Session I: Introduction to Linux/Unix and the command line
Learning Objectives

① Introduction to Linux and Unix
② The Unix files structure
③ Creating directories and navigating through the file structure
④ Some useful shortcuts and links
Learning Outcomes

① Understand the Unix file structure
② Learn how to create and access files and directories and navigate through them
③ Understand full and relative file paths: when and how to use them
④ Useful shortcuts
Introduction to Linux and UNIX
What is Linux?

- UNIX is an **Operating System** (OS) initially developed in the 1960.
- There are many different versions of UNIX, that share common similarities.
- The most popular varieties of UNIX are Solaris, **Linux** and MacOS.
- UNIX systems have a graphical user interface (GUI) making easier the environment.
Linux vs Unix

- Linux is a “clone” of the original Unix but doesn’t contain its code
- Linux is free and open source, the original Unix is not (but some of its derivatives are)
- All command lines work the same on both
Why Linux?

- Linux is free and the most popular distributions are Ubuntu, Fedora/Red Hat, Mandriva, etc.
- Low cost and very stable system
- Most secure OS
- Best multi-user and multi-tasking OS
- The world’s fastest super computers run Linux
- Fast developing OS (many developers)

Very popular as servers OS (stability, reliability)
Linux distributions

• Different Linux distributions are available http://distrowatch.com/

• Ubuntu distribution is easy and convenient to use for beginners

• A simple guide to install Ubuntu in your machine: http://www.ubuntu.com/download/desktop/install-ubuntu-desktop
Parts of the UNIX System

- The core operating system
- Accessible via the Terminal
- Programs and tools

Adapted from: www.usna.edu
The Terminal

• A **terminal** refers to a wrapper program which runs a shell

• There are many different Unix shells, the most popular shell for interactive use include **Bash**: the default on most Linux installations
The terminal

Shell prompt

- User name
- Machine name

Even though it’s is a command line interface, the mouse is still handy (scroll, copy, paste, etc.)
File-system under UNIX
Linux files structure

- /: Root directory of the entire file system
  - home: Users home directories
  - dev: Device files
  - bin: Essential user commands binaries
  - etc: Configuration files

- Crick
- Watson
- Sanger

- Desktop
- Documents
- IBT2017
  - Genomics
  - Linux
    - Session 1
    - Session 2

Wastson’s files
• When you first log in on a UNIX system, the working directory is your home directory.

• While working you will be associated to one directory called the working directory or the current directory.

• An abbreviation of the working directory is displayed as part of the prompt on your terminal.

• The command `pwd` gives the absolute path of the working directory.
What is a path or a pathname?

- A path locates a given file in the system hierarchy
- An **absolute path** in the file system hierarchy for a given file or folder describes the parents all the way up to the root
- A **relative path** describes the path to the file starting from the current working directory
~ (your home directory)

- ~ refers to the home directory in a given file system
- The tilde ~ character can be used to specify paths starting at your home directory
The full path to file1 is:
/home/Watson/IBT2017/Linux/Session1/file1
Absolute path?

```
/home
/home/Watson
/home/Watson/IBT2017
/home/Watson/IBT2017/Linux
/home/Watson/IBT2017/Linux/Session1
/home/Watson/IBT2017/Linux/Session1/file1
/home/Watson/IBT2017/Linux/Session1/file2
```
Every directory has two special sub-directories:

- . (dot): the current directory
- .. (dot-dot): the parent directory
Relative path?

```
/home
 |
/  
 |
Watson
 |
Desktop  Documents  IBT2017
 |
Genomics  Linux
 |
./Linux
 |
Session 1  Session 2
 |
file1  file2  ./Linux/Session1/file1
```
First test of the terminal

- Open the Terminal on your system
- The shell prompt will appear

Ok, let’s try some typing!
Creating directories and navigating through the file structure
## Commands for manipulating directories

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mkdir</code></td>
<td>Make directory: creates a new directory</td>
</tr>
<tr>
<td><code>rmdir</code></td>
<td>Removes a directory</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Displays the absolute path of the current working directory</td>
</tr>
<tr>
<td><code>cd</code></td>
<td>Change directory: allows moving from one directory to another</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Lists a directory content</td>
</tr>
</tbody>
</table>
pwd command

- **pwd**: print working directory
- Displays the **absolute path** of your current location in the file system
- Try `pwd` on your terminal
- You should see: `/home/YourUsername`
Is command

• `ls` lists the content of the current directory by default

• Command structure: `ls [OPTION] [dirname]

• Some useful options:
  • `-l`: shows sizes, modified date and time, file or folder name and owner of file and permissions
  • `-a`: List all files including hidden file starting with ‘.’
  • `-lh`: shows sizes in easier readable format
  • `-R`: recursively lists sub-directories
  • `-IS`: sorting by file sizes
Create a directory

- **mkdir**: makes a directory
- Command structure: `mkdir dir_name [path]`

- `mkdir dir_name`: would create a directory with the specified `dir_name`
- The new created directory will be created in your current working directory
- If you want to create it elsewhere, you have to specify the path: `mkdir dir_name path`
Commands basic structure

command [-options] [arguments]

Example:

ls -lh /home/Watson/IBT2017
pwd
mkdir Test1
What you should know about file names in Linux

• No real distinction between the names of ordinary files and the names of directory files.
• No two files in the same directory can have the same name.
• Files in different directories can have the same name.
• Linux is case-sensitive: Sanger, sanger and SANGER are different and would represent three distinct files.
• In most cases, file extensions are optional (.txt, .exe, etc.)
Move in the files system

- **cd**: change the working directory
- Command structure: `cd <path>`

- The path name of the directory you want to move to should be specified
- You can specify either the absolute path or the relative path
Move in the files system: example

• Move to Watson directory
  1. cd /home/Watson
  2. cd ../..
  3. cd ..

• Move to Genomics directory
  1. cd /home/Watson/IBT2017/Genomics
  2. cd ../Genomics
  3. cd ..
     cd Genomics
Remove a directory

- `rmdir`: removes a directory
- Command structure: `rmdir dirname [path]`
- It would remove the `dirname` directory
- The directory should be in your current working directory
- If you want to remove it from elsewhere, you have to specify the path: `rmdir dirname path`
- `rmdir` works if there is no contents in the directory
Remove a directory

• `rmdir` works if there is no contents in the directory
• If the directory contains files or sub-directories, an error message will appear: “Directory not empty”
• There is an option to remove `-r`, which stands for recursive, that will recursively remove a directory and its contents
How to get help for a command from the terminal?

• `man commandname` displays
Some useful shortcuts and Links
Useful Links

- **cd**: cd followed by nothing will change the working directory to your home directory
- **cd ~user_name**: moves to the specified user home directory
Linux, some useful CLI key combinations

- **Ctrl+A**: remove the cursor to the beginning of the command line
- **Ctrl+C**: end a running program and return to the prompt
- **Ctrl+D**: logout from the current shell session: equivalent to exit
- **Tab**: autocomplete a file name
- **Tab Tab**: displays command completion possibilities
- **Ctrl+L**: clear the terminal
Exercícios

• https://github.com/amelgh/Introduction_To_Linux
Session II: Manipulating files
Learning Objectives

① Creating new files
② Basic manipulating files and directories commands
③ Viewing files contents
Learning Objectives

① Learn how to create new files
② Learn how to view files content
③ Learn basic commands to manage files
④ Learn some useful wildcards
Creating new files
Basics manipulating file commands

• **touch** is used to create, change and modify timestamps of a file

• **touch** command creates an empty (zero byte) new file using this
  
  Structure: `touch filename`

• Create more than one single file
  
  `touch filename1 filename2 filename3`
touch command options

• `-a`: change the access time only
• `-c`: if the file does not exist, do not create it
• `-d`: update the access and modification times
• `-m`: change the modification time only
• `-r`: use the access and modification times of file
• `-t`: creates a file using a specified time
Text editors

- **nano**: a simple and easy-to-use text editor
- Is installed by default in Ubuntu and many other Linux distros
- It’s a WYSIWYG editor: “what you see is what you get.” What you type directly goes into the text input
- **vim, emacs, gedit, Geany**: excellent programs but do require some learning
Get started with nano

- **name of the program and version number,**
- **the name of the file you are editing,**
- **shortcuts**
Let’s play with nano

- Open the terminal
- Log in (/home/Username)
- Type `pwd` to check
- Move to the directory Session1 directory under the Linux

```
cd Session1
```
```
cd ./IBT2017/Linux/Session1
```
Get started with nano

• `nano file1`
• Type “my first test file with webminal”
• Hit enter to move to another line and type “the second line of test”
• One you finish typing, hit Ctrl+x
• Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
• Hit Y
• `nano file2`
• Type “my second test file with webminal” and any other 4 lines of text
Some nano shortcuts

• To search for a text string, hit **Ctrl+W**, and enter your search term
• This search can then be cancelled mid-execution by hitting **Ctrl+C** without destroying your buffer
• **Ctrl+X**: finish typing and close an open file

Remember: **nano** pathnamen

- Opens the file if it’s existing already, you can modify and save changes
- Creates a new file in the specified path if it does not exist
Basic manipulating file commands
Displaying whole content of a file or parts of it (default + options)

- **cat**: view the content of a short file
  
syntax `cat <filename>`

- **more**: view the content of a long file and navigate through it
  
syntax `more <filename>`

- **less**: view the content of a long file, by portions
  
syntax `less <filename>`

- **head**: view the first lines of a long file
  
syntax `head <filename>`

- **tail**: view the last lines of a long file
  
syntax `tail <filename>`
View file content: less command

- **less** command displays a text file content, one page at a time
- **Structure**: `less filename`
- Move a page down: either use the page down key or **space**
- To **exit** less, type **q**
- To go to the end of the text file, type **g**
• **head** command displays a text file content, by default: 10 first lines at a time
• Syntax: `head <options> <filename>`

• **tail** command displays a text file content, by default: 10 last lines at a time
• Syntax: `tail <options> <filename>`
Basic manipulating file commands

- **Copy, move and remove**

- **cp**: copy files and directories
  Structure `cp <pathfrom> <path to>`

- **mv**: move or rename files and directories
  Structure `mv <pathfrom> <path to>`

- **rm**: remove files and directories
  Structure `rm pathname`
Copying command: `cp`

- Simplest form: `cp file1 file2`
  
  ➡ Copy the contents of file1 into file2. If file2 does not exist, it is created. Otherwise, file2 is silently overwritten with the contents of file1.

- `cp filename dirpath`
  
  ➡ Make a copy of the file (or directory) into the specified destination directory
Other examples: cp

• Add the interactive mode with the option -i
• `cp -i file1 file2`
  ➔ Same as the previous one. However, if file2 exists, the user is notified before overwriting file2 with the content of file1
• `cp –R pathdir1 pathdir2`
  ➔ Copy the contents of the directory dir1. If directory dir2 does not exist, it is created. Otherwise, it creates a directory named dir1 within directory dir2
The `mv` command moves or renames files and directories depending on how it is used:

- **To rename a file:**
  
  ```sh
  mv filename1 filename2
  ```

  If file2 exists, its contents are silently replaced with the contents of file1. To avoid overwriting, use the interactive mode:

  ```sh
  mv -i filename1 filename2
  ```

- **To move a file (or a directory) to another directory:**
  
  ```sh
  mv file dirpath
  ```

- **To move different files (or a directory) to another directory:**
  
  ```sh
  mv file1 file2 file3 dirpath
  ```

- **To move directory to another directory:**
  
  ```sh
  mv dir1 dir2
  ```

  If dir2 does not exist, then dir1 is renamed dir2. If dir2 exists, the directory dir1 is moved within directory dir2.
The *rm* command *deletes* files and directories

**To remove a file:**

```
rmd filename
```

**To remove many files:**

```
rmd filename1 filename2
```

Add the interactive mode to prompt user before deleting with **–i**

```
rmd -i filename1 filename2
```

Delete directories with all their contents

```
rmd -r dir1 dir2
```
Be careful with rm!

• Linux *does not have an undelete command*
• Once you delete something with `rm`, it's gone!
• You can inflict terrific damage on your system with `rm` if you are not careful, particularly with wildcards
• Try this trick before using `rm`: *construct your command using `ls` instead first*
Wildcards

• Since the shell uses filenames so much, it provides special characters to help rapidly specifying groups of filenames

• A group of special characters are called wildcards allow selecting filenames based on pattern of characters
## Wildcards examples

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>Any file name starting with a</td>
</tr>
<tr>
<td>*</td>
<td>All possible filenames</td>
</tr>
<tr>
<td>A*.fasta</td>
<td>All filenames that begin with A and end with .fasta</td>
</tr>
<tr>
<td>???? .vcf</td>
<td>Any filenames that contain exactly 4 characters and end with .vcf</td>
</tr>
<tr>
<td>[abc]*</td>
<td>Any filename that begins with &quot;a&quot; or &quot;b&quot; or &quot;c&quot; followed by any other characters</td>
</tr>
<tr>
<td>[:upper:]*</td>
<td>Any filename that begins with an uppercase letter. This is an example of a character class</td>
</tr>
</tbody>
</table>
## Wildcards

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches any characters</td>
</tr>
<tr>
<td>?</td>
<td>Matches any single character</td>
</tr>
<tr>
<td>![characters]</td>
<td>Matches any character that is not a member of the set characters</td>
</tr>
</tbody>
</table>
| [characters]   | Matches any character that is a member of the set characters. The set of characters may also be expressed as a POSIX character class such as one of the following:  
|                | [:alnum:] Alphanumeric characters                                        |
|                | [:alpha:] Alphabetic characters                                           |
|                | [:digit:] Numerals                                                       |
|                | [:upper:] Uppercase alphabetic characters                                |
|                | [:lower:] Lowercase alphabetic characters                                |

Source: [http://linuxcommand.org](http://linuxcommand.org)
Download files from the web

• **wget** stands for "web get". It is a command line utility which downloads files over a network.

• Go under Genomics

• **wget** `http://plasmodb.org/common/downloads/Current_Release/Pfalciparum3D7/fasta/data/PlasmoDB-28_Pfalciparum3D7_AnnotatedProteins.fasta`
Session III:
Extracting information from files
Learning Objectives

① Learn how to search patterns in files and how to extract specific data
② Learn how to sort files content
③ Learn basic commands to compare files content
④ Learn results redirection
⑤ Learn commands combination
Learning Outcomes

① Be able to search patterns in files extract specific data
② Be able to sort files content
③ Be able to use some basic commands to compare files content
④ Know how to write commands results into a file
⑤ Be able to combine different commands
Basic operations on files and data extraction
Some statistics about your file content: wc command

- **wc** prints newline, word, and byte counts for each file
- **syntax**: `wc <options> <filename>`
- **Some useful options**:
  - `-c`: prints the byte counts
  - `-m`: prints the character counts
  - `-l`: prints the newline counts
- For more info about the different commands use `man commandname`
Basics operation on files

- **sort**: reorder the content of a file “alphabetically”
  
syntax: `sort <filename>`

- **uniq**: removes duplicated lines
  
syntax: `uniq <filename>`

- **join**: compare the contents of 2 files, outputs the common entries
  
syntax: `join <filename1> <filename2>`

- **diff**: compare the contents of 2 files, outputs the differences
  
syntax: `diff <filename1> <filename2>`
Sorting data

• **sort** outputs a sorted order of the file content based on a specified sort key (default: takes entire input)

• Syntax: `sort <options> <filename>`

• Default field separator: **Blank**

• Sorted files are used as an input for several other commands so sort is often used in combination to other commands

• For `<options>` see **man**
Sorting data: examples

- Sort alphabetically (default option): `sort <filename>`
- Sort numerically: `sort -n <filename>`
- Sort on a specific column (n°4): `sort -k 4 <filename>`
- Sort based on a tab separator: `sort -t $'\t' <filename>`
- ...

Extracting data from files

- **grep**: to search for the occurrence of a specific pattern (regular expression using the wildcards...) in a file
  
  Syntax: `grep <pattern> <filename>`

- **cut**: is used to extract specific fields from a file

  Syntax: `cut <options> <filename>`
grep command

- **grep** ("global regular expression profile") is used to search for the occurrence of a specific pattern (regular expression...) in a file
- Grep output the whole line containing that pattern
- For `<options>` see man

**Example:**

Extract lines containing the pattern `xxx` from a file:

```
grep xxx <filename>
```

Extract lines that do not contain pattern `xxx` from a file:

```
grep -v xxx <filename>
```
Let’s consider a file named “ghandi.txt”

$ cat ghandi.txt

The difference between what we do and what we are capable of doing would suffice to solve most of the world's problems

$ grep what ghandi.txt

The difference between what we do and what we are capable of doing

$ grep -v what ghandi.txt

would suffice to solve most of the world's problems
cut command

• **cut** is used to extract specific fields from a file
• Structure: `cut <options> <filename>`
• For `<options>` see `man`
• Important options are
  - `-d` (field delimiter)
  - `-f` (field specifier)

*Example:*

`extract fields 2 and 3 from a file having ‘space’ as a separator`

`cut -d’ ‘ -f2,3 <filename>`
uniq command

• **uniq** outputs a file with no duplicated lines
• Uniq requires a sorted file as an input
• Syntax: **uniq** <options> <sorted_filename>
• For <options> see man
• Useful option is -c to output each line with its number of repeats
Join command

• **join** is used to compare 2 input files based on the entries in a common field (called “join field”) and outputs a merged file
• join requires **sorted files** as an input
• Lines with identical “join field” will be present **only once** in the output
• Structure:
  ```
  join <options> <filename1> <filename2>
  ```
• For **<options>** see man
diff command

- **diff** is used to compare 2 input files and displays the different entries
- Can be used to highlight differences between 2 versions of the same file
- Default output: common lines not showed, only different lines are indicated and shows what has been added (**a**), deleted (**d**) or changed (**c**)
- Structure: `diff <options> <filename1> <filename2>`
- For `<options>` see **man**
Outputs redirection and combining different commands
Commands outputs

• By default, the standard output of any command will appear to the terminal screen.
• Redirection of the output result to a file is possible.
• This is particularly useful for big files
• Syntax: command options filename.in > filename.out
Outputs redirection

- If the file exists, the result will be redirected to it.
- If the file does not exist, it will be automatically created and the result redirected to it.

```bash
$ cat ghandi.txt  
The difference between what we do and what we are capable of doing would suffice to solve most of the world's problems
$ cut -d' ' -f2,3 ghandi.txt  
difference between what we suffice to of the

$ cut -d' ' -f2,3 ghandi.txt > ghandi.txt.out
$ cat ghandi.txt.out  
difference between what we suffice to of the
```
Commands combination

- The **standard output** of any command will be **one unique output**
- As seen previously, this output can be **printed** in the screen or **redirected to a file**
- However, the **output** result of a command can also be **redirected to another command**
- This is particularly useful when several operations are needed for a file, with no need to store the intermediate outputs
Commands combination: example

• Combining several commands is done thanks to the use of a “|” character

• Structure:
  command1 options1 filename1.in |command2 options2 > filename.out

• This can be done for as many commands as needed