

# Linux shell / shell scripting


## Basic level

Magali COTTEVIEILLE - September 2009

## Last time we saw...

- ▶ **Directories:** `pwd`, `cd`, `rm -r`, `mkdir`, `du -sh`
- ▶ **Rights:** `ls -l`, `chown` and `chmod`
- ▶ **Jobs:** `top`, `ps x`, `kill -9`
- ▶ **The manual pages:** `man`

# Linux directories

	Directory	Use to store this kind of files
		
<b>bin</b>	<b>/bin</b>	All your essential program (executable files)
<b>sbin</b>	<b>/sbin</b>	All super user executable (binaries) files
<b>home</b>	<b>/home</b>	Your sweet home.
<b>etc</b>	<b>/etc</b>	Configuration file of your Linux system are here.
<b>lost+found</b>	<b>/lost+found</b>	When your system crashes, this is the place where you will get your files.
<b>var</b>	<b>/var</b>	Mostly the file(s) stored in this directory changes their size i.e. variable data files. Data files including spool directories and files, administrative and logging data, and transient and temporary files etc.
<b>usr</b>		Central location for all your application program, x-windows, man pages, documents etc are here. Programs meant to be used and shared by all of the users on the system. Some of the important sub-directories are as follows:
<b>lib</b>		<b>bin</b> - sub-directory contains executables
<b>tmp</b>		<b>sbin</b> - sub-directory contains files for system administration i.e. binaries
<b>dev</b>	<b>/usr</b>	<b>include</b> - sub-contains C header files
		<b>share</b> - sub-contains contains files that aren't architecture-specific like documents, wallpaper etc.
		<b>X11R6</b> - sub-contains X Window System
	<b>/lib</b>	All shared library files are stored here. This libraries are needed to execute the executable files (binary) files in /bin or /sbin.
	<b>/tmp</b>	Temporary file location
	<b>/dev</b>	Special Device files are store here. For e.g. Keyboard, mouse, console, hard-disk, cdrom etc device files are here.
	<b>/mnt</b>	Floppy disk, CD-Rom disk, Other MS-DOS/Windows partition mounted in this location (mount points) i.e. temporarily mounted file systems are here.
	<b>/boot</b>	Linux Kernel directory
	<b>/opt</b>	Use to store large software applications like star office.
	<b>/proc</b>	This directory contains special files which interact with kernel.

# Secure connections and transfers

## ▶ Connections:

- ▶ `ssh 156.111.6.184`  
`ssh -X magali@156.111.6.184`

## ▶ transfers:

- ▶ `sftp magali@156.111.6.184, then put/get`
- ▶ `scp (-r) files machine:path`  
`scp *.pam 156.111.6.184:/usr10/magali`

## ▶ Example: connect and transfer files on the master node of the cluster

- ▶ Connect with `ssh -X magali@156.111.6.1977`
- ▶ Check the disk space:  
`/home/magali> df -h ./`

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sdc1	2.2T	1.3T	985G	56%	/master.raid
- ▶ Check the space usage of your files:  
`magali/Master_cluster> du -hs ctftilt`  
`262M ctftilt`
- ▶ Transfer files from the cluster to your computer:  
`scp -r Xlk 156.111.6.46:/space`
- ▶ Note: to log on a node from the master node:  
`% rsh node42`

# Miscellaneous

- ▶ `whoami` --- returns your username. Sounds useless, but isn't. You may need to find out who it is who forgot to log out somewhere
- ▶ `date` --- shows the current date and time.
- ▶ **Archiving: tar**
  - ▶ `tar -cvf myarchive.tar *.pam`
  - ▶ `tar -cvzf myarchive.tar.gz *.pam`
  - ▶ `tar -xvf myarchive.tar`
  - ▶ `tar -xvzf myarchive.tar.gz`
- ▶ **File Compression:**
  - ▶ `gzip filename` --- Produces `filename.gz`
  - ▶ `gunzip filename` --- uncompresses files compressed by `gzip`.
  - ▶ `compress/uncompress`: `.Z` format
  - ▶ `zip/unzip`: `.zip` format
- ▶ **Redirection: > and |**  
`cat file1.txt file2.txt > file3.txt`  
`ps -ef | more`

# What is a C shell script ?

- ▶ Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use the commands one by one, you can store this sequence of commands in a text file and tell the shell to execute this text file instead of entering the commands.
- ▶ In addition, most UNIX shells have the capability of command flow logic ( goto, if/then/else/endif etc.), retrieving the command line parameters, defining and using (shell) variables etc.
- ▶ Useful to create our own commands, save lots of time, automate some tasks

# Getting started with Shell Programming

- ▶ You need your favorite text editor: nedit, emacs, vi, ...
- ▶ After writing shell script set execute permission for your script as follows  
chmod permissions your-script-name  
\$ chmod +x your-script-name  
\$ chmod 755 your-script-name
- ▶ Execute your script:  
./your-script-name  
In the last syntax ./ means current directory
- ▶ Example  
\$ vi first.csh  
#!/bin/csh  
# Comment: My first shell script  
clear  
echo "Knowledge is Power"
- ▶ After saving the above script, change the rights and launch it:  
./first.csh
- ▶ Tip: try to give file extension such as .csh or .sh, which can be easily identified by you as shell script.
- ▶ Note, the examples will be specific of the csh. These might not work with bash.

# Variables in Shell

- ▶ In Linux (Shell), there are two types of variable:
- ▶ System variables - Created and maintained by Linux itself.
- ▶ User defined variables - Created and maintained by user.
- ▶ Examples of system variables:
  - ▶ SHELL
  - ▶ PATH
  - ▶ OSTYPE
- ▶ Your start file (`.cshrc`) is a shell script, automatically sourced each time you open a terminal window. If you make a modification to your `.cshrc`, to see the effects, either you must source it, either open a new terminal.
- ▶ If you want to know the value of a variable:

```
$ echo $USERNAME  
$ echo $HOME
```



# How to define User defined variables (UDV)

- ▶ To define UDV use following syntax (csh & tcsh):

```
set variable = value, ex:  
set start = 1  
set coos = "../coords/sndc"
```

- ▶ Variables are case-sensitive, just like filenames in Linux
- ▶ Do not use `?`, `*` *etc.* to name your variable names (they are “wildcards”)
- ▶ To print or access UDV use following syntax :
  - ▶ `$variablename` is the content of `variablename`
  - ▶ `echo $variablename` to print the content of `variablename`
- ▶ Shell Arithmetic: `expr op1 math-operator op2`, ex:

```
% expr 1 + 3  
4  
% set sum = `expr 1 + 3`  
% echo $sum  
4
```

# Wild cards \* ? [...]

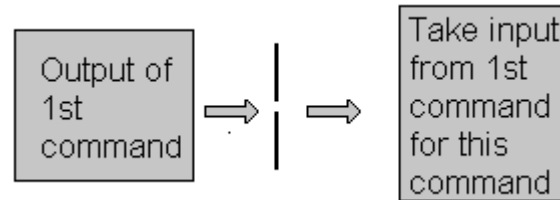
*	Matches any string or group of characters.	\$ ls * \$ ls a* \$ ls *.c \$ ls ut*.c
?	Matches any single character.	\$ ls fo?
[...] [...-...]	Matches any one of the enclosed characters A pair of characters separated by a minus sign denotes a range.	\$ ls [abc]*

# Redirection of Standard output/input i.e. Input - Output redirection

- ▶ Mostly all command gives output on screen or take input from keyboard, but it's possible to send output to file or to read input from file.
  - ▶ `% ls` command gives output to screen; to send output to file of `ls` command give command
  - ▶ `% ls > filename` -- It means put output of `ls` command to filename.
- ▶ There are 4 main redirection symbols `>`, `>>`, `<`, `<<`
  - ▶ (1) `>` Redirector Symbol
  - ▶ (2) `>>` Redirector Symbol (append to the end)
  - ▶ (3) `<` Redirector Symbol: To take input to Linux-command from file instead of key-board. For e.g. To take input for `cat` command give  
`% cat < myfiles`
  - ▶ (4) `<<` special form of redirection is used in shell scripts. Ex:  
`ctftilt.exe << eof`  
... (options)  
`eof`  
the input is taken from the current file (usually the shell script file) until the string following the "`<<`" is found.

# Pipes

- ▶ A pipe is a way to connect the output of one program to the input of another program without any temporary file.



- ▶ Syntax:

`command1 | command2`

- ▶ Examples:

- ▶ `ls | more`

- ▶ `ps -ef | grep spider`

# Control structures (if)

- ▶ The C shell has control structures similar to the C programming language. These are foreach, if, switch and while. These are usually used in shell scripts
- ▶ `if (condition) simple command`
- ▶ `if (condition)`  
`then`  
`commands`  
`...`  
`fi`
- ▶ `if (condition)`  
`then`  
`commands`  
`...`  
`else`  
`commands`  
`...`  
`fi`
- ▶ You can have nested if-else-fi

# Loops

▶ while:

```
while (expr)
  command(s)
end
```

▶ foreach:

```
foreach var (wordlist)
  command(s)
end
```

▶ label:

```
label
...
goto label
```

# Interfacing SPIDER and C shell

## Command VM in SPIDER

### Ex:

```
; ----- Input files -----
FR L
[selection_file]../2-selection_on_visual      ; Selection doc file

FR L
[MRC_mic]../Micrographs_MRC/Xlk{*****[mic]} ; Input MRC images

; ----- END BATCH HEADER -----
MD
set MP
0
MD
VB OFF
MD
RESULTS OFF
; ----- start sbgrid if not already done:
; VM
; source /programs/labcsshr

UD N [max]
[selection_file]

DO [i]=4041,[max]
UD [i],[mic]
[selection_file]

; ctftilt options:
; Input image file name
; Output diagnostic file name
; CS[mm], HT[kV], AmpCnst, XMAG, DStep[um], PAve:2.26,300.0,0.1,100000.0,15.0,2
; Positive defocus values for underfocus
; Box, ResMin[A], ResMax[A], dFMin[A], dFMax[A], FStep: 128,200.0,6.0,9000.0,50000.0,500.0

VM
rm mic.pow

; ----- Now call ctftilt
VM
./call_ctftilt.sh [MRC_mic].mrc >> ctftilt.log

ENDDO

EN D
```