@
do
  echo $i
done
```

```
silvia@zenbook:~/Desktop$ ./script_6.sh apple banana orange
apple
banana
orange
```

Shell scripts and text editors (5)

Often you need to do an operation only when a certain condition is verified, or to add a control that checks if the number of input arguments is correct. This can be done using the ***if - elif - else*** statements:

guess_my_age.sh (~/Desktop) - gedit

```
1 #! /bin/bash
2 # This script shows the usage of conditional statements
3 my_num=$1
4 if [ $my_num -ge 33 ]
5 then
6   echo "I am not that old! Grr..."
7 elif [ $my_num -le 31 ]
8 then
9   echo "Thanks, very kind of you :)"
10 else
11   echo "Yay, correct answer!"
12 fi
13
```

```
silvia@zenbook:~/Desktop$ ./guess_my_age.sh 50
I am not that old! Grr...
silvia@zenbook:~/Desktop$ ./guess_my_age.sh 27
Thanks, very kind of you :)
silvia@zenbook:~/Desktop$ ./guess_my_age.sh 32
Yay, correct answer!
```

However, you don't necessarily need to use a text editor... bash commands can also be entered directly in the terminal (provided they're really short and you don't need to re-run them over and over!):

```
silvia@zenbook:~/Desktop$ for i in 1 2 3; do echo $i; done
1
2
3
```

Practical session 

Practical session 3 – exercises

1) Open a text editor of your choice. Write a bash script that uses a for loop to show the name of each **`.txt`** file in your folder and prints the number of lines in the file. Your “workshop folder” should still contain the two files created in Practical session II.

2) Modify the script above such that it prints the filename and **`Okay`** only if the word count is greater or equal 6, otherwise it prints the filename and message **`Too few lines!`**

Practical session 3 – solutions

1)

```
#! /bin/bash
for f in `ls *.txt`
do
  echo $f
  wc -l $f
done
```

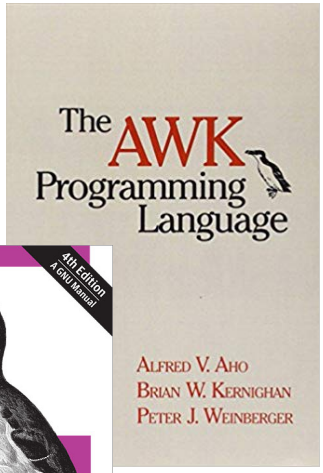
2)

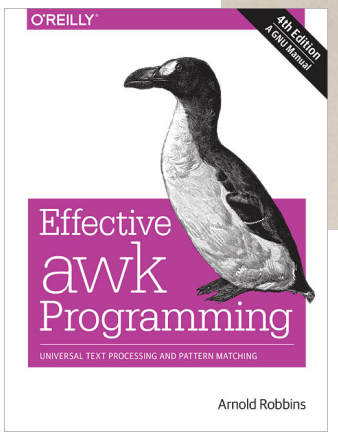
```
#! /bin/bash
for f in `ls *.txt`
do
  if [ `wc -l $f | cut -d ' ' -f 1` -ge 6 ]
  then
    echo $f
    echo "Okay"
  else
    echo $f
    echo "Too few lines"
  fi
done
```

An extremely powerful programming language: AWK

AWK is a very powerful programming language specifically designed for text processing. It is often used as a data extraction tool, but it can do many other things, including but not limited to:

- operations (e.g. sum of the numbers in a given column of a file)
- printing selected fields in a given order
- reporting matching lines and/or substituting them with a new word/character
- counting the number of non-empty lines
- deleting white spaces or adding certain characters before even/odd lines
- converting hex strings to decimal
- etc.





A very useful thing about AWK is the possibility to do quite complicated operations in the so-called ***one liners***. The following are two examples of AWK one liners; the first converting the file we saw in the previous slide into a FASTA format, and the second reporting lines longer than 10 bp (and their line #):

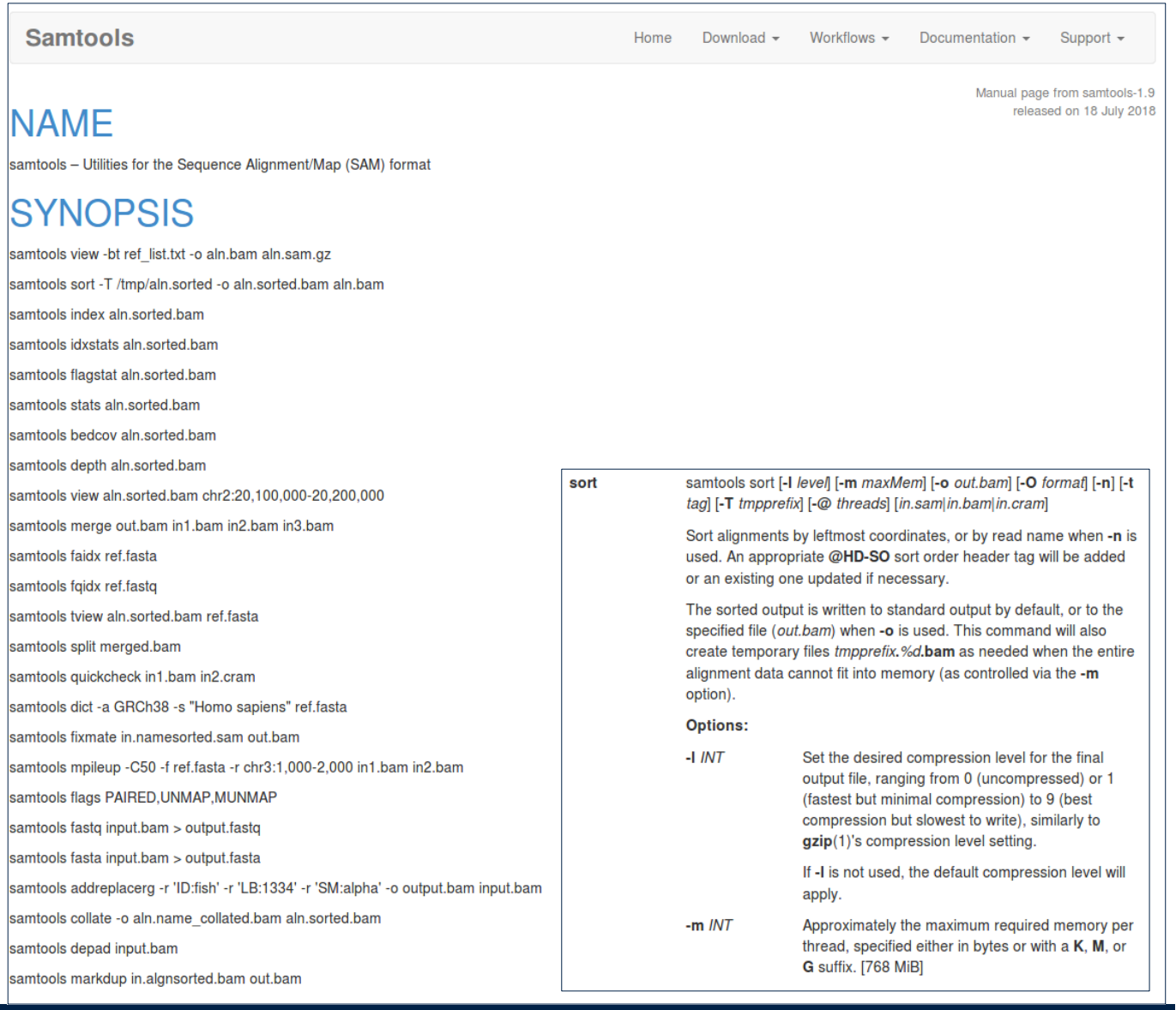
```
silvia@zenbook:~/Desktop$ awk '{n++; print ">seq_"n; print}' my_sequences.txt
>seq_1
TCGAAAG
>seq_2
AAGTCGAAACT
silvia@zenbook:~/Desktop$ awk 'length($0)>10 {print FNR, $0}' my_sequences.txt
2 AAGTCGAAACT
```

Samtools, Bamtools, Bedtools, Bcftools (1)

The standard file formats used in genomics / transcriptomics are only a few (e.g. FASTA, FASTQ, BAM, CRAM, SAM, BED, GTF/GFF, VCF), but they are quite commonly used. A number of suites have been developed to handle these files:

- ***Samtools*** is *"a set of utilities that manipulate alignments in the BAM format. It imports from and exports to the SAM format, does sorting, merging and indexing, and allows to retrieve reads in any regions swiftly.*"

  Several samtools commands can be piped after each other by using "-" to indicate standard input / output. Warnings and error messages are printed to the standard error.



Samtools

Home Download Workflows Documentation Support

Manual page from samtools-1.9
released on 18 July 2018

NAME

samtools – Utilities for the Sequence Alignment/Map (SAM) format

SYNOPSIS

```
samtools view -bt ref_list.txt -o aln.bam aln.sam.gz
samtools sort -T /tmp/aln.sorted -o aln.sorted.bam aln.bam
samtools index aln.sorted.bam
samtools idxstats aln.sorted.bam
samtools flagstat aln.sorted.bam
samtools stats aln.sorted.bam
samtools bedcov aln.sorted.bam
samtools depth aln.sorted.bam
samtools view aln.sorted.bam chr2:20,100,000-20,200,000
samtools merge out.bam in1.bam in2.bam in3.bam
samtools faidx ref.fasta
samtools fqidx ref.fastq
samtools tview aln.sorted.bam ref.fasta
samtools split merged.bam
samtools quickcheck in1.bam in2.cram
samtools dict -a GRCh38 -s "Homo sapiens" ref.fasta
samtools fixmate in.namesorted.sam out.bam
samtools mpileup -C50 -f ref.fasta -r chr3:1,000-2,000 in1.bam in2.bam
samtools flags PAIRED,UNMAP,MUNMAP
samtools fastq input.bam > output.fastq
samtools fasta input.bam > output.fasta
samtools addreplacerg -r 'ID:fish' -r 'LB:1334' -r 'SM:alpha' -o output.bam input.bam
samtools collate -o aln.name_collated.bam aln.sorted.bam
samtools depad input.bam
samtools markdup in.algnsorted.bam out.bam
```

sort samtools sort [**-l** *level*] [**-m** *maxMem*] [**-o** *out.bam*] [**-O** *format*] [**-n**] [**-t** *tag*] [**-T** *tmpprefix*] [**-@** *threads*] [*in.sam*|*in.bam*|*in.cram*]

Sort alignments by leftmost coordinates, or by read name when **-n** is used. An appropriate **@HD-SO** sort order header tag will be added or an existing one updated if necessary.

The sorted output is written to standard output by default, or to the specified file (*out.bam*) when **-o** is used. This command will also create temporary files *tmpprefix.%d*.**bam** as needed when the entire alignment data cannot fit into memory (as controlled via the **-m** option).

Options:

-l *INT* Set the desired compression level for the final output file, ranging from 0 (uncompressed) or 1 (fastest but minimal compression) to 9 (best compression but slowest to write), similarly to **gzip**(1)'s compression level setting.

If **-l** is not used, the default compression level will apply.

-m *INT* Approximately the maximum required memory per thread, specified either in bytes or with a **K**, **M**, or **G** suffix. [768 MiB]

Samtools, Bamtools, Bedtools, Bcftools (2)

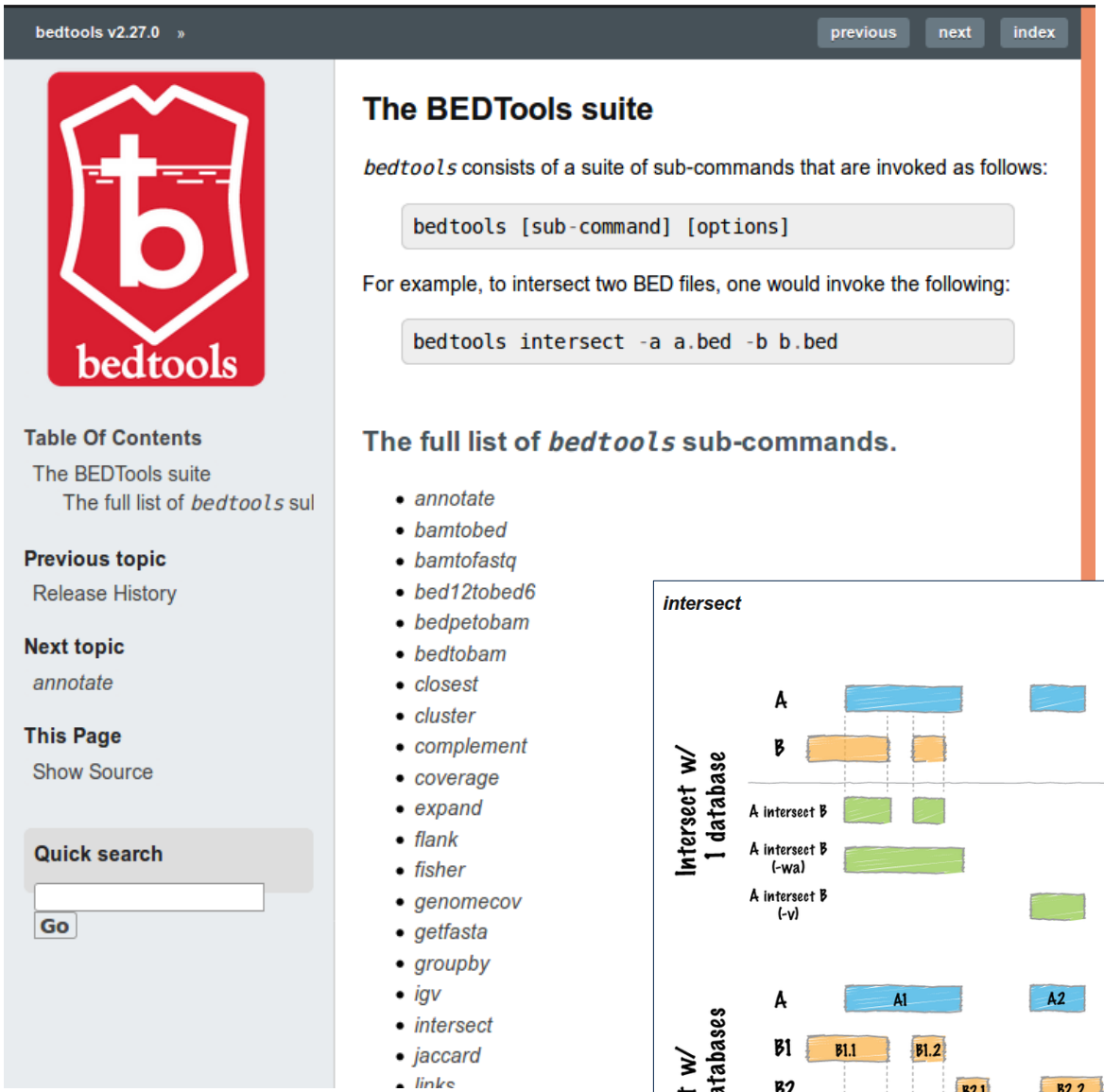
- ***Bamtools*** is a suite of utilities for handling BAM files.

  Most of the operations can also be done with Samtools, but this is more intuitive to use (e.g. no numerical flags to remember), although it is slower...

```
[silvia@rescomp1 Teaching]$ /apps/well/bamtools/2.3.0/bin/bamtools --help

usage: bamtools [--help] COMMAND [ARGS]

Available bamtools commands:
       convert         Converts between BAM and a number of other formats
       count           Prints number of alignments in BAM file(s)
       coverage        Prints coverage statistics from the input BAM file
       filter          Filters BAM file(s) by user-specified criteria
       header          Prints BAM header information
       index           Generates index for BAM file
       merge           Merge multiple BAM files into single file
       random          Select random alignments from existing BAM file(s), intended more as a
       resolve         Resolves paired-end reads (marking the IsProperPair flag as needed)
       revert          Removes duplicate marks and restores original base qualities
       sort            Sorts the BAM file according to some criteria
       split           Splits a BAM file on user-specified property, creating a new BAM output
       stats           Prints some basic statistics from input BAM file(s)

See 'bamtools help COMMAND' for more information on a specific command.

[silvia@rescomp1 Teaching]$ /apps/well/bamtools/2.3.0/bin/bamtools filter --help
Description: filters BAM file(s).

Usage: bamtools filter [-in <filename> -in <filename> ... | -list <filelist>] [-out <filename>
egion <REGION>] [ [-script <filename] | [filterOptions] ]

Input & Output:
  -in <BAM filename>               the input BAM file(s) [stdin]
  -list <filename>                 the input BAM file list, one
                                   line per file
  -out <BAM filename>              the output BAM file [stdout]
  -region <REGION>                 only read data from this
                                   genomic region (see documentation for more
                                   details)
  -script <filename>               the filter script file (see
                                   documentation for more details)
  -forceCompression                if results are sent to stdout
                                   (like when piping to another tool),
                                   default behavior is to leave output
                                   uncompressed. Use this flag to override
                                   and force compression

General Filters:
  -alignmentFlag <int>             keep reads with this *exact*
                                   alignment flag (for more detailed queries,
```

Samtools, Bamtools, Bedtools, Bcftools (3)

- ***Bedtools*** is "*a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable genome arithmetic: that is, set theory on the genome. For example, bedtools allows one to intersect, merge, count, complement, and shuffle genomic intervals from multiple files in widely-used genomic file formats such as BAM, BED, GFF/GTF, VCF. While each individual tool is designed to do a relatively simple task (e.g., intersect two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the UNIX command line.*"



Certain commands also have schemes to describe in a clear graphical way all the different options

Samtools, Bamtools, Bedtools, Bcftools (4)

- ***Bcftools*** is "*a set of utilities that manipulate variant calls in the Variant Call Format (VCF) and its binary counterpart BCF. All commands work transparently with both VCFs and BCFs, both uncompressed and BGZF-compressed.*"

  Like Samtools, "*BCFtools is designed to work on a stream. It regards an input file "-" as the standard input (stdin) and outputs to the standard output (stdout). Several commands can thus be combined with Unix pipes.*"

LIST OF COMMANDS

For a full list of available commands, run **bcftools** without arguments. For a full list of available options, run **bcftools** *COMMAND* without arguments.

- **annotate** .. edit VCF files, add or remove annotations
- **call** .. SNP/indel calling (former "view")
- **cnv** .. Copy Number Variation caller
- **concat** .. concatenate VCF/BCF files from the same set of samples
- **consensus** .. create consensus sequence by applying VCF variants
- **convert** .. convert VCF/BCF to other formats and back
- **csq** .. haplotype aware consequence caller
- **filter** .. filter VCF/BCF files using fixed thresholds
- **gtcheck** .. check sample concordance, detect sample swaps and contamination
- **index** .. index VCF/BCF
- **isec** .. intersections of VCF/BCF files
- **merge** .. merge VCF/BCF files files from non-overlapping sample sets
- **mpileup** .. multi-way pileup producing genotype likelihoods
- **norm** .. normalize indels
- **plugin** .. run user-defined plugin
- **polysomy** .. detect contaminations and whole-chromosome aberrations
- **query** .. transform VCF/BCF into user-defined formats
- **reheader** .. modify VCF/BCF header, change sample names
- **roh** .. identify runs of homo/auto-zygosity
- **sort** .. sort VCF/BCF files
- **stats** .. produce VCF/BCF stats (former vcfcheck)
- **view** .. subset, filter and convert VCF and BCF files

Thank you for your attention!

Questions?

silvia@well.ox.ac.uk

bioinformatics@well.ox.ac.uk