Epitech training

Yoann Cerda

Who am I?

- Yoann Cerda
- Master degree from Epitech
- 2 years working experience in front-end development
- 3 years working experience in back-end development Python/Golang

Contact:

- tuxlinuxien@gmail.com
- Yoann-yongan (weixin)
- +86 13021259141

What's the goal?

This training is to prepare you dealing with Epitech working environment.

This training will cover:

- Linux/*NIX
- Shell
- C/C++
- Network

What's should I know?

As future Epitech students

- Search engines are your best friends
- RTFM (Read The Fucking Manual)
- A project finished at 99% is like a 0% one
- SegmentationFaults lead to 0
- Any allocated memory must be freed
- User inputs must be validated because users are dumb
- Copy/pasting code from internet leads to -42/20 score and maybe module failing.
- During team project, you need to understand all the codebase
- Projects should be finished on time
- No complains will be accepted if you didn’t have time to finish your project.
- Cheating is allowed but getting caught is NOT.

Questions?

Session #1
Linux

History

The history of Linux began in 1991 with the commencement of a personal project by Finnish student Linus Torvalds to create a new free operating system kernel. Since then, the resulting Linux kernel has been marked by constant growth throughout its history. Since the initial release of its source code in 1991, it has grown from a small number of C files under a license prohibiting commercial distribution to the 4.2.3 version in 2015 with more than 18 million lines.

History

- Linux is an implementation of UNIX.
- The Linux operating system is written in the C programming language.
- There's a Linux for everyone.
- Linux uses GNU tools, a set of freely available standard tools for handling the operating system.

How does it work?

- Kernel-land vs User-land
- Everything is a file
  - Directory, RAM, GPU are file you read from and write to
- Superuser “su” vs user
  - It’s like god mode, you can have access to all the system and break everything in the same time
- Graphical vs text mode
- I/O

Terminal

The Linux console is a system console internal to the Linux kernel (a system console is the device which receives all kernel messages and warnings and which allows logins in single user mode). The Linux console provides a way for the kernel and other processes to send text output to the user, and to receive text input from the user. The user typically enters text with a computer keyboard and reads the output text on a computer monitor. The Linux kernel supports virtual consoles - consoles that are logically separate, but which access the same physical keyboard and display. The Linux console (and Linux virtual consoles) are implemented by the VT subsystem of the Linux kernel, and do not rely on any user space software. This is in contrast to a terminal emulator, which is a user space process that emulates a terminal, and is typically used in a graphical display environment.

Terminal

- ls
  - Directory listing
- cd
  - Change directory
- pwd
  - Shows the current directory
- touch
  - Creates an empty file
- cat
  - Shows a file content into your terminal
- mkdir
  - Create a directory

Terminal

- rm
  - Remove file / directory
- cp
  - Copy file / directory
- sudo
  - Give your current user “root” permissions
- su
  - Change your current user to root during your session
- poweroff (root)
  - Shutdown your system
- reboot (root)
  - Reboot your system

Terminal

The terminal may look quite difficult to use at the beginning however, the more you will use it then the more you will see this tool is powerful and allows you to manage your system faster.

During system maintenance, most of the commands will require terminal usage since you will perform low level operations.

Command-lines will help you to understand how your system is working, fix it, optimise it based on your need and the most important, have a secure system.

Terminal

IMPORTANT

Never copy paste a command line without understanding it!

Shell

A Unix shell is a command-line interpreter or shell that provides a traditional Unix-like command line user interface. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute, or by creating text scripts of one or more such commands. Users typically interact with a Unix shell using a terminal emulator, however, direct operation via serial hardware connections, or networking session, are common for server systems. All Unix shells provide filename wildcarding, piping, here documents, command substitution, variables and control structures for condition-testing and iteration.

Shell

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a wrapper.

A shell script is usually have .sh file extension.

File system permission

Most file systems have methods to assign permissions or access rights to specific users and groups of users. These systems control the ability of the users to view, change, navigate, and execute the contents of the file system.

Example:

Run the following command: **ls -la ~/**

```
-rw-r--r--
```

- User can read/write
- Group can read
- Others can read

File system permission

Permission will allow you to isolate files or scripts on your system. If you create a shell script, you need to make sure it will have execution permission (at least for your user).

If you want to run you own script you need to specify it own path like

- ~/file.sh
- ./file.sh
- $PWD/file.sh
- /home/yoann/my/path/file.sh

Environment

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.

They are part of the environment in which a process runs. For example, a running process can query the value of the TEMP environment variable to discover a suitable location to store temporary files, or the HOME or USERPROFILE variable to find the directory structure owned by the user running the process.

Example:

```
$> export
```

Exercise

Write a shell script that creates a directory called “**ex1_dir**” and then create empty files called

a.txt, b.txt, c.txt, […], z.txt within this directory

Your shell script should be called, dumb_excercise_1.sh

Exercise

The first line of your script should start with the following **shebang** :

```
$> cat dumb_excercise_1.sh
#!/bin/sh

[... code ...]
$>
```

Also don’t forget to provide execution permission to your script.

Exercise

Output

$> ./dumb_excercise_1.sh
$> ls ./ex1_dir
a.txt b.txt c.txt d.txt e.txt […] z.txt

Exercise

You have 1 hour