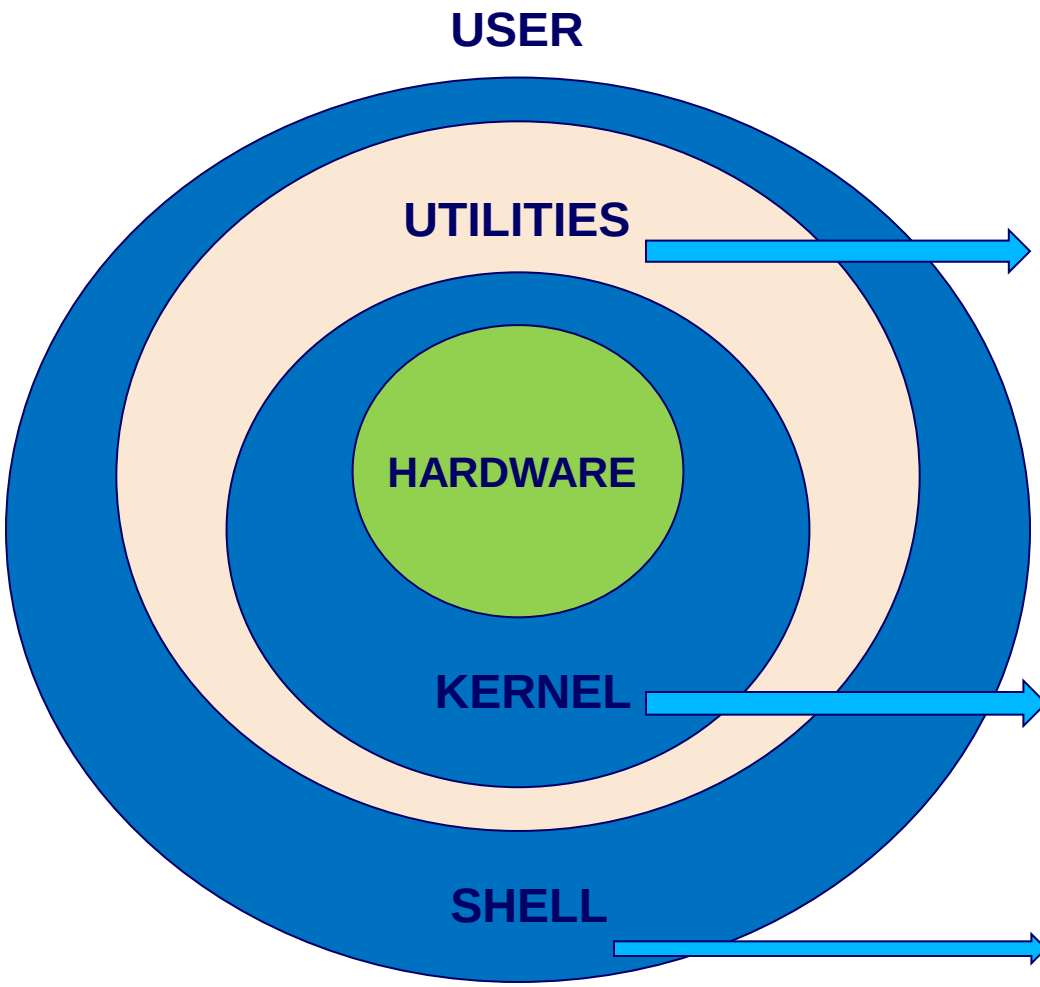
LPIC-1 TRAINING COURSE

Topic 103: GNU and UNIX command

Contents

1. Understanding Command-Line Basics

The structure of Unix OSes



❖Can be used separately or put together in various ways to carry out useful tasks
❖Interact with the kernel through ***system call***

❖Interacts directly with the hardware through ***device drivers*** those are built into the kernel
❖Provides sets of services that can be used by programs, insulating these programs from the underlying hardware

❖Reads and interprets commands to execute (a) program(s)
❖Also be a programming language

What is a shell

- A program that accepts and executes commands
- Provides added functionalities such as:
  - Command/filename completion.
  - History manipulation.
  - Definition schemes for aliases/functions to allow creating custom behaviours.
- Supports programming contructs to build complex commands from smaller parts (a script)

Shell categories

- Bourne shell compatible:
  - ***Bourne shell (sh):*** original Unix shell, written by Steve Bourne
  - ***Bourne-again shell (bash):*** default shell on most Linux systems & Mac OS X
  - ***Korn shell (ksh):*** developed by David Korn
- C shell compatible:
  - ***C shell (csh):*** the syntax was strongly influenced by the C programming language.
  - ***Extended C shell (tcsh):*** has replaced ***csh*** entirely on some versions of UNIX.

Internal and External Commands

- Internal commands are built into the shell
  - Eg: **`cd`**, **`pwd`**, **`echo`**, **`exec`**, **`time`**, **`set`**, **`exit`**
- External command are installed programs
  - Program file must be marked as executable
  - Should be run by typing a full path
    - Eg: **`/home/job/myprog`**, **`~/myprog`**, **`./myprog`**
  - Shell can check it **PATH** variable to find a program to execute
    - Eg:

```
echo $PATH
/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin
which rpm
/bin/rpm
```

Commands and Sequences

- Commands have ***command name***, ***options*** and ***paramters***
  - Eg: **`rpm -q gcc-c++`**
- All characters after a **#** character are ignored
- Command return value: ***0*** for success, *non-zero* for failure
  - Get the return value of the previous command: **`echo $?`**
- Command sequences:
  - Execute commands in succession:
    `cmd1;cmd2;cmd3`
  - Execute command only if the previous command return success:
    `cmd1&&cmd2&&cmd3`
  - Execute command only if the previous command return failure:
    `cmd1||cmd2||cmd3`

Perform Some Shell Command Tricks

- Command completion:
  - Eg: **`cront<TAB>, system-con <TAB><TAB>`**
- Command history: up/down key, **Ctrl-R**, **`history, !n`**
- Move whithin line: left/right key, **Ctrl+A**, **Ctrl+E**
- Transpose text: **Ctrl+T**
- Change case: **Esc-U**, **Esc-L**, **Esc-C**

Environment Variables

- Hold data to be referred to by the variable name
- Part of the environment of a program
- Working with environment variable:
  - Create a variable:
    MYVAR="Hello"
  - Export a variable to be environment variable
    export MYVAR
    - Single form: **export MYVAR="Hello"**
  - Read data from variable: use **$MYVAR**
    echo $MYVAR
  - Remove a variable:
    unset MYVAR
  - View entire environment: **env**

Common Environment Variables

Name	Function
USER	The name of the logged-in user
UID	The numeric user id of the logged-in user
HOME	The user's home directory
PWD	The current working directory
SHELL	The name of the shell
$	The process id (or *PID*of the running bash shell (or other) process
PPID	The process id of the process that started this process (that is, the id of the parent process)
?	The exit code of the last command

Getting Help

- **man** (manual) is a Linux text-based help system
  - Categorized into several sections
    - **`man man`**
  - To learn about a command:
    `man [section] cmd`
    - Move forward a page: spacebar
    - Move back a page: **Esc-V**
    - Search for text: **/<keyword>**
    - Exit man page: **`q`**
  - To search for a command doing specific work:
    `man -k <work>`
    - You may have to run **`makewhatis`** first
- **info** pages: same purpose as **man**, but use hypertext format

2. Using Streams, Redirection and Pipes

Types of Streams

- 3 standard I/O streams
  - ***stdin***: standard *input* stream, provides input to commands
    - Default is from keyboard
  - ***stdout***: standard *output* stream, displays output from commands
    - Default is on the screen
  - ***stderr***: standard *error* stream, displays error output from commands
    - Default is on the screen

Redirecting Input and Output

- Redirecting Output:
  - Send **stdout** to file (create/overwrite): **`cmd > file`**
  - Send **stdout** to file (create/append): **`cmd >> file`**
  - Send **stderr** to file (create/overwrite): **`cmd 2> file`**
  - Send **stderr** to file (create/append): **`cmd 2>> file`**
  - Send both **stdout** and **stderr** to file: **`cmd &> file`**
- Redirecting Input:
  - Read **stdin** from file: **`cmd < file`**
  - Read **stdin** on the following lines (a.k.a ***here document***):

```
cmd << EOF
your text here
EOF
```

- Cause the specified file to be used for both **stdin** and **stdout**: **`cmd <> file`**
- Direct **stdout** to both console and file: **`cmd | tee file`**

Piping Data Between Programs

- Data pipe (pipeline) redirect the 1st program's **stdout** to the 2nd program's **stdin**
- Denoted by a vertical bar (|)
- Example:

```
ls -a /tmp | grep txt
ls -a /tmp | grep txt | wc -l
```

Generating Command Lines

- **`xargs`** builds a command from its standard input
  - Syntax:
    `xargs [options] [command [initial-arguments]]`
  - Example:
    `find ./ -name “*~” | xargs rm`
- Backtick (`) is similar to **`xargs`** in many ways
  - Example:
    `rm `find ./ -name “*~”`

Exercise

1. View your current working directory
2. List the content of your current directory
3. List all the commands that started with **system**
4. View your command history
5. Re-run the first command in your history
6. View your current shell
7. View your **PATH**
8. Find out where the command setup is
9. Test to see whether your system has a **/tmp/testing** directory or not
10. View the return value of the previous command
11. Re-run the command in step 9 with an addition: if this directory doesn't exist, create it.
12. How many **rpm** package are installed in your system?
13. Create a text file that contents just one lines: **`echo "I am here"`** with **`echo`** command. Name that file **hello**
14. Make that file executable with **`chmod a+x hello`**. Can you run this file just by typing it name? How can you run it?
15. Where can you place that file so that you can run it just by typing it name? Test your opinion.

Hints to the Exercise

1. View your current working directory: **`pwd` or `echo $PWD`**
2. List the content of your current directory: **`ls`**
3. List all the commands that started with **system**: **`system<TAB><TAB>`**
4. View your command history: **`history`**
5. Re-run the first command in your history: **`!!`**
6. View your current shell: **`echo $SHELL`**
7. View your **PATH**: **`echo $PATH`**
8. Find out where the command setup is: **`which setup`**
9. Test to see whether your system has a **/tmp/testing** directory or not: **`ls /tmp/testing;echo $?`**
10. View the return value of the previous command: **`echo $?`**
11. Re-run the command in step 9 with an addition: if this directory doesn't exist, create it. **`ls /tmp/testing || mkdir /tmp/testing`**
12. How many **rpm** package are installed in your system? **`rpm -qa | wc -l`**
13. Create a text file that contents just one lines: **`echo "I am here"`** with **echo** command. Name that file **hello**. **`echo 'echo "I am here"' > /tmp/hello`**
14. Make that file executable with **`chmod a+x hello`**. Can you run this file just by typing it name? How can you run it? **`/tmp/hello`**
15. Where can you place that file so that you can run it just by typing it name? Test your opinion. *Place this file in **/usr/bin** or add **/tmp** to your **PATH** (`PATH=$PATH:/tmp`)*

3. Processing Text Using Filters

File-Combining Commands

- Combining files with **`cat`**
  - Eg: **`cat first.txt second.txt > combined.txt`**
- Joining files by field with **`join`**
  - Eg: **`join first.txt second.txt > joined.txt`**
- Merging files line by line with **`paste`**
  - Eg: **`paste first.txt second.txt > pasted.txt`**

```
cat first.txt
1 shoes
2 laces
3 socks
```

```
cat second.txt
1 $40.00
2 $1.00
3 $2.00
```

```
cat combined.txt
1 shoes
2 laces
3 socks
1 $40.00
2 $1.00
3 $2.00
```

```
cat joined.txt
1 shoes $40.00
2 laces $1.00
3 socks $2.00
```

```
cat pasted.txt
1 shoes 1 $40.00
2 laces 2 $1.00
3 socks 3 $2.00
```

File-Transforming Commands

- Converting tabs to spaces with **expand**
  - Syntax: **expand [-t num] [filename]**
- Converting spaces to tabs with **unexpand**
  - Syntax: **unexpand [-t num] [filename]**
- Displaying files in octal with **od**
  - Syntax: **od [filename]**
- Sorting files with **sort**
  - Syntax: **sort [-f] [-n] [-r] [-k field] [filename]**
- Breaking a file in to pieces with **split**
  - Syntax: **split [-b size | -c size | -l line] [infile] outfile**
- Translating characters with **tr**
  - Syntax: **tr [options] SET1 [SET2]**
- Deleting duplicate lines with **uniq**
  - Syntax: **uniq [filename]**

File-Formatting Commands

- Reformatting paragraphs with **`fmt`**
  - Example: **`fmt -w 80 mytext.txt`**
- Numbering lines with **`nl`**
  - Example: **`nl -b a mytext.txt`**
- Preparing a file for printing with **`pr`**
  - Example: **`pr myfile.txt`**

File-Viewing Commands

- Viewing the starts of files with **head**
  - Syntax:
    head [-n line] [filename]
- Viewing the ends of files with **tail**
  - Syntax:
    tail [-f [--pid=PID]] -n line [filename]
- Paging through files with **less**
  - Syntax:
    less [filename]

File-Summarizing Commands

- Extracting text with **cut**
  - Syntax:
    cut [-c list] [-f list -d char] [filename]
  - Example:
    ifconfig eth0|grep HWaddr|cut -f 11 -d " "
- Obtaining a word count with **wc**
  - Syntax: **wc [-l|-w|-c|-m|-L] [filename]**

Exercise

Using text filters to answer these questions regarding **/var/log/messages** file.

1. Display the first 25 messages
2. Display the last 15 messages
3. Display the messages from 70 to 85
4. View this file in following mode
5. Display all messages that is generated in the last hour. Number these messages
6. Display only *facilities* (showed in column 5) that generate messages
7. How many *facilities* are there in step 5

Hints to the Exercise

1. Display the first 25 messages
   `head -n 25 /var/log/messages`
2. Display the last 15 messages
   `tail -n 15 /var/log/messages`
3. Display the messages from 70 to 85
   `head -n 85 /var/log/messages | tail -n 16`
4. View this file in following mode
   `tail -f /var/log/messages`
5. Display all messages that is generated in the last hour. Number these messages
   `grep “Nov 11 20:” /var/log/messages | nl`
6. Display only *facilities* (showed in column 5) that generate messages
   `cut -f 5 -d “ ” /var/log/messages`
7. How many *facilities* are there in step 5
   `cut -f 5 -d “ ” /var/log/messages | sort | uniq | wc -l`

4. Using Regular Expressions

What are Regular Expressions

- A formula for maching strings that follow some pattern
  - So called **regexp**
- Many tools support *regexp*: **`vi`, `grep`, `sed`, `awk`, `perl`...**
- Regular expression terminologies:
  - **Literal**: any character used in a search or maching expression (Eg: ***ind*** to search for w***ind***ows)
  - **Metacharacter**: special character that have unique meaning (Eg: *, ?, ^, $)
  - **Escape sequence**: a way of indicating that we want to use one of our metacharacters as literal (Eg: \?, *)
  - **Search expression**: the pattern we use to find what we want
  - **Target string**: the string that we want to search for

Metacharacters

- **.** matches any single character
- **$** matches the end of a line
- **^** matches the beginning of a line
- ***** matches zero or more occurences of the character immediately preceding
- **** quoting character
- **[ ]** matches anyone of the characters between the brackets. **[^]** will negates the expression
- **-** (use inside **[ ]**) define a range, eg [1-9]
- **( )** group parts of expression together
- **|** OR 2 conditions together
- **?** Matches 0 or 1 occurrence of the character

Using *grep*

- **grep** searches for specified string in files
- Syntax:

  grep [options] regexp [files]

  - Options:

-c	count matching lines
-f file	takes pattern input from file
-i	ignore case
-r	recursive
-v	output lines not containing **regexp**

- Examples:

```
grep -r eth[01] /etc/*
ps ax | grep xterm
```

Examples with *grep*

- Example Strings:
  - `STRING1: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)`
  - `STRING1: Mozilla/4.75 [en] (X11;U;Linux2.2.16-22 i586)`
- Search with: **`grep RegExp STRING1 STRING2`**

RegExp	STRING1 matched words	STRING2 matched words
`m`	`compatible`	`<no matches>`
`5 [`	`<no matches>`	`Mozilla/4.75 [en]`
`in[du]`	`Windows`	`Linux2`
`.in`	`Windows`	`Linux2`
`x[0-9A-Z]`	`<no matches>`	`Linux2`
`W*in`	`Windows`	`Linux2`
`[^A-M]in`	`Windows`	`<no matches>`
`[a-z]\)$`	`DigExt)`	`<no matches>`
`^([L-Z]in)`	`<no matches>`	`<no matches>`
`(W|L)in`	`Windows`	`Linux`

Using *sed*

- **sed** directly modifies the contents of files, sending the changed file to standard output
- Syntax:
  sed [options] -f script-file [input-file]
  sed [options] script-text [input-file]
- Examples:
  - Substituing words:
    sed 's/{oldwords}/{newwords}/' input-file
  - Multiple substituing:
    sed -e 's/{old1}/{new1}/' -e 's/{old2}/{new2}/' input-file
  - Selective substituing on lines contain **pattern**
    sed '/{pattern}/s/{old}/{new}/' input-file
  - Deleting words:
    sed 's/{words}//g' input-file
  - Delete lines contain word:
    sed '/{word}/d' input-file

Exercise

1. Create a new file called **myfile** containing the lines:

```
Using grep,
fgrep and
egrep
to grep for 99% of the cats
% these are two
% commented lines
```

2. Use **grep** to:
   a. Output only uncommented lines (lines not start with **%)**
   b. Find all lines containing '**grep**' exactly (not '**egrep**' nor '**fgrep**'. Use **-w** to match the word)
   c. Find lines containing words starting with an '**a**'
3. Use **sed** to do the following changes to myfile:
   a. In the first line, substitue '**grep,**' with '**soap**'
   b. Delete '**fgrep**' in the second line
   c. Substitue '**egrep**' with '**water**'
   d. In the fourth line, replace 'grep for' with '**wash**'
   e. Delete all commented lines (lines start with **%**)

Hints to Exercise

1. Create a new file called **myfile**

```
cat <<EOF >myfile
Using grep,
fgrep and
egrep
to grep for 99% of the cats
% these are two
% commented lines
EOF
```

2. Use **grep** to:
   a. Output only uncommented lines (lines not start with **%)**
      `grep -v ^% myfile`
   b. Find all lines containing '**grep**' exactly (not '**egrep**' nor '**fgrep**'. Use **-w** to match the word)
      `grep -w grep myfile`
   c. Find lines containing words starting with an '**a**'
      `grep -w a.* myfile`
3. Use **sed** to do the following changes to myfile:
   a. In the first line, substitue '**grep,**' with '**soap**'
      `sed '/Using/s/grep,/soap/' myfile`
   b. Delete '**fgrep**' in the second line
      `sed 's/fgrep//' myfile`
   c. Substitue '**egrep**' with '**water**'
      `sed 's/egrep/water/' myfile`
   d. In the fourth line, replace '**grep for**' with '**wash**'
      `sed 's/grep for/wash/' myfile`
   e. Delete all commented lines (lines start with **%**)
      `sed '/^%/d' myfile`

5. Managing Processes

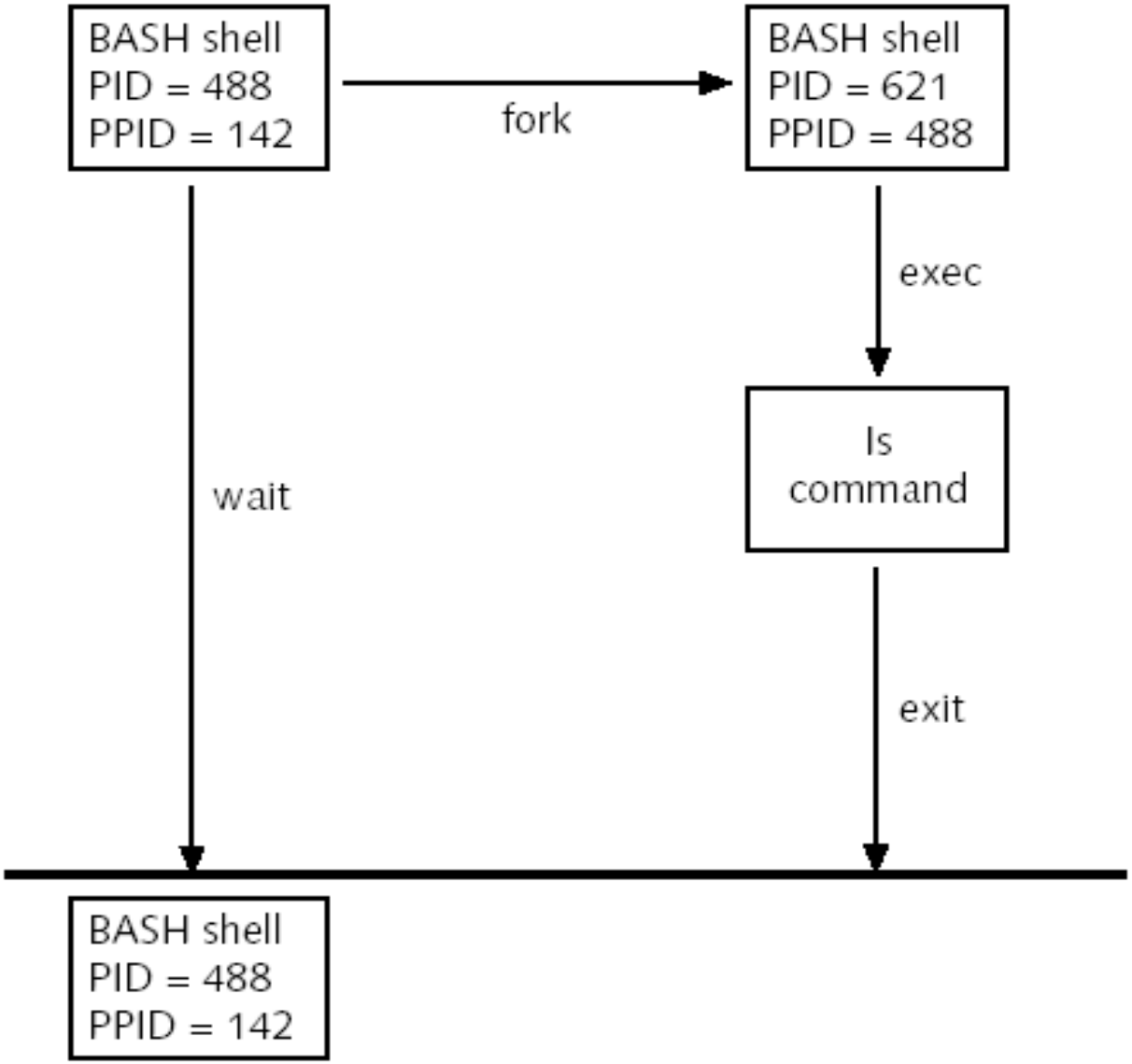
Linux Processes

- ***Program***: Structured set of commands stored in an executable file
  - Executed to create a process
- ***Process***: Program running in memory and on CPU
  - ***User process***: Process begun by user on a terminal
  - ***Daemon process***: System process
    - Not associated with a terminal

Linux Processes (cont')

- ***Child process***: Process started by another process (parent process)
- ***Parent process***: Process that has started other processes (child processes)
- ***Process ID (PID)****:* Unique identifier assigned to a process
- ***Parent Process ID (PPID)***: Parent process's PID

Process Execution



Examining Process Lists

- **ps** displays processes's status
  - Syntax: **ps [options]**
  - Options:

-e	display all processes on the system
-x	display all processes owned by the user
-u user|UID	display processes owned by a given user
-f	display extra information
-H	display process hierarchy

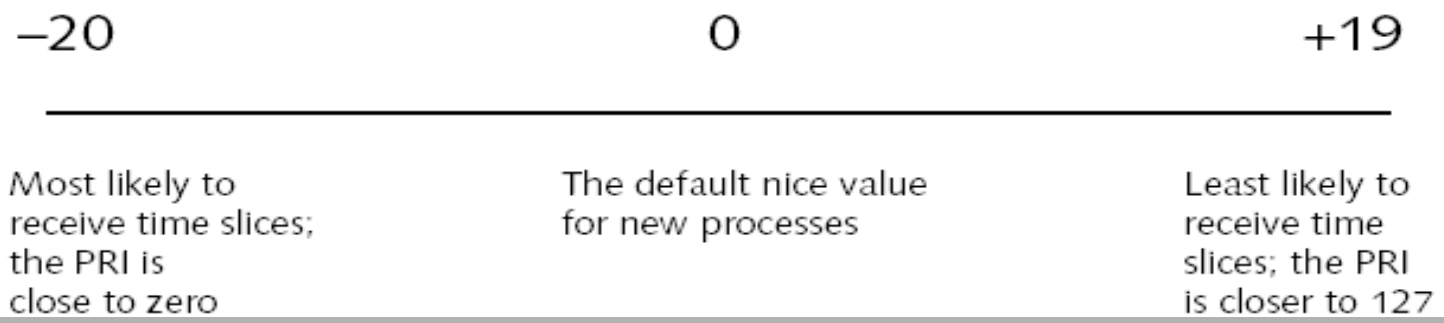
- **top** displays processes's status and updates its every few seconds
  - Syntax: **top [-d delay] [-p PID] [-n inter] [-b]**
  - Commands in top: **h** (help), **k** (kill a process), **q** (quit), **r** (change process's priority), **s** (change the display's update rate), **P** (sort the display by CPU usage), **M** (sort the display by memory usage)
- **jobs** displays processes associated with your session
  - Syntax: **jobs**

Foregound and Background Processes

- Foreground processes: BASH shell must wait for termination
  - Foreground process may be paused with **Ctlr+Z**
  - Resume a paused process in foreground: **`fg %jobID`**
    - **JobID** can be viewed by **`jobs`** command
- Background processes: BASH shell does not wait for termination
  - Resume a paused process in background: **`bg %jobID`**
  - Run a new process in background: **`command &`**
- **nohup**: run a program that will continue running even when you log out
  - Syntax: **`nohup command [&]`**

Managing Process Priorities

- **Time slice**: Amount of time (in ms) a process is given on a CPU
  - More **time slice** = more execution time = executes faster
- PRIORITY dictates number of **time slices** a process gets
  - **PRIORITY=PRI+NI**
  - Set **NI** to indirect affect **PRIORITY**
- Processes start with NI of 0
  - Change a process's NI as it starts: **`nice NI command`**
  - Change NI of a running process: **`renice NI [-p PID]`**



Killing Processes

- **kill** command: sent a **signal** to process
  - 64 types of kill's **signal**, affect processes in different ways
  - Common administrative kill signals:

SIGNAL	SIGID	Description
SIGHUP	**1**	Stops a process then restarts it with the same PID
SIGINT	**2**	Interupt signal (weekest signals). Same as using **Ctrl+C**
SIGQUIT	**3**	A.k.a core dump. Terminates a process and dump it memory to a file called **core** in the current working directory. Same as using **Ctrl+**
SIGTERM	**15**	Software termination. Default kill signal used by **kill** command
SIGKILL	**9**	Absolute kill signal (strongest signals). Forces the Linux kernel to stop executing the process

  - Syntax: `kill -s SIGNAL PID`
    `kill -SIGID PID`
  - If no **SIGNAL** given, **SIGTERM** assumed
- **killall**: kills all instances of a program by command name
  - Syntax: `killall command`

Exercise

1. Log into a virtual terminal (**Ctrl+Alt+F1**), view the **PID** of your current shell
2. Run **yes** command in background and redirect all output and error to **/dev/null**
3. View the **PID** and **PPID** of this **yes** process
4. Use **top** to view the CPU usage and NICE value of this **yes** process
5. Run 2 another **yes** processes in background (redirect all output and error to **/dev/null**), one with the NICE value of **-10** and one with the NICE value of **10**
6. Use **top** to view the CPU usage and NICE value of these 3 **yes** processes. Which one takes the most of CPU's time?
7. In **top**'s screen, press **r** to change the NICE value of the **yes** process with lowest priority to **-19** and notice the change of CPU usage.
8. In **top**'s screen, press **k** to kill the **yes** process with highest NICE value.
9. Exit **top** and kill all running **yes** processes with one command.

Hints to Exercise

1. Log into a virtual terminal (**Ctrl+Alt+F1**), view the **PID** of your current shell
   `echo $$`
2. Run **`yes`** command in background and redirect all output and error to **/dev/null**
   `yes >& /dev/null &`
3. View the **PID** and **PPID** of this **yes** process
   `ps -ef | grep yes`
4. Use **top** to view the CPU usage and NICE value of this **yes** process
   `top`
5. Run 2 another **yes** processes in background (redirect all output and error to **/dev/null**), one with the NICE value of **-10** and one with the NICE value of **10**
   `nice --10 yes > /dev/null &`
   `nice -10 yes > /dev/null &`
6. Use **top** to view the CPU usage and NICE value of these 3 **yes** processes. Which one takes the most of CPU's time?
   `top`
7. In **top**'s screen, press **r** to change the NICE value of the **yes** process with lowest priority to **-19** and notice the change of CPU usage.
   `Press r -> Type the PID of the process -> Type -19`
8. In **top**'s screen, press **k** to kill the **yes** process with highest NICE value.
   `Press k -> Type the PID of the process -> Press Enter to send default SIGNAL`
9. Exit **top** and kill all running **yes** processes with one command.
   `killall yes`

6. File and Directory Management

Listing directories entries with *ls*

- Syntax: **`ls [options] path`**
  - **`path`** can be *absolute* or *relative*
  - Options:
    - **`-a`** list special (hidden) files also
    - **`-l`** display some of the information stored in the inode
    - **`-t`** sort output by modification time
    - **`-S`** sort output by size
    - **`-r`** sort output in the reverse order

Change working directory with *cd*

- Syntax: **`cd path`**
  - **`path`** can be *absolute* or *relative*
- Some special path:
  - Change to parent directory
    `cd ..`
  - Change to your home directory
    `cd`
    `cd ~`
  - Change to ***user***'s home directory
    `cd ~username`
  - Change to the previous working directory
    `cd -`
- **`CDPATH`** contains a set of directories that should be searched when resolving relative paths

Copying, moving and deleting files

- **cp**: make a copy of one or more files or directory

  `cp [options] SOURCE DEST`

  - Use **`-r`** (recursive) option to copy subdirectory recursively
- **mv**: move or rename one or more files or directory
- **rm**: remove one or more files

Creating and removing directories

- **mkdir**: make directories
  - use ***–p*** option to make nested subdirectory
- **rmdir**: remove empty directories
- **rm -r**: remove both files and non-empty directories

Touching files with *touch*

- **touch** update file access and modification time or create empty files
- Set file modification time with **–d** or **–t**

```
[ian@echidna lpi103-2]$ touch -t 200908121510.59 f3
[ian@echidna lpi103-2]$ touch -d 11am f4
[ian@echidna lpi103-2]$ touch -d "last fortnight" f5
[ian@echidna lpi103-2]$ touch -d "yesterday 6am" f6
[ian@echidna lpi103-2]$ touch -d "2 days ago 12:00" f7
[ian@echidna lpi103-2]$ touch -d "tomorrow 02:00" f8
[ian@echidna lpi103-2]$ touch -d "5 Nov" f9
```

- **touch** can use the timestamp from reference file with ***–r*** option

touch –r ref_file filename

Finding files with *find* command

- **find** can search for files or directory using name, owner, size, timestamp…
- Syntax:
  `find DIRECTORY CRITERIA [-exec COMMAND {} \;]`
- Examples:
  - finding files by name:
    `find / -name *.conf`
  - finding files by type:
    `find . –type d –name bin`
  - finding files by timestamp:
    `find /var –mmin –60  –mmin +10 –type f -print`
  - finding files by size (b: block, c: Byte, k: kB)
    `find ~ -size -30k –size +10k -ls`
  - finding and acting on files:
    `find . –empty –exec rm –r {} \;`

Identifying files

- Linux does not require file's suffixes to identify a file type
- **file** command displays the type of data in one or more file
- Each file is classify using 3 types of test
  1. ***stat***: determine whether a file is empty or a directory
  2. ***magic*** test: check the file *magic number*
  3. ***language*** test: look at the file content to determine a programing languange

Compressing files

- **gzip**: uses the Lemple-Ziv algorithm
  - compression: **`gzip filename`**
  - uncompression: **`gunzip filename`**
- **bzip2**: uses the Burrows-Wheeler algorithm
  - compression: **`bzip2 filename`**
  - uncompression: **`bunzip2 filename`**
- Other compression tools: **compress/uncompress**, **zip/unzip**…

Archiving files

- General approaches to backup:
  - Full backup
  - Incremental backup
  - Differental backup
- **tar**, **cpio** and **dd** are commonly used for backup groups of files or whole partitions

Using *tar*

- Create an archive file from a set of input files and restores files from such archive
- Syntax: **`tar [options] filenames…`**
- Options:

`-f`	input/output location
`-c`	create an archive
`-x`	extract an archive
`-v`	verbose output
`-z`	use gzip compression
`-j`	use bzip2 compression

- Examples:
  - Create an archive of home dir:
    `tar –cvzf /tmp/homdir.tar.gz ~`
  - Restore files from archive:
    `tar –xvzf /tmp/homdir.tar.gz`

Using *cpio*

- Operate in modes:
  - Create an archive: *copy-out* mode (-o)
    `cpio -o name-list > archive`
  - Restore from an archive: *copy-in* mode (-i)
    `cpio -i < archive`
  - Copy files to another location: *copy-pass* mode (-p)
    `cpio -p destination < name-list`
- Example:
  - **`ls | cpio -o > /tmp/test.cpio`**
  - **`cpio -i < /tmp/test.cpio`**

Using *dd*

- **dd** copies an input file to an output file
  - can perform conversions (eg: lowercase to uppercase)
  - can reblock a file
  - can skip or include only selected blocks of a file
  - can read and write to raw devices (eg: /dev/sda)
- Exampes:
  - **`dd if=sourcefile conv=ucase of=destfile`**
  - **`dd if=/dev/sda2 of=/backup/sda2.bk`**

Exercise

1. Make a new directory in **/tmp** called **bin**
2. Create a file called **newfile** in **/tmp/bin** with **cat**
3. Go to the root (**/**) directory. View the content of **newfile** from there
4. Which is the shortest command which will take you back to **/tmp/bin**
5. Which is the shortest command which will take you to your home directory
6. Which is the quickest way to make three new directory **/dir1/dir2/dir3**
7. Remove the **dir2** directories with **rmdir**, then remove **dir1** with **rm**
8. Copy the file **/boot/grub.conf** to **/tmp/grub.conf.bak**. Use **find** to find this new file.
9. Find all files in your home directory that have been modified from the last Sunday
10. Archive all files you found in Step 9 with **cpio**. Store the archive file in **/tmp**
11. View the size of the archive file.
12. Compress the archive file in Step 10 with **gzip**. What's its size after being compressed?

Exercise

1. Make a new directory in **/tmp** called **bin:** `mkdir /tmp/bin`
2. Create a file called **newfile** in **/tmp/bin** with **cat:**
   ```
   cat >/tmp/bin/newfile
   your text goes here
   [Ctrl+D]
   ```
3. Go to the root (**/**) directory. View the content of **newfile** from there: `cd /; cat /tmp/bin/newfile`
4. Which is the shortest command which will take you back to **/tmp/bin:** `cd -`
5. Which is the shortest command which will take you to your home directory: `cd`
6. Which is the quickest way to make three new directory **/dir1/dir2/dir3:**
   ```
   mkdir -p /dir1/dir2/dir3
   ```
7. Remove the **dir2** directories with **rmdir**, then remove **dir1** with **rm**
   ```
   rmdir /dir1/dir2/dir3; rmdir /dir1/dir2
   rm -r /dir1
   ```
8. Copy the file **/boot/grub.conf** to **/tmp/grub.conf.bak**. Use **find** to find this new file.
   ```
   cp /boot/grub.conf /tmp/grub.conf.bak
   find / -name grub.conf.bak
   ```
9. Find all files in your home directory that have been modified from the last Sunday
   ```
   find ~ -mtime -5 #assuming today is Friday, so the last Sunday is 5 days earlier
   ```
10. Archive all files you found in Step 9 with **cpio**. Store the archive file in **/tmp**
    ```
    find ~ -mtime -5 | cpio -o > /tmp/myarchive.cpio
    ```
11. View the size of the archive file: `du -h /tmp/myarchive.cpio`
12. Compress the archive file in Step 10 with **gzip**. What's its size after being compressed?
    ```
    gzip /tmp/myarchive.cpio
    du -h /tmp/myarchive.cpio.gz
    ```

7. File editing with *vi*

vi's modes

- Start editing by **`vi filename`**
- 2 modes of ***vi***
  - ***Command mode***: perform editing operation
    - moving around the file
    - searching, deleting, changing text
    - saving, replacing file
    - …
  - ***Insert mode***: type new text into the file at the insertion point
    - press **Esc** to return to *command mode*

Command mode: moving arround

- Moving arround commands:
  - **h** move left one character (or left-arrow key)
  - **j** move down to the next line (or down-arrow key)
  - **k** move up to the previous line (or up-arrow key)
  - **l** move right one character (or right-arrow key)
  - **w** move to the next word
  - **e** move to the next end of word
  - **b** move to the previous beginning of word
  - **Ctlr+F** scroll forward one page
  - **Ctlr+B** scroll backward one page
- Can use with number to execute many times
  - Eg: **5h** will move left 5 characters

Command mode: Getting out of vi

- Pressing Esc will leave *insert mode* and return to *command mode* to:
  - **:q!** quit vi and abandon all changes
  - **:w!** write the file (overwrite existing file)
  - **:w! *filename*** write to a new file
  - **ZZ** write the file if modified, then exit
  - **:e!** reload file from disk for editing
  - **:!** run a shell command

Command mode: Searching text

- You can search for text in file using regular expression
  - **/ <string>** search forward for <string>
  - **?<string>** search backward for <string>
  - **n** repeat the last search in either direction

Entering insert mode

- **i** insert before the character at the current position
- **a** insert after the character at the current position
- **c** change the current character
- **o** open a new line below the curent line
- **dd** delete the current line
- **x** delete the character at the cusor position
- **yy** copy the current line
- **p** put the last deleted/copied text after the current character

Exercise

1. As **root**, copy **/var/log/messages** to **/tmp**. Using **vi**'s search utility to find lines contain "***kernel***"
2. Copy **/boot/grub.conf** to **/tmp**, edit this file and try to copy/paste with **yy/p** and cut/paste with **dd/p**
3. Undo all the changes with **u**
4. Investigate the outcome of **:x**, **ZZ**, **:quit**, **:wq** and **:q!**. Which ones save and which one don't?
5. Investigate the outcome for the various inserting mode: **A**, **a**, **O**, **o** , **S** and **s**

Thank You !

BACKUP SLIDES