

Mills HPC Tutorial Series

# Linux Basics II

# Objectives

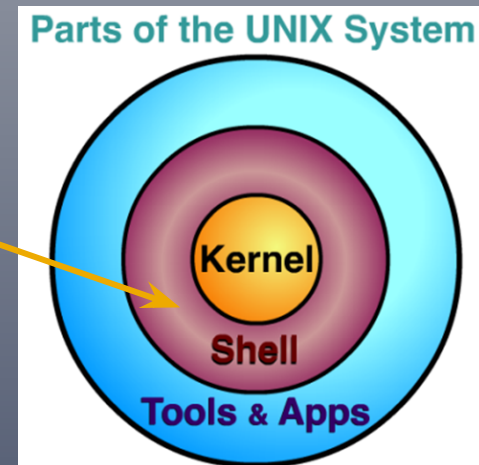
- Bash Shell
- Script Basics
- Script Project
  - This project is based on using the Gnuplot program which reads a command file, a data file and writes an image file as an x-y plot. Firefox will be used to view the image.
- Python by Example

# Bash Shell

---

# Shell Basics

- The shell is a command interpreter. We are using the bash shell (/bin/bash).
- It is the insulating layer between the operating system kernel and the user.
- It is also a powerful programming language.
- A shell program is called a script.



# Script Basics

---

# What is a script?

- Nothing more than a list of system commands stored in a file.
- More than just saving time for repetitive tasks.
- Can be modified and customized for particular applications.
- Documents workflow for projects.

# Get Exercises (Mills account)

1. If you have an account on the Mills cluster, use SSH to connect

```
ssh -Y username@mills.hpc.udel.edu
```

2. Copy the exercise directory `mlbII` into your home directory and change to it.

```
cp -r ~trainf/mlbII $HOME  
cd ~/mlbII
```

# Get Exercises (wget)

1. If you do not have an account on the Mills cluster, then download the exercise file `mlbII.tar.gz` using `wget` into your home directory.\*

```
cd $HOME
wget http://www.udel.edu/it/research/files/cluster/workshop/mlbII.tar.gz
```

2. Untar and uncompress the exercise file to create the `mlbII` directory and change to it.

```
tar -zxvf mlbII.tar.gz
cd mlbII
```

\* Note `wget` is available on most Gnu/Linux distributions.



# Script Basics: source

- hello1

```
$ more hello1
    ... Display contents of hello1 file ...
$ source hello1
Hello,
$ myvar=World
$ source hello1
Hello, World
$
```

# Script Basics: sha-bang & export

- hello2

```
$ more hello2
    ... Display contents of hello2 file ...
$ ./hello2
-bash: ./hello2: Permission denied
$ ls -l hello2
-rw-r--r-- 1 trainf everyone 46 Jun 20 14:10 hello2
$ chmod u+x hello2
$ ./hello2
Hello,
$ export myvar
$ ./hello2
Hello, World
$
```

# Script Basics: Special Characters

- `#` comment except `#!` (sha-bang)
- `' '` suppress all meaning (single quotes)
- `" "` suppress all meaning except `$`, `\`, ``` (double quotes)
- `` `` value of string is output of the command (back quotes)
- `\` to get a literal special character - escape (backslash)
- `;` command separator
- spaces are important

# Script Basics: Special Characters

- hello3

```
$ more hello3
    ... Display contents of hello3 file ...
$ ./hello3
It's "Hello, World" from the variable $myvar on: Thu Jun 21 12:31:08 EDT 2012
$
```

# Script Project

---

# Script Project

**Part 1:** Build a Gnuplot command file (STDOUT).

**Part 2:** Read a data file (STDIN) and create a new data file suitable for Gnuplot using an x, y pair on each line (STDOUT) with error checking (STDERR).

**Part 3:** Execute the gnuplot command with the command file as the argument.

# What is Gnuplot?

- A portable command-line driven graphing utility available on Linux and many other platforms
- Supports many different types of 2D and 3D plots
- Supports many different types of output files such as svg, png, etc.
- See <http://www.gnuplot.info/> for more information

# Script Project

```
$ cd $HOME  
$ mkdir project-bash  
$ cd project-bash
```



Part 1

# Script Project

echo, source, if - then, case, function

---

# Part 1: echo

Display message on screen.

```
echo [options]... [string]...
```

**-n** Do not output the trailing newline.

# Part 1: Testing echo

```
$ cp ~/mlbII/echo2 .
$ more echo2
    ... Display contents of echo2 file ...
$ ./echo2 >commands
$ wc -l commands
3 commands
$ more commands
    ... Display contents of commands file ...
$
```

# Part 1: source & if – then

Run commands from a file.

```
source filename [arguments]
```

Conditionally perform a command.

```
if [ test-commands ]; then  
    consequent-commands  
else  
    alternate-consequent-commands  
fi
```

# Part 1: case

Conditionally perform a command.

```
case word in  
    pattern)  
        command-list  
    ;;  
    pattern)  
        command-list  
    ;;  
esac
```

# Part 1: Testing source, if – then & case

```
$ cp ~/mlbII/echo4 .
$ more echo4
    ... Display contents of echo4 file ...
$ cp ~trainf/mlbII/fig1rc .
$ cp ~trainf/mlbII/fig2rc .
$ more fig1rc
    ... Display contents of fig1rc file ...
$ more fig2rc
    ... Display contents of fig2rc file ...
$ cp fig1rc .echorc
$ ./echo4
    ... Display output from echo4 ...
$ tail -5 fig2rc > .echorc
$ ./echo4
    ... Display output from echo4 ...
$
```

# Part 1: function

Define a *function\_name* that can be called to execute commands.

```
function function_name {  
    command-list  
}
```

# Part 1: function

```
$ cp ~/mlbII/part1.sh .  
$ more part1.sh  
    ... Display contents of part1.sh file ...  
$
```



Part 2

# Script Project

read, if - then - elif, while, let, if with "and",  
return, function

---

# Part 2: read

Read a line from standard input.

```
read    [-ers] [-a aname] [-p prompt]  
         [-t timeout] [-n nchars] [-d delim]  
         [name...]
```

**-r** If this option is given, backslash does not act as an escape character.

# Part 2: Testing read

```
$ cp ~/mlbII/read1 .
$ more read1
    ... Display contents of read1 file ...
$ cp ~/mlbII/read2 .
$ more read2
    ... Display contents of read2 file ...

$ ./read1
1 1.8 2 data x y ← type this and press return
1 1.8 2 data x y
$ ./read1
1 1.8\ ← type this and press return
1 data x y ← type this and press return
1 1.81 data x y
$ ./read2 ← type this and press return
1 1.8 2 data x y
1, 1.8
$ ./read2 ← type this and press return
1 1.8\
1, 1.8\
$
```

# Part 2: if – then – elif

Conditionally perform a command.

```
if [ test-commands ]; then  
    consequent-commands  
elif [ more-test-commands ]; then  
    more-consequent-commands  
fi
```

- n True if tests nonzero (contains data).
- z True if tests zero (no data).

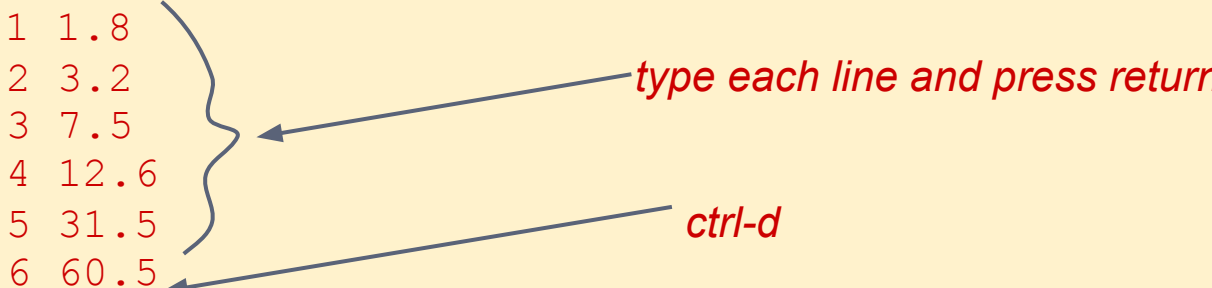
# Part 2: while

Execute consequent-commands as long as test-commands has an exit status of zero

```
while test-commands; do  
    consequent-commands  
done
```

# Part 2: Testing if – then – elif & while (good file)

```
$ cp ~/mlbII/while1 .
$ more while1
    ... Display contents of while1 file ...
$ cat > goodfile
1 1.8
2 3.2
3 7.5
4 12.6
5 31.5
6 60.5
$ ./while1 <goodfile > good.dat
$ more good.dat
    ... Display contents of good.dat file ...
$
```



The diagram shows a list of six lines of text: "1 1.8", "2 3.2", "3 7.5", "4 12.6", "5 31.5", and "6 60.5". A blue bracket on the left side of these lines is connected by an arrow to the text "type each line and press return". Another blue arrow points from the text "ctrl-d" to the line "6 60.5".

# Part 2: Testing if – then – elif & while (bad file)

```
$ cp goodfile badfile
$ vim badfile
    ... Delete 7.5 on line 3, save file and exit ...
$ more badfile
    ... Display contents of badfile file ...
$ ./while1 < badfile > bad.dat
line too short
$ more bad.dat
    ... Display contents of bad.dat file ...
$
```

# Part 2: Testing if – then – elif & while (warning file)

```
$ cp goodfile warningfile
$ vim warningfile
    ... Change line 3 and 6 to the following lines
        3 7.5 4.5
        6 60.5 too much data
    ...
$ more warningfile
    ... Display contents of warningfile file ...
$ ./while1 < warningfile > warning.dat
line too long, unexpected: 4.5
line too long, unexpected: too much data
$ more warning.dat
    ... Display contents of warning.dat file ...
$
```



# Part 2: let & if with “and”

Perform arithmetic on shell variables.

```
let expression [expression]
```

Test-commands using and

```
if [ expr1 -a expr2 ]; then  
    if both expr1 and expr2 are true.  
    consequent-commands  
fi
```

# Part 2: Testing let & if with “and”

```
$ cp ~/mlbII/while2 .
$ more while2
    ... Display contents of while2 file ...
$ ./while2 < goodfile > good.dat && echo "good data file"
good data file
$ ./while2 < badfile > bad.dat && echo "good data file"
line 3 too short
$ ./while2 < warningfile > warning.dat && echo "good data file"
line 3 too long, unexpected 4.5
line 6 too long, unexpected too much data
good data file
$
```

# Part 2: return

Causes a shell function to exit with the return value n.

```
return [n]
```

# Part 2: function

```
$ cp ~/mlbII/part2.sh .  
$ more part2.sh  
    ... Display contents of part2.sh file ...  
$
```

Part 3

# Script Project

## Putting it all together

# Part 3: Putting it all together

Get functions: die, gnucommands, datafile

```
source functions.sh
```

Get variables from run control file

```
[ -e .makefigrc ] || die "file \".makefigrc\" does not exist"  
source .makefigrc
```

Check for data file and set command file

```
[ "$dataFile" ] || die "no data file name specified"  
commandFile=${commandFile:-$dataFile.gnuplot}
```

# Part 3: Putting it all together

## Make output files

- dataFile using function datafile

```
datafile >$dataFile || die "some lines too short"
```

- commandFile using function gnucommands

```
gnucommands >$commandFile
```

- imageFile using Gnuplot

```
gnuplot $commandFile
```

# Part 3: Putting it all together

```
$ cp ~/mlbII/makefig1 .
$ more makefig1
    ... Display contents of makefig1 file ...
$ cp ~/mlbII/functions.sh .
$ more functions.sh
    ... Display contents of functions.sh file ...
$ cp fig1rc .makefigrc
$ ./makefig1 <badfile && echo "figure ready"
line 3 too short
makefig: some lines too short
$ ./makefig1 <warningfile && echo "figure ready"
line 3 too long, unexpected 4.5
line 6 too long, unexpected too much data
figure ready
$ ./makefig1 <goodfile && echo "figure ready"
figure ready
```



# Part 3: Putting it all together

```
$ firefox fig1.svg &
[1] 487
$ jobs
[1]+  Running                  firefox fig1.svg &
$ cp fig2rc .makefigrc
$ ./makefig1 <goodfile && echo "figure ready"
figure ready
$ firefox fig2.png
$ jobs
[1]+  Running                  firefox fig1.svg &
$ ps
  PID TTY          TIME CMD
  487 pts/6        00:00:01 firefox
  519 pts/6        00:00:00 dbus-launch
 2350 pts/6        00:00:00 ps
26767 pts/6        00:00:00 bash
```

# Part 3: Putting it all together

```
$ kill %1
$ jobs
[1]+  Terminated                  firefox fig1.svg
$ ps
  PID TTY          TIME CMD
 2993 pts/6        00:00:00 ps
26767 pts/6        00:00:00 bash
$  firefox &
[1] 13038
$ ps
  PID TTY          TIME CMD
13038 pts/6        00:00:00 firefox
13067 pts/6        00:00:00 dbus-launch
13171 pts/6        00:00:00 ps
26767 pts/6        00:00:00 bash
$ kill 13038
```

# Exercises

---

# Exercises

- Complete *Bash scripting Tutorial* [http://www.linuxconfig.org/Bash\\_scripting\\_Tutorial](http://www.linuxconfig.org/Bash_scripting_Tutorial)
- Complete *Advanced Bash-Scripting Guide* <http://tldp.org/LDP/abs/html/>