Bash Lecture 1 - Basics

Material taken from

https://github.com/bertocco/bash_lectures

Why this lecture series

- ★To know UNIX/Linux command line
- ★To gain ability in command line usage
- ★To introduce scripting (in bash)
- ★To have some basic programming
- ★To introduce python programming
- ★To introduce useful python libraries

Bibliography and learning materials

★ Bibliography:

https://www.rigacci.org/docs/biblio/online/sysadmin/toc.htm

https://www.tldp.org/LDP/abs/html/

★ Learning Materials:

http://www.ee.surrey.ac.uk/Teaching/Unix/

https://github.com/gtaffoni/Learn-Python/blob/master/Lectures/ShellLecture01.pdf

https://github.com/gtaffoni/Learn-Python/blob/master/Lectures/ShellLecture02.pdf

https://github.com/bertocco/bash_lectures

Arguments of this lesson

- ★ How the shell works with you and linux
- ★ Features of a shell
- ★ Manipulating the shell environment

What is a shell?

★ SHELL is the human interface point for UNIX

SHELL is a program layer that provides an environment to enter commands and parameters to produce a given result.

To meet varying needs, UNIX has provided different shells. They differ in the various options they give the user to manipulate the commands and in the complexity and capabilities of the scripting language.

- Bourne (sh)
- Bourne Again (bash)
- Korn (ksh)
- C shells (csh)
- TC-Shel (tcsh)
- Z shell (zsh)

Why bash?

★Flexible

★ More friendly than others

★The default in the most part of linux distributions

UNIX commands (1)

- ★ General form of a command:

  command [flags] [argument1] [argument2] …

  Example:

  `ls -a -l` *or* `ls -al`

- ★ Arguments can be optional or mandatory

- ★ All commands have a return code (0 if OK)

  Read return code: `echo $?`

  The return codes can be used as part of control logic in shell scripts

- ★ All UNIX commands have an help:

  `man command` or `man <number> command

UNIX commands (2)

- ★ All commands:
  - – accept inputs from the standard input,

    is where UNIX gets the input for a command
  - – display output on standard output

    is where UNIX displays output from a command
  - – display error message on standard error

    is where UNIX displays any errors as a result of the execution of a command
- ★ UNIX has redirection capabilities: to redirect one or more of these (see advanced & scripting lesson)

Exercises

★ Verify that you are using bash:

`echo $SHELL`

★Explore help command

type `help`

★ Explore help for `ls` command

type `man command_name`

★List files `ls` or `ls -l` and check differences

★ List all files `ls -al`

★List files by date (direct and reverse order) `ls -trl`

Command `echo` and strings

- ★ When one or more strings are provided as arguments, echo by default repeats those strings on the screen.

  Example (try)

  ```
  echo This is a pen.
  ```

  It is not necessary to surround the strings with quotes, as it does not affect what is written on the screen. If quotes (either single or double) are used, they are not repeated on the screen (try `echo “This is a pen.”`).
- ★ `echo` can also show the value of a particular variable if the name of the variable is preceded directly (i.e., with no intervening spaces) by the dollar character ($), which tells the shell to substitute the value of the variable for its name. Example (try):

  ```
  x=5;   echo The number is $x.
  ```

Command `echo` examples

★ echo This is a pen.

★ x=5
```
echo The number is $x.
echo “The number is $x.”
```

★ Simple backup script
```
OF=/home/me/my-backup-$(date +%Y%m%d).tgz
tar -czf $OF <path>/dir_or_file_to_tar
ls -l     (to check the result)
```

`export`

`export` exports environment variables (also to children of the current process). Example:

```
ubuntu~$ export a=test_env
ubuntu:~$ echo $a
test_env
ubuntu:~$ /bin/bash
ubuntu:~$ echo $a
test_env
ubuntu:~$ exit
exit
ubuntu:~$ echo $a
test_env
```

The *${SHLVL}* variable is an integer that tracks how many nested shells (subshells) you currently have open in your terminal session.

`export` called with no arguments prints all of the variables in the shell's environment.

`unset` frees variables

User related commands: passwd

- ★`passwd` changes user’s password

  Example: type `passwd`

```
$ passwd
Changing password for bertocco.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Sorry, passwords do not match
passwd: Authentication token manipulation error
passwd: password unchanged
$ passwd
Changing password for bertocco.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

User related commands: who and whoami

★`who` show who is logged on

Print information about users who are currently logged in.

★`whoami`

Print the user name associated with the current effective user ID.

Exercise:

try the commands and then type `man who`

and try some option

File manipulation commands (1)

It exists a set of file manipulation commands to manage files and directories.

To use these commands, the user needs to have right on the file to manage.

```
drwxrwxr-x   2 bertocco bertocco     20480 Dec 14 09:00 BACKUP
```

owner *group* *size* *last_access_date* *file-name*

drwxrwxr-x permissions representation:

d means it is a directory (- for a file)

rwx means readable, writable, executable by owner

rwx readable, writable, executable by group

r-x readable, NOT writable, executable by others

File Permissions

Understand the meaning of:

```
drwxrwxr-x   2 bertocco bertocco      4096 Apr 26  2018 config
-rw-rw-r--   1 bertocco bertocco     10240 Mar 13  2017 config.tar
-rw-------   1 bertocco bertocco 960065536 Dec  3 22:02 core.3040
-rw-rw-r--   1 bertocco bertocco   7290880 May  8  2017 demo_EGIconf.tar
drwxr-xr-x.  4 bertocco bertocco      4096 Dec  7 15:57 Desktop
drwx------. 12 bertocco bertocco      4096 Aug 13 19:01 dev
drwxr-xr-x. 14 bertocco bertocco      4096 Nov 28 17:18 Documents
drwxr-----. 13 bertocco bertocco      8192 Dec 10 12:35 Downloads
drwxrwxr-x   2 bertocco bertocco       147 Apr 24  2018 exchange
-rw-r--r--   1 bertocco bertocco       181 Apr 13  2017 filmatini_utili.txt
```

Change File Permissions

Change read permission (similarly for write ‘w’ and execute ‘x’):

```
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod -r pippo     # remove all read permissions. Check:
$ ls -l pippo
--w--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod +r pippo    # add all read permissions, Check:
$ ls -l pippo
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod -r pippo     # remove a new time all permissions, to restart from
--w--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod u+r pippo     # add read permission to user
$ ls -l pippo
-rw--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod g+r pippo     # add read permission to group
$ ls -l pippo
-rw-rw---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod a+r pippo     # add read permission to all
$ ls -l pippo
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
```

Main file manipulation commands

- \`touch\` creates a file
- \`mkdir  mydir\` creates a directory (where you are)
- \`mkdir -p /*onedir/*twodir/threedir\`
- \`rmdir mydir\` delete an empty directory
- \`rm -rf\` force to recursively delete a non empty directory
- \`cp file1 file2\` copy file1 on file2 (overwriting it if already exists,
  creating file2 if it does not exist)
- \`cp -i file1 file2\` before copy asks “are you sure?”
- \`cp file1 file2\` remove the -i flag if set
- \`rm file1\` removes file1
- \`mv file1 file2\` moves file1 on file2 # it is the same of :
- \`cp file1 file2\`; \`rm file1\`

File manipulation commands: Exercises

- Create a file
- Create a directory
- Create a directory tree
- Create files in the directory tree
- Remove a file
- remove a directory (empty and not empty)
- remove a directory tree (empty and not empty)
- Rename a file