Bash Lecture 2 – Advanced Bash

Arguments of this lesson

- ★Bash configuration and user’s environment manipulation (export, alias)
- ★Locating commands (which)
- ★File information commands (find, file)
- ★UNIX processes
- ★Process related commands (kill, ps, wait, nohup, sleep)
- ★File content related commands (more, less, tail, read, tee, wc)
- ★Redirection
- ★File content search commands (grep)
- ★Status commands (date)
- ★Unix wildcards (* ? [ ])

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 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.

Complete description at:

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

Main user bash configuration file is ~/.bashrc

If not there simply create one.

Main system configuration file is /etc/bash.bashrc

Set user’s PATH environment variable

In

"$HOME/.bashrc"

```
# set PATH so it includes user's private bin directories
PATH="$HOME/bin:$HOME/.local/bin:$PATH"
```

`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 in

  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

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
- \`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

`nohup`

★When you are executing processes under UNIX, they can be running in foreground or background. In a foreground process, you are waiting at the terminal for the process to finish. Under such circumstances, you cannot use the terminal until the process is finished. You can put the foreground process into background as follows:

```
ctrl-z

bg
```

The processes in UNIX will be terminated when you logout of the system or exit the current shell whether they are running in foreground or background. The only way to ensure that the process currently running is not terminated when you exit is to use the nohup command.

`nohup`

The nohup command has default redirection for the standard output. It redirects the messages to a file called nohup.out under the directory from which the command was executed. That is, if you want to execute a script called sample_script in background from the current directory, use the following command:

```
nohup sample_script &
```

The & (ampersand) tells UNIX to execute the command in background. If you omit the &, the command is executed in foreground. In this case, all the messages will be redirected to nohup.out under the current directory. If the nohup.out file already exists, the output will be appended to it.

`nohup`: Examples

```
nohup grep sample_string * &

nohup grep sample_string * > mygrep.out &

nohup my_script > my_script.out &
```

`sleep`

`sleep` wait for a certain period of time between execution of commands. This can be used in cases where you want to check for, say, the presence of a file, every 15 minutes. The argument is specified in seconds.

Examples: If you want to wait for 5 minutes between commands, use:

```
sleep 300
```

Small shell script that reminds you twice to go home, with a 5-minute wait between reminders:

```
echo "Time to go home"
sleep 300
echo "Final call to go home ....."
```

File Content Related Commands

★Commands that can be used to look at the contents of the file or parts of it. You can use these commands to look at the top or bottom of a file, search for strings in the file, and so on.

\`more\`

- \`more\` can be used to display the contents of a file one screen at a time. By default, the more command displays one screen worth of data at a time. The more command pauses at the end of display of each page. To continue, press a space bar so that the next page is displayed or press the Return or Enter key to display the next line. Mostly the more command is used where output from other commands are piped into the more command for display.
- Try

\`less\`

- ★\`less\` is to view the contents of a file. This may not be available by default on all UNIX systems. It behaves similarly to the more command. The less command allows you to go backward as well as forward in the file by default.
- ★Try
- ★Cat <a big file> | less

`tail`

★`tail` to display, on standard output, a file starting from a specified point from the start or bottom of the file. Whether it starts from the top of the file or end of the file depends on the parameter and flags used. One of the flags, -f, can be used to look at the bottom of a file continuously as it grows in size. By default, tail displays the last 10 lines of the file.

`tail` exercises

```
tail -f500 /var/log/syslog
```

list of flags that can be used with the tail command:

-c number to start from the specified character position number.

-b number to start from the specified 512-byte block position number.

-k number to start from the specified 1024-byte block position number.

-n number to start display of the file in the specified line number.

-r number to display lines from the file in reverse order.

-f to display the end of the file continuously as it grows in size.

`read`

`read` is used in shell scripts to read each field from a file and assign them to shell variables.

A field is a string of bytes that are separated by a space or newline character. If the number of fields read is less than the number of variables specified, the rest of the fields are unassigned.

Flag -r to treat a \(backslash) as part of the input record and not as a control character.

`read` Examples

Example following is a piece of shell script code that reads first name and last name from namefile and prints them:

- create the file

```
cat <<EOF > names_list.txt
Sara Bertocco
Mario Rossi
John Doe
EOF
```

- Read the file by line and print on standard output

```
while read -r lname fname
do
	echo $lname","$fname
done < names_list.txt
```

`read` Examples

Example following is a piece of shell script code that reads a file by line:

```
while read -r line
do
    printf 'Line: %s\n' "$line"
done < names_list.txt
```

The file name can be indicate also with full path name.

`tee`

`tee` to execute a command and want its output redirected to multiple files in addition to the standard output, use the tee command. The tee command accepts input from the standard input, so it is possible to pipe another command to the tee command.

The default of the tee command is to overwrite the specified file.

-a is an optional flag to append to the end of the specified file

`tee` Examples (try)

- use the cat command on file1 to display on the screen and make a copy of file1 on file2, use the tee command as follows:

```
cat file1 | tee file2 | more
```

- make the same but appending file1 to the end of an already existing file2 using the flag -a :

```
cat file1 | tee -a file2 | more
```

`wc`

`wc` counts the number of bytes, words, and lines in specified files. A word is a number of characters stringed together delimited either by a space or a newline character.

Following is a list of flags that can be used with the wc command:

-l to count only the number of lines in the file.
-w to count only the number of words in the file.
-c to count only the number of bytes in the file.

You can use multiple filenames as argument to the wc command.

`wc` exercices

```
wc file

wc -w file

cat <file> | wc -l

wc -w <file1> <file2>
```

Special characters: Pipe

Pipe [|]. Passes the output (stdout) of a previous command to the input (stdin) of the next one, or to the shell. This is a method of chaining commands together.

```
echo ls -l | sh
#  Passes the output of "echo ls -l" to the shell,
#+ with the same result as a simple "ls -l".
```

```
cat *.lst | sort | uniq
# Merges and sorts all ".lst" files, then deletes duplicate lines.
```

A pipe sends the stdout of one process to the stdin of another. In a typical case, a command, such as cat or echo, pipes a stream of data to a command that transforms it in input for processing:

```
cat $filename1 $filename2 | grep $search_word
```

Redirection with pipe and tee examples

Examples of redirection of the output of a command to be used as input of another:

- Display the output of a command (in this case ls) by pages:
  ```
  ls -la | less
  ```
- Count files in a directory:
  ```
  ls -l | wc -l
  ```
- Count the number of rows containing of the word “canadesi” in the file vialactea.txt
  ```
  grep canadesi vialactea.txt | wc -l
  ```
- Count the number of words in the rows containing the word “canadesi”

`tee` is useful to redirect output both to stdout and to a file. Example:

```
find . -name filename.ext 2>&1 | tee -a log.txt
```

This will take stdout and append it to log file. The stderr will then get converted to stdout which is piped to tee which appends it to the log and sends it to stdout which will either appear on the tty or can be piped to another command.

To go deep: https://stackoverflow.com/questions/2871233/write-stdout-stderr-to-a-logfile-also-write-stderr-to-screen

Exercise: redirection

Create a directory and file tree like this one:

```
my_examples  /ex1.dir
             /ex2.txt
             /ex3.dir
             /ex3.dir/file1.txt
             /ex3.dir/file2.txt
             /ex3.dir/file3.txt
```

Remove read permissions to directory *ex2.dir*

Redirect output on a file. Error is displayed on terminal

Redirect error on a file. Output is displayed on terminal

Verify the content of the files

Stderr redirected to file

Redirect output and errors symultaneously

Use pipe to redirect the output of a command to another command and to a file

Use tee to redirect output both to stdout and to a file

File Content Search Commands

For searching for a pattern in one or more files, use the grep series of commands. The grep commands search for a string in the specified files and display the output on standard output.

`grep`

`grep` extended version of grep command. This command searches for a specified pattern in one or more files and displays the output to standard output. The pattern can be a regular expression to match any single character.

***** to match one or more single characters that precede the asterisk.

^ to match the regular expression at the beginning of a line.

$ to match the regular expression at the end of a line.

+ to match one or more occurrences of a preceding regular expression.

? to match zero or more occurrences of a preceding regular expression.

[ ] to match any of the characters specified within the brackets.

`grep` Examples

Let us assume that we have a file called file1 whose contents are shown below using the more command:

```
more file1
*****  This file is a dummy file *****
which has been created
to run a test for egrep
grep series of commands are used by the following types of
people
   programmers
   end users
Believe it or not, grep series of commands are used by pros and
novices alike
*****  THIS FILE IS A DUMMY FILE *****
```

`grep` Examples

- If you are just interested in finding the number of lines in which the specified pattern occurs, use the -c flag as in the following command:

```
grep -i -c dummy file1
```

- If you want to get a list of all lines that do not contain the specified pattern, use the -v flag as in the following command:

```
grep -i -v dummy file1
```

- If you are interested in searching for a pattern that you want to search as a word, use the -w flag as in the following command:

```
grep -w grep file1
```

`date`

`date` command to display the current date and time in a specified format. If you are root user, use the date command to set the system date.
To display the date and time, you must specify a + (plus) sign followed by the format. The format can be as follows:
%A to display date complete with weekday name.
%b or %h to display short month name.
%B to display complete month name.
%c to display default date and time representation.
%d to display the day of the month as a number from 1 through 31.
%D to display the date in mm/dd/yy format.
%H to display the hour as a number from 00 through 23.
%I to display the hour as a number from 00 through 12.
%j to display the day of year as a number from 1 through 366.
%m to display the month as a number from 1 through 12.
%M to display the minutes as a number from 0 through 59.
%p to display AM or PM appropriately.
%r to display 12-hour clock time (01-12) using the AM-PM notation.
%S to display the seconds as a number from 0 through 59.

`date`

Other format flags:

%T to display the time in hh:mm:ss format for 24 hour clock.

%U to display the week number of the year as a number from 1 through 53 counting Sunday as first day of the week.

%w to display the day of the week as a number from 0 through 6 with Sunday counted as 0.

%W to display the week number of the year as a number from 1 through 53 counting Monday as first day of the week.

%x to display the default date format.

%X to display the time format.

%y to display the last two digits of the year from 00 through 99.

%Y to display the year with century as a decimal number.

%Z to display the time-zone name, if available.

`date`: Exercises

Try some example of `date` command usage with different display of day, month, year

- ★If you want to display the date without formatting, use date without any formatting descriptor as follows:

```
date
Sat Dec  7 11:50:59 EST 1996
```

- ★If you want to display only the date in mm/dd/yy format, use the following commands:

```
date +%m/%d/%y
12/07/96
```

- ★If you want to format the date in yy/mm/dd format and time in hh:mm:ss format, use the following command:

```
date "+%y/%m/%d %H:%M:%S"
96/12/07 11:57:27
```

- ★Following is another way of formatting the date:

```
date +%A","%B" "%d","%Y
Sunday,December 15,1996
```

Linux wildcards

There are three main wildcards in Linux:

Asterisk (*) – matches one or more occurrences of any character, including no character.

Question mark (?) – represents or matches a single occurrence of any character.

Bracketed characters ([ ]) – matches any occurrence of character enclosed in the square brackets.