Linux Command Line Interface (CLI)

Why do people like Linux?

- **High security**
- **High stability**
- **Ease of maintenance**
- **Runs on any hardware**
- **Free**
- **Open source**
- **Ease of use**
- **Customizable**
- **Great for education**
- **Community**

You should know that...

Everything you do is a command under the hood.

Linux File Hierarchy Standard (FHS)

Directory	Description
/	*Primary hierarchy* root and root directory of the entire file system hierarchy.
/etc	Host-specific system-wide configuration files
/home	Users' home directories, containing saved files, personal settings, etc.
/root	Home directory for the root user.
/tmp	Temporary files (see also `/var/tmp`). Often not preserved between system reboots, and may be severely size restricted.
/var	Variable files—files whose content is expected to continually change during normal operation of the system such as logs.

Your first command will be:

who mum likes

A path.

A **path**, the general form of the name of a file or directory, specifies a unique location in a file system.

An **absolute** or **full** path points to the same location in a file system, regardless of the current working directory. To do that, it must include the **root** directory.

By contrast, a **relative** path starts from some given working directory, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory. If the working directory is not the file's parent directory, a file not found error will result if the file is addressed by its name.

What happens? Stop it!

Chill. Just type **pwd** in your terminal.

pwd - Present Working Directory. It’s exactly where you are in the filesystem.

The path is delimited by / which splits directories. Each right directory is inside the left one. For instance, /home/ion/pentagon means that home directory is inside / (root), ion directory is inside home and pentagon is inside ion.

Your everyday commands

pwd	Return present working directory
cd <path_to_dir>	Change directory
ls <?path> [args] (-lah)	List directory contents
sudo <command>	Execute a command as superuser
history	Return list of used commands
cat <path_to_file>	Print a file content.
ps [args] (-ax)	Process status
irb	Run Interactive Ruby shell
gedit, vim, nano, open	Open a text editor.

Your everyday commands

touch <path>	Create a file
mkdir [args] <path>	Create a directory
rmdir <path>	Remove an empty directory
cp [args] <path_from> <path_to>	Copy smth
mv [args] <path_from> <path_to>	Move (rename) smth
rm [args] <path>	Remove smth
echo <string>	Print smth into your console

Some practice would be nice, huh?

- Create a directory named lesson1
- Go to the just created directory
- Create lesson1/test directory
- Create file_to_copy file in lesson1/test
- Open lesson1/test/file_to_copy and write down some content
- Copy lesson1/test/file_to_copy file to lesson1/
- Create lesson1/dir_to_remove directory
- Create lesson1/dir_to_remove/test.txt file
- Move lesson1/dir_to_remove/test.txt into lesson1
- Remove lesson1/dir_to_remove directory

Check your history!

```
6054  mkdir lesson1
6055  cd lesson1
6056  mkdir test
6057  touch test/file_to_copy
6058  echo "some awesome content" > test/file_to_copy
6059  cp test/file_to_copy .
6060  mkdir dir_to_remove
6061  touch dir_to_remove/test.txt
6062  mv dir_to_remove/test.txt .
6063  rmdir dir_to_remove
```

a > operator redirects output to file, overwriting it

```
$ ls
file_to_copy test        test.txt
```

a dot (dot) symbolizes the present current directory

Dude, you've cheated! What are these symbols? Is it legit?

Some useful stuff you should know

>	Redirect an output to file overwriting it
>>	Redirect an output to file appending it
| (a.k.a pipe)	Remove an empty directory
~	Current user’ home directory
.	Current directory (or a hidden file)
..	Previous directory
/	Root dir or nesting separator

Some practice again

1. Create file in /tmp dir and fill it with any string without opening the file
2. Show the content without opening the file
3. Create a new file
4. Append content of the file 1 into the file 3

```
→  myprojects touch /tmp/testfile
→  myprojects echo 'some content' > /tmp/testfile
→  myprojects cat /tmp/testfile
some content
→  myprojects touch file2
→  myprojects cat /tmp/testfile >> file2
```

Permissions

$ cd /etc
$ touch girl

touch: girl: Permission denied


```
→  /etc ls -lah
t
```

Modification | Ownership | Properties

```
drwxr-xr-x  84 root  wheel  2.6K Feb  6 10:56 .
drwxr-xr-x   6 root  wheel  192B Feb  6 10:51 ..
-rw-r--r--   1 root  wheel  515B Oct 18 01:39 afpovertcp.cfg
lrwxr-xr-x   1 root  wheel   15B Nov 13 21:20 aliases -> postfix/aliases
-rw-r-----   1 root  wheel   16K Sep 19 03:40 aliases.db
drwxr-xr-x  10 root  wheel  320B Feb  6 10:52 apache2
```

Modify permissions

chmod <mod> <path>

for changing file modification

read = 4, write = 2, execute = 1

chown <user>:<group> <path>

for changing file owner(ship)

root is a superman who doesn't need any permissions to perform any actions.
That's why you have to think twice before doing commands under root.

Dive into CLI.

- Create a file
- Write a bash command in it
- Modify permissions to make it executable
- Run it using $ ./<filename> or $ bash ./filename

- Interactive shell for bash can be reached through $ bash -e

Homework

- Write a script which prompts `I'm annoying script` into the terminal every 5 minutes.
- Try to not use your mouse. It's only needed when you serf browser or draw. Every file system manipulation should be done from the terminal.
- Read about:
  - shell configuration files (.bashrc, .zshrc)
  - aliases
  - crontab
- Install oh-my-zsh.
- Configure your terminal in a way you like. I suggest yaquake for linux and iterm2 for macos.