SCIENCE MEETS LIFE

Linux command line

Initiation

VIB

2019

Christof De Bo

Agenda

UNIX fundamentals

UNIX file system

Command line + operations on files

Standard output, input and building pipelines

Text mining

Scripting

Course materials

Theory

https://training.vib.be/initiation-linux-command-line

Exercises

http://wiki.bits.vib.be/index.php/Introduction_to_Linux_for_bioinformatics

Linux training setup

VNC Viewer for Google Chrome

IP: x.x.x.x:590X (1-5)

IP: x.x.x.x:590X (1-5)

Password:

TrainingVIBX

What is Linux/this course for you?

Why use Linux?

What is the difference with other operating systems?

Only for bioinformaticians?

Is it really for free?

…?

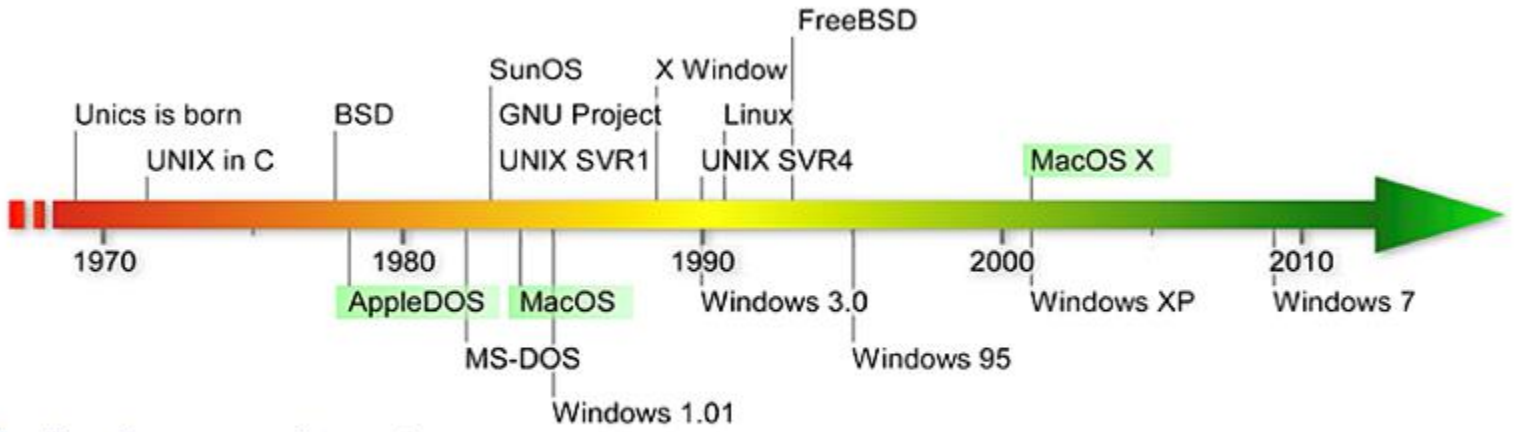
SCIENCE MEETS LIFE

UNIX fundamentals

VIB

UNIX family

UNIX is a family of computer operating systems



Timeline of some operating systems.
The timeline of some UNIX/UNIX-like (top) and other non-UNIX (bottom) operating systems is represented. One can notice that Apple's operating system evolved from non-UNIX (AppleDOS then Mac OS) to UNIX (OS X).

UNIX key features

Multiuser

Possible to connect to the same server (remote) and execute different programs at the same time

Multitasking

Multiple processes can run on the same server at the same time

Networking

The network is essential for remote access

Various user interfaces

Both text-only and graphical interfaces are available

UNIX OS

What?

Collection of software that manages computer hardware resources and provides common services for programs

Kernel

An OS kernel manages computer resources (e.g. CPU, RAM, internal/external devices) and allows programs to use these resources

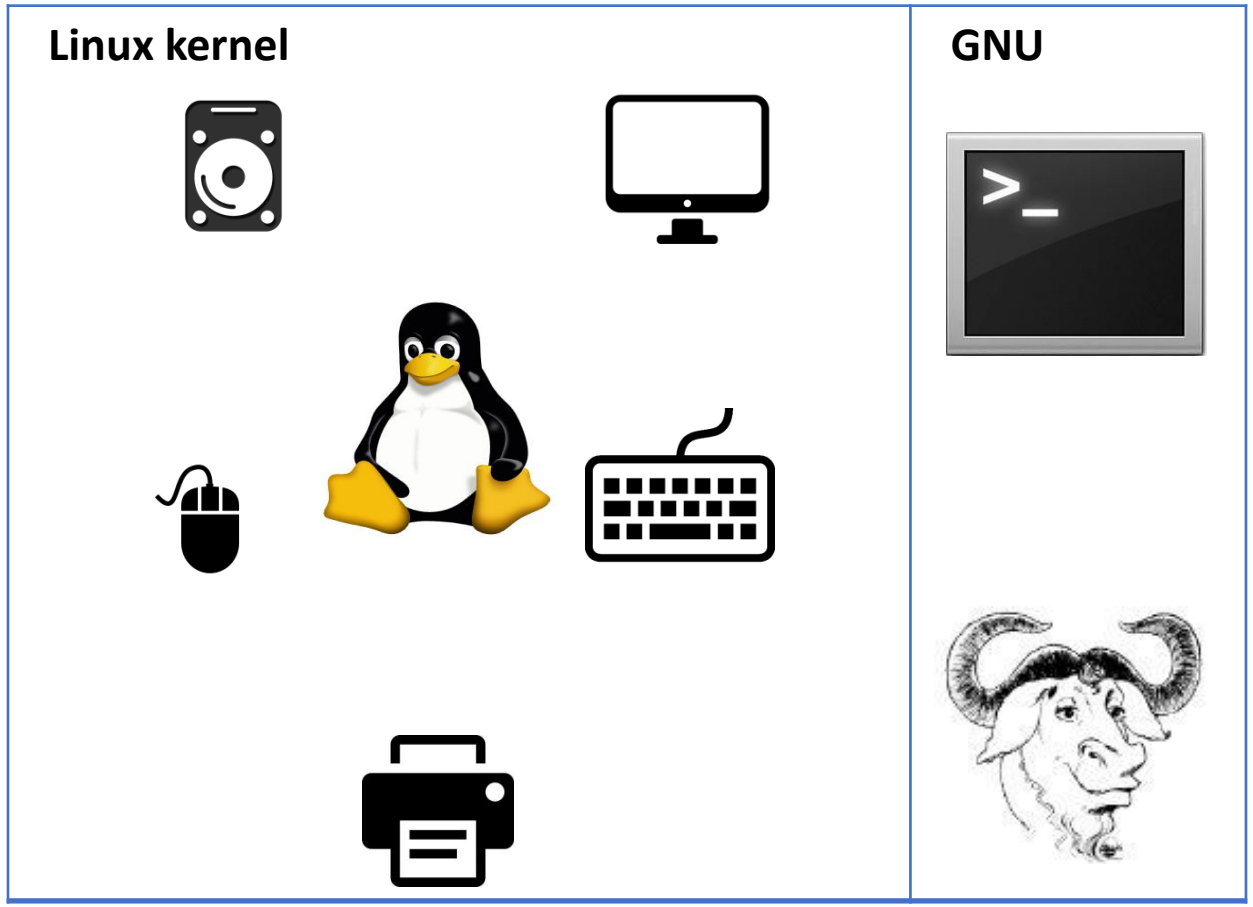
Shell

= Command Line Interpreter (CLI) is a text-only interface between the user and the kernel. Its function is to execute commands from the terminal window

Programs

External programs can be installed. The OS comes with many built-in utilities

GNU/Linux



Linux kernel

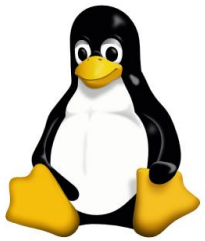
Takes care of managing computer recources

GNU

Provides the means of communication with the linux kernel through command

Linux distributions

Linux is a core operating system and can be used to build your own flavor if Linux. These flavors are called distributions

GNU/Linux	Ubuntu
 	

Distribution

Linux OS kernel
Graphical user interface
Tools and applications
Software manager
Configuration
Support (commercial)

+600 distributions in the world
(only a few are interesting)

Software in Linux

Open-source and free

just like Linux itself

Depending on the distribution

easier/harder to install ‘packages’ (= applications/apps)

More than 1 way to install software

depending on the distribution/tool

Software in Linux

- Software Manager
  - App store
  - Add custom repositories
- Installation files
  - Similar to Windows
  - Only use official websites
- Unpacked software
  - Dependencies!
- Executable binary files
  - Must math your system architecture
- Executable Scripts
  - e.g. Java, R, Python
- Command line

Software in Linux

Downloading and installing software can be done by using the command line:

Look for a tool in the command line:

```
bowtie
```

Result:

```
The program 'bowtie' is currently not installed. You can install it by
typing: sudo apt install bowtie
```

This is the command to install any tool from the command line

```
sudo apt install <program>
```

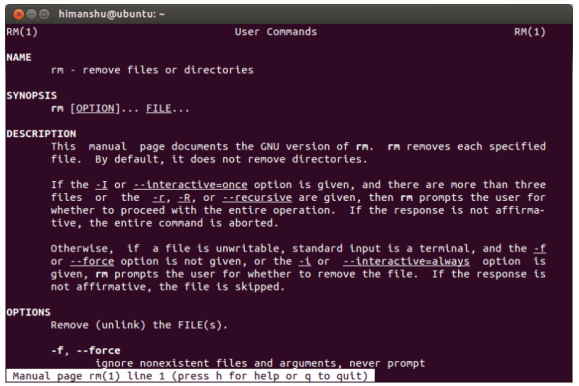
SCIENCE MEETS LIFE

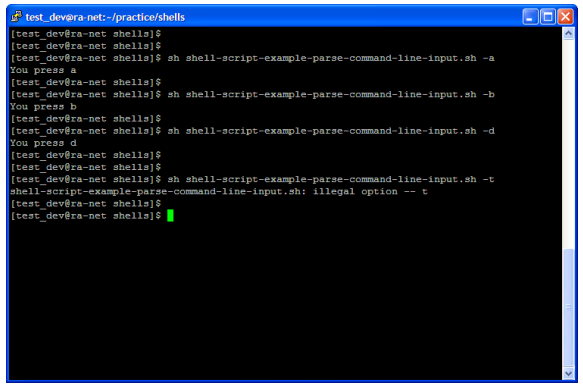
Command line

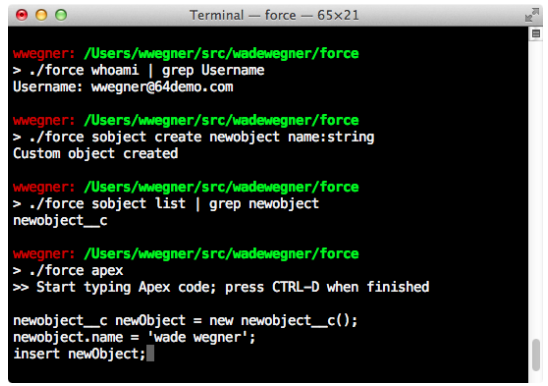
VIB

Command line interface

Command line interfaces existed way before the “fancy” Graphical User Interfaces (GUI) due to computer power/technology. Still the command line survived in all operating systems and is still useful!







Why use command line?

Programming language features (e.g. loops, variables, ...)

Commands can be assembled into scripts

Auto-completion

History of executed commands

Handle every file type (e.g. text, web resources, audio, ...)

Perform complex computation on remote servers

Display windows of remote application on your screen

...

Note: command-line and graphical user interface can be used at the same time, they co-exist and are available on the same time. Some tasks require or can only be achieved on one of them

Command line

House rules

The command line is always positioned somewhere in the file system (working directory)

The initial position is the home directory = ~

Everything you type is case-sensitive

First time you see the prompt line

```
username@machinename position $
```

After `$` comes your command

Executing a command = press Enter

Command line: Shell

Shell commands can be of four types

- Integrated into the shell (e.g. for, while,…)
- Binary executable programs
- Executable scripts (e.g. perl, python, R, …)
- Aliases (often used commands or “shortcuts”)

Shell commands have options (= parameters)

- Detailed control over the command
- Standard options / tool specific

Command line: Shell syntax

Command line convention used in this course

```
program -x -Y --option other_parameter
```

the program comes first, followed by space

options start with a '–' for short options or '––' for long options and they can be combined:

`–x –Y = –xY`

other parameters (arguments) follow at the end. This can be file- or directory names, sever addresses, etc.

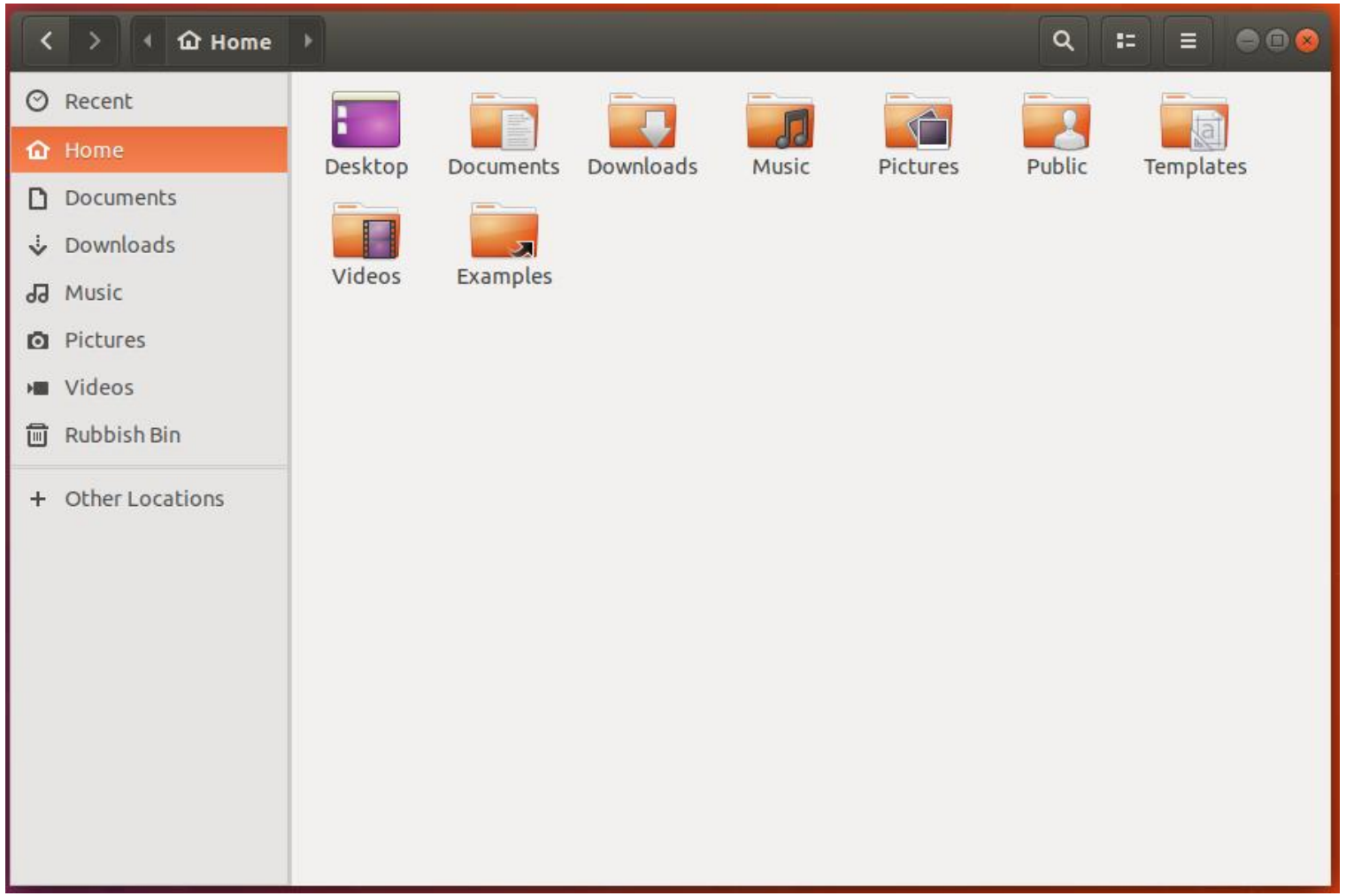
Tip: in the beginning use the cheat sheet or http://explainshell.com

SCIENCE MEETS LIFE

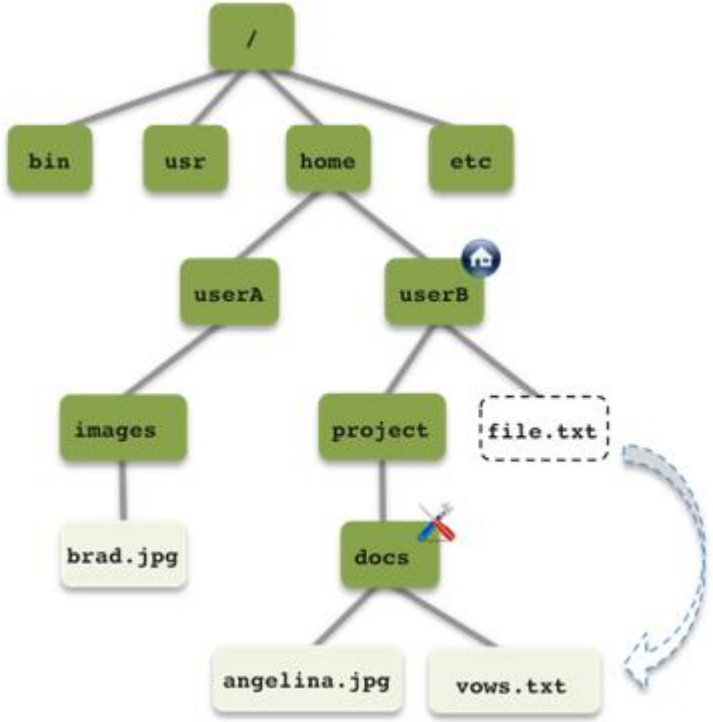
UNIX file system

VIB

UNIX file system: GUI



UNIX file system



File

Directory

Symbolic link

Home directory

Working directory

UNIX file system: files and extensions

For some operating systems, file extensions are important and define the file type. In e.g. Windows files have a three/four letter extension(e.g. .jpg, .exe, .docx, …)

In UNIX file extensions are arbitrary (no particular sense), at least for the operating system. A file can have several extensions and will be recognized by the file permissions and content.

Popular file extensions in bioinformatics:

.sh - .pl - .py	shell – perl – python scripts
.txt - .csv - .fasta	text – tabular – sequence files

UNIX file system: users and access rights

Every file and directory is protected. A set of permissions determines who can access a certain file and what kind of access is allowed.

User	the user who owns the file
Group	other users from the same group
Others	all others in the system

Read	display the file
Write	display and modify the content of the file
Execute	run a file (only for scripts and compiled programs)

UNIX file system: user, group and root

User

username

password

home directory

group

Group

primary group
(first group)

secondary group
(unlimited)

group name

Root

create and delete user accounts

modify access rights

install and remove programs

username = root

group name = root

Never work as root! Only use the root user when doing administrative tasks. On a server the root user will not be available. For special tasks, you can get root rights by using the command '`sudo`' followed by the rest of your command.

UNIX file system: commands

`pwd`

print working directory

`cd`

change directory

`ls`

list of content of the current directory

`chmod`

modify access rights

`touch`

make a file

UNIX file system: commands

`ls -a`

print a list of all files and directories (incl. hidden files). Hidden files can be recognized by '.' at the start of the name, '..' is the parent directory

`ls -l`

print a list of all files and directories in long format (incl. access rights)

```
unix-server:docs userB$ ls -l ~
total 0
lrwxrwxrwx 1 userB UXcourse 21 Jan 22 14:46 file.txt -> project/docs/vows.txt
drwxr-xr-x 3 userB UXcourse 17 Aug  1 08:59 project
unix-server:docs userB$
```

`cd ..`

change working directory to the parent directory

UNIX file system: access rights

`ls -l`

```
unix-server:docs userB$ ls -l ~
total 0
lrwxrwxrwx 1 userB UXcourse 21 Jan 22 14:46 file.txt -> project/docs/vows.txt
drwxr-xr-x 3 userB UXcourse 17 Aug  1 08:59 project
unix-server:docs userB$
```

_ ___ ___ ___ X owner group size date time filename (-> symbolic link)

object identifier user rights group rights access for others

object identifier	'-' (file), 'd' (directory), 'l' (symbolic link)
access rights	'r' (read), 'w' (write), x (execute) or '-' (absence of the permission)

UNIX file system: access rights

Access right can be modified using following command

```
chmod <ownership>'+/-'<access> filename
chmod <access-numbers> filename
```

ownership	user, group, and others (combination is possible)
access	read, write and execute
-numbers	read: 4, write: 2 and execute: 1
	r+w: 4 + 2 = 6, r+x: 4 + 1 = 5, r+w+x: 4 + 2 + 1 = 7
	each number refers to the ownership (ugo)

UNIX file system: access rights

```
chmod ugo+rw Smith.txt
```

Change permission for the user, group and others to read and write on the file Smith.txt

```
chmod 755 script.pl
```

Change permission for the user, group and others to read and execute the file Script.pl, only the user can write as well

UNIX file system

When entering a command you can use a wildcard character, this is used a substitute for one or many other characters. They are often used with file and directory name and filesystem commands

`*`	Match any number of characters
`?`	Match one character
`[ ]`	specify a range of characters on that position
`{ }`	specify a list of terms – separated by commas
`!`	Exclude this range of characters

UNIX file system: exercises

How many files will you find using the following command:

- `ls -l *.txt`
- `ls -l B*`
- `ls -l a[]a?.jpg`
- `ls -l *.*[!txt]`
- `ls -l {*.gif,*.png}`
- `ls -l Angelina[5-9].jpg`

```
unix-server:docs userB$ ls -l
-rw-r--r-- 1 userB UXcourse 2923151 Dec 31 23:59 angelina1.jpg
-rw-r--r-- 1 userB UXcourse 3601342 Dec 31 23:59 angelina2.jpg
-rw-r--r-- 1 userB UXcourse 3578959 Dec 31 23:59 angelina3.jpg
-rw-r--r-- 1 userB UXcourse 3280283 Dec 31 23:59 angelina4.jpg
-rw-r--r-- 1 userB UXcourse 3883053 Dec 31 23:59 angelina5.jpg
-rw-r--r-- 1 userB UXcourse 3551388 Dec 31 23:59 angelina6.jpg
-rw-r--r-- 1 userB UXcourse 3570123 Dec 31 23:59 angelina7.jpg
-rw-r--r-- 1 userB UXcourse 3095616 Dec 31 23:59 angelina8.jpg
-rw-r--r-- 1 userB UXcourse 3023818 Dec 31 23:59 angelina9.jpg
-rw-r--r-- 1 userB UXcourse  208488 Jan  1  2005 angelina.jpg
-rw-r--r-- 1 userB UXcourse   14529 Dec 31  2002 Billy.png
-rw-r--r-- 1 userB UXcourse   68745 Jan  1  2012 Brad_2012.png
-rw-r--r-- 1 userB UXcourse  329531 Jun  6  2008 Cannes_2008.txt
-rw-r--r-- 1 userB UXcourse   34537 Jul 12  2008 Knox.png
-rw-r--r-- 1 userB UXcourse    4262 Aug  1  2003 LaraC.txt
-rw-r--r-- 1 userB UXcourse   91481 Mar 15  2007 Pax.png
-rw-r--r-- 1 userB UXcourse   16901 May  1  2006 Shiloh.jpeg
-rw-r--r-- 1 userB UXcourse   11567 Aug  1  2005 Smith.txt
-rw-r--r-- 1 userB UXcourse    3556 Aug  1  2001 UNHCR.txt
-rw-r--r-- 1 userB UXcourse   36648 Jul 12  2008 Vivienne.png
-rwxr----- 1 userB UXcourse     861 Apr  1  2012 vows.txt
-rw-r--r-- 1 userB UXcourse    4872 Jul  6  2005 Zahara.gif
unix-server:docs userB$
```

Filesystem: exercises

Which one is not correct?

A) `ls -l -r -w 80 /userB`

B) `ls -rlw 80 /userB`

C) `ls -wrl 80 /userB`

D) `ls -w 80 -lr /userB`

UNIX file system: remote file organization

When connected to a remote server (with only command line interface available) the file organization will be done with commands. The most used commands in those case are:

`touch`	create a file
`cp`	copy
`mv`	move
`mkdir`	create directory
`rm`	remove file
`rmdir`	remove directory

UNIX file system: remote file organization

```
mkdir Data
```

Create a directory called ‘Data’ in the current working directory

```
mv  {*.jpg,*.png,*.gif} Pictures
```

Move all files with extension jpg, png and gif to directory ‘Pictures’

```
mv data.txt data_copy.txt
```

Rename data.txt to data.copy.txt in the current directory

```
cp data_copy.txt ~/
```

Copy data_copy.txt to the home directory

VIB

SCIENCE MEETS LIFE

UNIX file system: remote file organization

`rm ~/file.txt`

Remove file.txt from the home directory

`rmdir Data`

Remove directory 'Data' -> will only be possible when 'Data' in empty, otherwise -> error

`rmdir -f Data` or `rmdir --force Data`

Remove directory 'Data' including all content

`rmdir -r Data`

Remove directory 'Data' recursively

UNIX file system: remote file organization

Symbolic link files which points to the location of another file. You can do the same things with the original and with the link but when the file is moved from its location, the link is dead!

To create a symbolic link, move to the folder where the link must be created and execute following command:

```
ln -s ../folder/folder/file.txt link_to_file.txt
```

To delete a link, use 'unlink'

UNIX file system: remote file organization

There is more then one way to define the location of a file!

Relative path

relative to the current working directory (e.g. `/usr/local/`)

```
cd ../../home/user
cd Downloads
```

Absolute path

starting from the root directory

```
cd /home/user/Downloads
```

UNIX file system: download

wget

'wget' is a non-interactive network downloader. It downloads the file (link is argument) in the working directory. No need to browse to the website and save the file manually.

```
wget http://rapblegacy.jp/download/IRGSPb5.fa.gz
```

`-b`	go to background immediately after startup
`-w <seconds>`	wait for … seconds between the retrievals

UNIX file system: archive/compression

There a difference between creating an archive and compressing files. In UNIX you can create, update or expand an archive using 'tar'. Creating a compressed archive you need to include the bzip2 or gzip program.

```
tar mypictures.tar Pictures
```

archive the folder Pictures into a file called mypictures.tar

```
tar –zcvf mypictures.tar.gz Pictures
```

archive and compress the folder Pictures into a file called mypictures.tar.gz

UNIX file system: archive/compression

It not required but it's highly recommended to add .tar and .gz (when using gzip) or .bz2 (when using bzip2) extension for creating a compressed archive. Using filename extensions are useful for us (humans), for UNIX itself it doesn't matter.

-z	compress using gzip
-j	compress using bzip2
-c	create an archive
-x	extract from an archive
-v	print information on terminal
-f	use specified file
…	

UNIX file system: exercises

Wiki.bits.vib.be

- Command line
- File system

SCIENCE MEETS LIFE

Command line + operations on files

VIB

Finding files

The most used command for finding files is '`find`'

Find by name	`find ~ -name '*.'`
Find by size	`find ~ -name '*' -size +5M`
Find type	`find ~ -name '*o*' -type d`

(**f**ile, **d**irectory, **l**ink)

Finding files

Other option to find files and do many other things…

`-perm`

permission (111 or rwx)

`-exec`

execute a command on the found entities

e.g.

```
$ find -name *.gz -exec gunzip {}
```

File determination

As seen before, extensions are an option, nevertheless very useful. There are a series of common extension used by convention to make it easy for everyone.

To determine the file type (with our without extension), use this command:

```
file project/Pictures/Billy.png
```

Result:

```
project/Pictures/Billy.png: PNG image data, 300 x 158, 8-bits/clor
RGBA, non-interlaced
```

Common used extensions

.txt .csv .tab .fasta .seq .dna .sh .pl .py ...

File content

Only need a certain amount of lines at the beginning or end of your file?

`head data.txt`

first 10 lines of data.txt

`tail data.txt`

last 10 lines of data.txt

`head -50 data.txt`

first 50 lines of data.txt

`wc data.txt`

print the number of lines found in the file

File content

Want to browse through the entire file?

One of the commands to use is ‘cat’. This will display all content at once (not recommended for large files!) on the terminal.

```
cat data.txt
```

Another is ‘less’. This will display all content one screen at a time, like a text viewer (not a text editor). It will not read the entire content at once, which makes it faster.

```
less data.txt
```

File formats

Not every file is just plain text, some are comma- or tab-delimited or fasta format. There are some very useful command to retrieve information from these files.

e.g. csv files contains data is a comma-separated way, so the command ‘wc’ will print the number of words, but zero lines…

Try the command ‘less’ to look at a csv file

You will see several unexpected characters (e.g. ^M = MacOS newline character)

File formats

Delimiter separated values

tab – tsv

';' '|' ' '

FASTA

'>' header line character

sequence ends when a new '>' character appears

Swiss-Prot

every lines starts with a 2-character 'ID' and ends with '//'

File formats

There is a diversity of formats, even between operating systems, as you can see in the newline character

\n	UNIX (all flavors)
\r	MacOS
\r\n	Windows

Sometimes you need to convert text into a UNIX format to make it readable for Linux, then you can use this command

```
dos2unix
```

File formats

A second way of converting a file from Windows or MacOS to UNIX is the 'tr' command. This will translate or delete characters from any given file.

```
tr -d '\r' file.csv
```

delete all '\r' characters from file.csv

```
tr '[a-z]' '[A-Z]' < file.csv
```

transform all lower case characters to the corresponding upper case

Note: standard output is the terminal, redirection to another file is recommended!

File input/output

Under Linux there are three standard streams:

Standard input

channel (0) where programs receive their data, mostly this is the keyboard (unless redirected)

Standard output

channel (1) where programs print there outcome, by default this is the terminal window

Standard error

channel (2) where programs write output messages, by default this is the terminal window

File input/output

Redirecting input

Some commands take their input from the keyboard (standard input stream), but this can be modified. The input can be received from a file for example. To redirect the input of a command, use '<' or '0<' proceeded by the location/name of that file.

```
cat ID_B.txt
```

display the content of the file

```
sort < ID_B.txt
```

display the sorted content of the file without modifying the file itself

File input/output

When using the input redirection, '<' replaces the standard keyboard source by a file. In that case, the program expects only one source. Therefore, commands where multiple files are given as input, such as:

```
sort < ID*
```

will raise the following error: ID*: ambiguous redirect

However, many programs can handle several files when these are passed as argument – enumerated - on the command line :

```
sort ID_A.txt ID_B.txt ID_C.txt     /     sort ID*
```

File input/output

Redirecting output

Most UNIX programs print the output on the standard output(the terminal). To redirect the output to a file or create a new file you can use '>' , '1>' or '>>'

```
> file.txt
```

a new file is created, existing file is overwritten with output from the command

```
>> file.txt
```

a new file is created if it does not exist. Otherwise, the output is appended at the end of the existing file

File input/output

Redirecting standard error

Sometimes a commands outputs some warnings and errors on the screen. This is not standard output but standard error (channel 2). In the same way you redirect channel 1 to a file, you can export channel to a 'error.txt' file.

```
~ $ ls -lR / 2> error.txt
```

Note that if you write to a file, the content of that file will be replaced! ('>' vs '>>')

File input/output

INPUT

< filename	input from file
<< EOF	input until string EOF
<<<"this string is read"	input directly from string

OUTPUT

> filename	output to file
>> filename	append output to file

ERROR

2> /dev/null	output error to 'bit-heaven'

File input/output

The pipe '|'

We are able to retrieve the input from a file and redirect the output to another file. Sometimes we just want to redirect the output from one command directly to the input of another without creating a temporary file(s). Then we can use the pipe '|'

```
sort -u Data.txt | wc -l
```

print the number of unique lines of a file without creating any file

File input/output

Try the following command:

```
history | awk '{print $2}' |  sort | uniq -c | sort -nr | head -3
```

print the top 3 most popular commands from your history sorted from most to less popular.

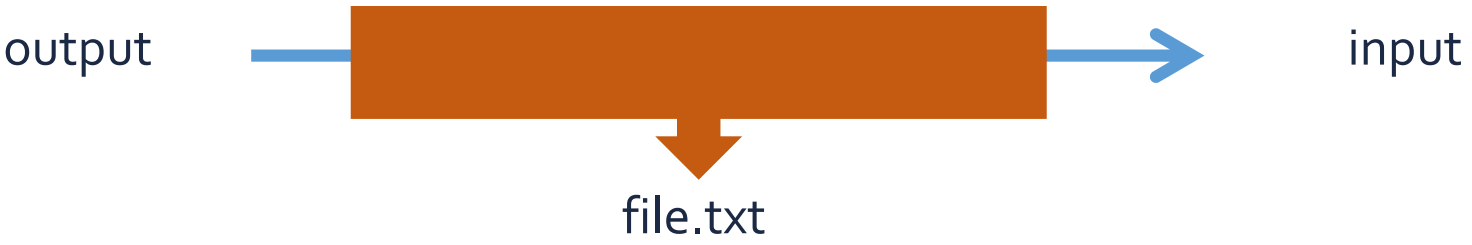
```
237 ls
180 cd
103 file
```

File input/output

tee

This will take the standard input (channel 0) and send it to both standard output (channel 1) and any file given

```
ls -l | tee file.txt | less
```



Multitasking with commands

When using the command line, you can run multiple commands in one line. The commands are then separated by ';'

```
$ wget http://homepage.tudelft.nl/19j49/t- SNE_files/tSNE_linux.tar.gz ; tar
-xvfz tSNE_linux.tar.gz
```

Other command separators

`&&`	only execute the command if the preceding one finished correctly
`||`	only execute the command if the preceding one didn't finish correctly (= plan B)

Background commands

A process can…

- run in the foreground
  - Command prompt is not available as long as the process is running
- run in the background
  - Command prompt is available, more processes can be started
- be suspended (status: stopped)
  - Process is paused
- be killed (status: terminated)
  - Process was frozen or gone into a loop -> has to be stopped

A process = running program and is identified by a unique process identifier (PID)

Background commands

When running a command or script, you sometimes have to wait until the command prompt is available again. The bigger the files/longer the script, the longer the wait…

You can open a second terminal, and a third, … or you can run your command in the background:

```
command &
```

e.g.

```
scripts/script_convert.py arg1 arg2 &
command1 | command2 > /Results/output.txt &
wget https://website.com/data/genome.fa &
```

Background commands

How do you manage all the processes running in the background? How to I bring them back to the front or terminate them?

`jobs -l`

list of all running processes (+ PID)

`fg x or fg %x`

bring the process with PID = x to the foreground

`kill x`

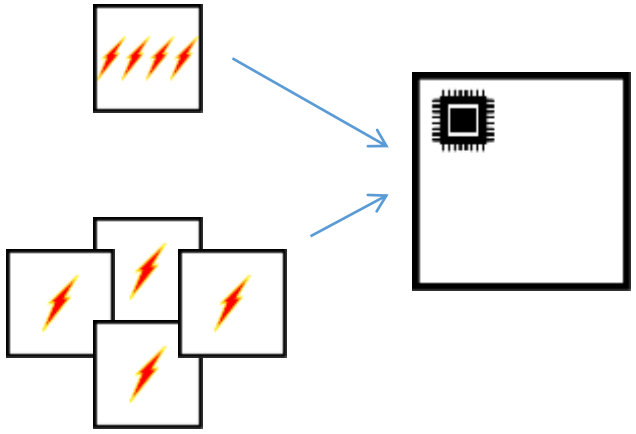
terminate background process with PID = x

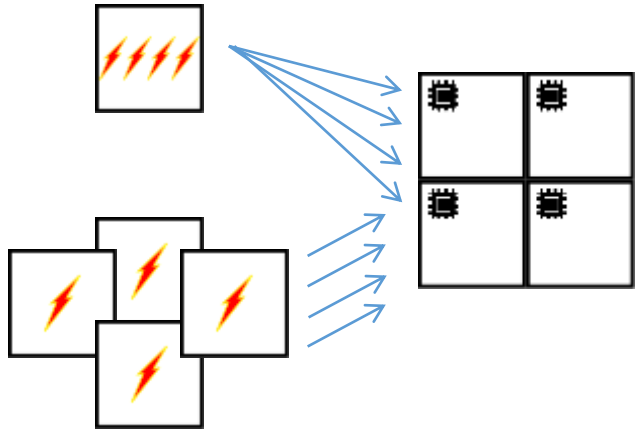
To send a command back to the background while it is running press ‘Ctrl+Z’ (suspend the process) and type ‘bg’

Processes and multitasking

Multiple processes can be run at the same time. These parallel processes will eventually slow down your computer. With a multi-core CPU you are able to divide the load between those cores, this is called multithreading.

A thread is a unit of execution that is contained in a process. The more cores, the more threads can be executed at the same time!





Processes and multitasking

Some commands that can help you with multitasking and multithreading:

`parallel`

run programs in parallel

`bowtie --threads x`

define number of threads (x) to run bowtie

SCIENCE MEETS LIFE

Text mining

VIB

File statistics

What is the size of my file (words, lines, characters)? The command to use is '`wc`'

`-l`	number of lines
`-w`	number of words
`-c`	number of bytes
`-m`	number of characters

More information on the files: `ls -lh`

File statistics

Can I compare files with each other? The command ‘diff’ compares two files and prints the differences. Some useful options are:

`-b`	ignore changes in white space
`-B`	ignore changes in blank lines
`-s`	report when two files are identical (default not shown)
`-q`	reports if files differ without any details of the differences

Even when you have an identical output when using the ‘wc’ command on two files, ‘diff’ can prove that the content is not identical!

Search file content

The ‘grep’ command is a powerful way to search a word or pattern in files. This command comes with several options, including:

-c	printing the number of resulting lines
-I	case-insensitive search
-v	print lines that not match the pattern
-n	add lines number in front of result
-r/-R	search all files recursively under each directory

e.g.

```
grep –c ‘course’ draft.txt
grep ‘course’ draft.txt
grep –r ‘course’ ~
```

Search file content

Task: find the number of entries of different file formats. Luckily it's easy to search into structured files! Some examples:

FASTA

```
grep –c '^>' file.fa
```

Swiss-Prot

```
grep –c '^ID' file.txt
```

CSV

```
grep –c '.' file.csv
```

TSV

```
grep –c '.' file.tab
```

Search file content

How to extract selected parts of lines from a file

cut

How to report one line if several identical and adjacent lines are found

uniq

Search file content

Extract file content

'awk' is a very useful command in file manipulation. It can scan for patterns and carry out associated actions when the pattern is found.

```
awk –F delimiter '{print $x }'
```

-F delimiter	field separator (default: white space)
$x	field number ($0: complete line)

Search file content

File conversion

e.g.: a .bed file needs to be converted into a .gff file. With ‘awk’ this is very easy!

.bed

```
chr1 2025600 2027271 AT1G06620.10 + 2025617 2027094 0 3541,322,429, 0,833,1242
```

.gff

```
chr1 Ensembl Repeat 2419108 2419128 42 . . hid=trf; hstart=1; hend=21
```

Search file content

```
chr1 2025600 2027271 AT1G06620.10 + 2025617 2027094 0 3541,322,429, 0,833,1242
```

```
$ awk '{print $1"\tawk\tmRNA\t"$2"\t"$3"\t" $5"\t"$6"\t0\t"$4 }' TAIR9_mRNA.bed
```

```
chr1    awk    mRNA    2025600        2027271        0      +      0      AT1G06620.1
```

SCIENCE MEETS LIFE

Scripting

VIB

Why use scripts?

Nothing can stop automation!

Scripts

Scripts are a series of commands that can be executed after each other. Let's try!

Make a file in your ~ with the name 'space_left'. Enter these lines:

```
df -h
echo "======="
du -sh */
```

Now execute the script:

```
bash space_left
```

Scripts

To convert a simple text file into a script, you need to ass a shebang as first line, this will tell the terminal which program should read and execute this text file.

```
#!/bin/bash
#!/bin/perl
#!/bin/python
...
```

Note: make your text-file/script executable before you try to launch it!

```
chmod ugo+x space_left
```

Scripts: arguments

You can pass on arguments to a script: they are stored in variables called `$1`, `$2`, …

Make a file called 'arguments.sh' with following content:

```
#!/bin/bash
firstarg=$1
secondarg=$2
echo "You have entered \"$firstarg\" and \"$secondarg\""
```

Now execute your script

Scripts: arguments

Now execute your script followed by the arguments:

```
./agurments five six seven eight
You have entered “five” and “six”
```

The string after the command is chopped on the white spaces. Let’s try other options…

Scripts: arguments

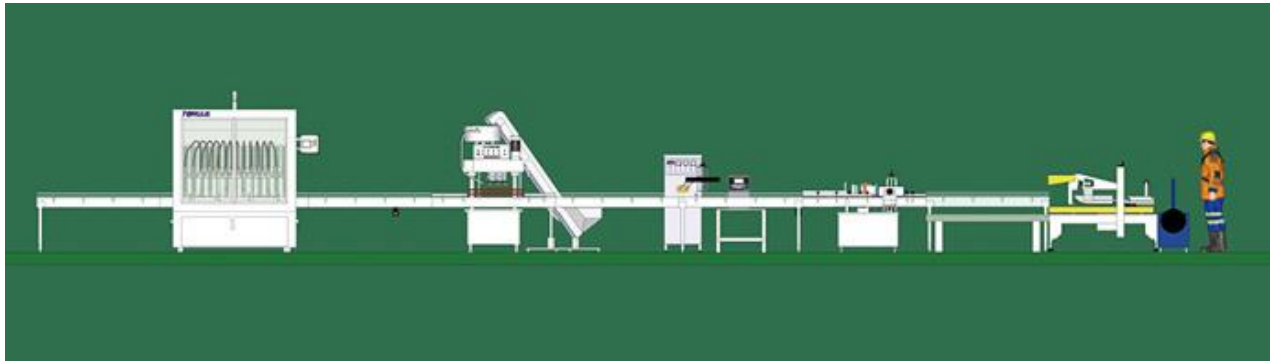
```
./arguments “five six” seven eight
You have entered “five six” and “seven”

./arguments five\ six\ seven\ eight
You have entered “five six seven eight” and “”

./arguments “five six seven eight”
You have entered “five six seven eight” and “”
```

Scripts

Comment character

- Lines starting with # are ignored (except #!)
- Use comments to explain what the script is doing and what the input and output is

Scripts perform a small unique task. Many scripts chained after each other create a bigger piece of software, similar to an assembly line.



Scripts: sharing and code development

To facilitate collaboration, you can develop and share your scripts in a version controlled online repository. These repositories work via command line tools like ‘git’ or ‘mercurial’.

www.github.com

www.BitBucker.com

SCIENCE MEETS LIFE

Tips and tricks

VIB

Tips and tricks

How to copy files between Windows and Linux?

Copy a file from a remote host to the local host

```
scp your_username@remotehost.edu:foobar.txt /some/local/directory
```

Copy a file from a local host to a remote host

```
scp foobar.txt your_username@remotehost.edu:/some/remote/directory
```

Tips and tricks

locate

Very quick and helpful way to find the location of files. It won't find the newest files you created, first update the database by running

```
updatedb
```

e.g.

```
locate *.txt
```


VIB
science meets life