❗ Page under costruction

Command Line: first commands

Today we will use a first set of commands.

General rules

- ❗ Read carefully each paragraph before typing commands.
- ❗ Use the **TAB completion**. Always.
- ❗ Strictly follow the instructions using the suggest filenames: creativity is not a positive quality for Bash beginners ;)

Log in

Enter in your VM, and restore your **screen**.

If you need help, see:

- How to log into your VM
- Using screen
- A simple screencast of a login session

Listing files

The general syntax is: **ls [options] [files]**. Both the options and the files are optional, and files can be files of directories. Now we introduce *some* of the options:

Option	Description
`-a`	Also show hidden files
`-l`	Long format, will show one file per line, with size, owner, date…
`-h`	Used with `-l`, will display file size in *human readable* format (e.g. 2.3Mb instead of 2298011)
`-d`	Show directories as files, without listing their content

The options can be combined together, the following two commands are identical:

```
ls -l -h -a
ls -lha
```

If we want to list the files present at the *root*, we don't need to move there, but simply ask `ls` which path to scan for you:

```
ls /
```

Here another example:

```
ls /homes/qib/examples/
```

You can type as many paths (files or directories) as needed in a single ls command:

```
ls -l ~/.bashrc ~/.screenrc /homes/qib/examples/
```

Using the "shell expansion": wildcards to select multiple files

As we noticed, `ls` can receive more than one file. Usually, though, we don't type each single item to be listed, but rather we use *wildcards*, then the shell will *expand* our shortcuts into a list of paths. There are wildcards, ranges and lists to be used.

Symbol	**Meaning**	**Example**
*	Any set of characters (any length)	***.fasta**: all files ending with ".fasta"
?	A single character	**A???.txt**: files starting with A, followed by exactly 3 chars, endin by ".txt"
[a-z]	Range: any single lowercase letters	**file1[a-c].txt**: files called file1a, file1b and file1c, ending with ".txt"
[0-9]	Range, any single digit	**reads_R[1-2].fastq**: reads_R1.fastq and reads_R2.fastq
{a,b}	Comma separated list of words	**photo_{andrea,john}.jpg**: photo_andrea.jpg and photo_john.jpg

Toy files

There is a directory with some example files. First we will copy it to our home using the `cp` (copy) command and the `-r` (recursive) switch, since we want also the content of the directory[1]:

```
cp -r /homes/qib/examples/ ~
```

We should be in our home directory. Check with `pwd`, in case type `cd` to quickly return to your home directory.

To enter the new directory, type (**remember the TAB**):

```
cd examples
```

Now, using `cd` and `ls` try figuring out:

- How many directories are inside the examples directory[2]
- The content of each directory[3]

Creating a directory, coping some files

Create a directory called `copies` inside the examples directory. There are many ways: **if** you are already inside “examples”, just:

```
mkdir copies
```

Otherwise you have to craft the proper relative or absolute path.

Let's try again to copy some files. In particular we want a selection of files inside the *phage* directory:

```
# If we are not inside the examples directory:
cd ~/examples/
# Copy some files
cp -v phage/*.f?? copies/
```

In this case we use a new switch, `-v` (verbose) that will print all the files copied (useful when we want to see the progress). Using both * and ? wildcards we select all the files having an extension of three chars, the first being “f” (e.g. fna, faa).

Terminal demos

⚠ This feature has been removed

This server has a sort of cinema, to play recorded screencasts directly at the command line. The command is `playdemo` followed by the name of the film you want to see (type “playdemo” alone to see some titles). Example:

```
# See a small cinema on "find"
playdemo find
```

Find

The `find` command can print all the files from a starting path, including directories and subdirectories.

Some examples:

```
# Print all files and directories in my home
find ~

# Print all files and directories in a specific path
find /usr/lib/ssl

# Print only directories / files
find ~ -type d
find ~ -type f

# Print files in a home with a specific extension
find ~ -name "*.txt"
```

Viewing text files

The simples command is `cat` (concatenate), that can print the content of one or more files. Example:

```
cat ~/examples/files/wine.csv
```

Can you type it using a *relative path*?

When a file is very large, it's very convenient to have a look at a fraction of it. The commands head and `tail` allows to print only the first (or last) lines of a file. By default 10 lines, but you can change this with `-n`:

```
head ~/examples/files/wine.csv
head -n 3 ~/examples/files/wine.csv
tail -n 5 ~/examples/files/wine.csv
```

Do you remember man? Good, as we can now use a new command to interactively view text files that will behave as *man*:

```
# Run it, then press 'q' to exit:
less ~/examples/phage/vir_genomic.gff

# To disable wordwrap and see clearly the lines:
less -S ~/examples/phage/vir_genomic.gff
```

Counting lines

Counting the number of lines of a file is a common task. The wc (wordcount) command can do this, and something more.

```
# Count lines, words, characters of a file:
wc ~/examples/files/introduction.txt

# Count only lines:
wc -l ~/examples/files/introduction.txt

# Also on multiple files
wc -l ~/examples/phage/*.*
```

Extracting matching lines

`grep` is a powerful command to extract lines containing a pattern. The simples use is "grep wordtosearch file":

```
grep ">" ~/examples/phage/vir_protein.faa
```

In this case the word we looked for is simply the > character, that is, we extracted all the lines containing it. We are not going to expand this, but you can perform complex searches using a language called *regular expressions*.

Some switches: `-c` to count the number of matching lines, `-i` to perform a case insensitive search, `-v` to print the lines **not** containing the pattern.

- Presentation on regular expressions for grep

Redirecting the output

So far every command we issues gave us some text lines that we inspected, but we never saved them for long term storage. Consider the following command:

```
find ~/examples -type d
```

If we want to save the output in a new file, the shell offer us a redirection symbol:

```
find ~/examples -type d > ~/examples/directories.txt
```

With this command we created a new file, called ~/examples/directories.txt, where the output of find was stored. Note that if the file was already present, it would have been overwritten!

Our commands print two type of text

We explained the behaviour of most commands as a set of characters printed on our screen. This is a simplification: the characters printed can be either real output or user messages (technically called *standard output* and *standard error*). The '>' sign will redirect the standard output (or STDOUT), but sometimes we are interested in the standard error (or STDERR). Try:

```
weather.pl > ~/weather.out
```

What can you note?

```
weather.pl 2> ~/weather.err
```

Now you know how to redirect the standard error (i.e. using 2>).

Let's make a real world example: when we align short reads against the reference we expect the output to be the alignments (in SAM/BAM format), but the program can be interested in printing some user information (e.g. alignment progress, how many unmapped reads...), so will use the standard error.

Try the paths

Go to your home directory. Try counting the lines from two files you choose inside your home, plus

`/etc/passwd`. [4]

Now count the lines of `/etc/passwd`, but using a relative path! [5]

Go to the `~/examples/scripts/` and try to list the files included in the ~/examples/scripts/files, using the relative path. [6]

Finally, always from the `~/examples/scripts/` directory. Save into a file called *phage_files_lines.txt* placed inside your home the number of lines of each file inside the *examples/phage* directory. Use only relative paths. [7]

Answers

1) see [https://www.computerhope.com/unix/ucp.htm] for details on cp

2) there are 4 subdirs

3) *archives* contains 2 files, *files* 7, *phage* 17 and finally *scripts* 2

4) An example can be

```
wc -l examples/files/introduction.txt  examples/phage/md5checksums.txt
/etc/passwd
```

5) Something like

```
wc -l ../../../etc/passwd
```

6) like

```
ls -l ../files/
```

7) Example:

```
wc -l ../phage/*.* > ../../phage_files_lines.txt
```