Bash Lecture 2 – Advanced Bash Commands

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/bertocco/abilita_info_units_2021

Arguments of this lesson

- ★Bash configuration and user's environment manipulation
- ★Locating commands (which)
- ★File information commands (find, file)
- ★UNIX processes
- ★Process related commands (kill, ps, wait)

Bash configuration

Bash has more configuration startup files.
They are executed at bash start-up time.
The files and sequence of the files executed differ from the type of shell. Shell can be:

- ★Interactive
- ★Non-interactive
- ★Login shell
- ★Non-login shell

Bash types

★Interactive: means that the commands are run with user-interaction from keyboard. E.g. the shell can prompt the user to enter input.

★Non-interactive: the shell is probably run from an automated process. Typically input from standard input and output to log file.

★Login: shell is run as part of the login of the user to the system.

★Non-login: any shell run by the user after logging on, or run by any automated process not coupled to a logged in user.

Bash startup files (1)

http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

★Interactive login shell, or with –login:

/etc/profile, if that file exists

~/.bash_profile,

~/.bash_login

~/.profile, in that order, and reads and executes

--noprofile option may be used to inhibit this behavior.

When an interactive login shell exits, or a non-interactive login shell executes the exit builtin command, Bash reads and executes commands from the file ~/.bash_logout, if it exists.

Bash startup files (2)

http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

★interactive non-login shell

[Example: when you open a new terminal window by pressing Ctrl+Alt+T, or just open a new terminal tab.]

bash reads and executes commands from ~/.bashrc, if that file exists.

--norc option ti inhibit this behaviour.

--rcfile file option will force Bash to read and execute commands from file instead of ~/.bashrc.

So, typically, your ~/.bash_profile contains the line

if [-f ~/.bashrc]; then . ~/.bashrc; fi

after (or before) any login-specific initializations.

Bash startup files (3)

http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

★Invoked non-interactively

[Example: to run a shell script]

bash looks for the variable BASH_ENV in the environment, expands its value if it appears there, and uses the expanded value as the name of a file to read and execute. Bash behaves as if the following command were executed:

if [-n "$BASH_ENV"]; then . "$BASH_ENV"; fi

but the value of the PATH variable is not used to search for the filename.

If a non-interactive shell is invoked with the --login option, Bash attempts to read and execute commands from the login shell startup files.

Set user's PATH environment variable

```
$ cat .profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or
# ~/.bash_login exists.

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
    fi
fi
# set PATH so it includes user's private bin directories
PATH="$HOME/bin:$HOME/.local/bin:$PATH"
```

~/.bashrc file

User specific, hidden by default.

~/.bashrc

If not there simply create one.

System wide:

/etc/bash.bashrc

`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

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

`unset` frees variables

Shell alias

A shell alias is a shortcut to reference a command.
It can be used to avoid typing long commands or as a means to correct incorrect input.

Example: it is used to set default options on commands

alias ls=`ls -l`
alias rm=`rm -i`

Exercises:

1) try to define and use the previous aliases
2) Define the aliases in the ~/.bashrc, open a new terminal and verify the aliases running them

Locating commands

★To execute a command, UNIX has to locate the command before it can execute it

★UNIX uses the concept of search path to locate the commands.

★Search path is a list of directories in the order to be searched for locating commands. Usually it contains standard paths (/bin, /*usr*/bin, …)

★Modify the search path for your environment modifying the PATH environment variable

`which`

★`which` can be used to find whether a particular command exists in you search path. If it does exist, which tells you which directory contains that command.

Examples (try with existing and not existing commands):

```
which  pippo
which gedit
which vim
```

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 information commands

★Each file and directory in UNIX has several attributes associated with it. UNIX provides several commands to inquire about and process these attributes

`find`

★`find` searches for the particular file giving the flexibility to search for a file by various attributes: name, size, permission, and so on.

Command general form:

find directory-name search-expression

`find` Examples (try)

```
find . -name pippo
find /etc -name networking
find /etc -name netw    # nothing found
find /etc -name netw\*

find -size 18        # 18 blocks files
find -size 1024c   # 1024 bytes

find . -print
```

Read the manual and try other options

Try, if possible, a find case insensitive

`file`

★`file` can be used to determine the type of the specified file.

Examples (try):

$ file /etc/networking/interfaces

/etc/networking/interfaces: cannot open `/etc/networking/interfaces' (No such file or directory)

$ file /etc/network/interfaces

/etc/network/interfaces: ASCII text

UNIX Processes

Usually, a command or a script that you can execute consists of one or more processes.

The processes can be categorized into the following broad groups:

- ★ Interactive processes, which are those executed at the terminal. Can execute either in foreground or in background. In a foreground process, the input is accepted from standard input, output is displayed to standard output, and error messages to standard error. In background, the terminal is detached from the process so that it can be used for executing other commands. It is possible to move a process from foreground to background and vice versa (<ctrl+bg>; <ctrl+fg>.
- ★ Batch processes are not submitted from terminals. They are submitted to job queues to be executed sequentially.
- ★ Deamons are never-ending processes that wait to service requests from other processes.

Process Related Commands

★a command or a script that you can execute consists of one or more processes.

The main are:

- `ps`
- `kill`
- `wait`
- `nohup`
- `sleep`

`ps`

★`ps` command is used to find out which processes are currently running.

Exercises:

- Try the following commands, check the differences in the output. Read the flag meaning using `man ps`:

  ps

  ps -ef

  ps -aux

`kill`

★`kill` is used to send signals to an executing process. The process must be a nonforeground process for you to be able to send a signal to it using this command.

★The default action of the command is to terminate the process by sending it a signal. If the process has been programmed for receiving such a signal. In such a case, the process will process the signal as programmed.

★ You can kill only the processes initiated by you. However, the root user can kill any process in the system.

★The flags associated with the kill commands are as follows:

-l to obtain a list of all the signal numbers and their names that are supported by the system.

-'signal number' is the signal number to be sent to the process. You can also use a signal name in place of the number. The strongest signal you can send to a process is 9 or kill.

`kill` Exercises

- ★Look for a process PID of a process belonging of you (using ps) and kill it using two different signals: -9 and -15.
- ★List all available signals and red the differences between the two signal previously used

`wait` with exercises

★`wait` is tp wait for completion of jobs. It takes one or more process IDs as arguments. This is useful while doing shell programming when you want a process to be finished before the next process is invoked.
If you do not specify a process ID, UNIX will find out all the processes running for the current environment and wait for termination of all of them.

★Examples:

- `wait` If you want to find out whether all the processes you have started have completed
- `wait 15060` If you want to find out whether the process ID 15060 has completed

★The return code from the wait command is zero if you invoked the wait command without any arguments. If you invoked the wait command with multiple process IDs, the return code depends on the return code from the last process ID specified.

`wait`: exercise

★From a shell launch an infinite process using:
`while true; do echo looping; sleep 2; done`

★From another shell find the pid of this process using `ps` command

★From a third shell launch a process waiting for the end of the initial infinite loop
pid=<your_process_pid>; wait $pid

★From a fourth shell kill the first process (pid)

★ Check in the third shell that your waiting process ended

NOT WORKING using shells. Needs scripting: next time.