Lecture – 17

# SECTION -C

## Getting Started with UNIX

# Running Jobs in Background:

- A multi-tasking system lets a user to do more than one job at a time. Since there can be only one job in the foreground, the rest of the jobs have to run in the background.
- There are two ways of doing this – with the shell's *&* operator and the **nohup** command.
- The latter permits you to log out while your jobs are running, but former doesn't allow that (except in the C shell and Bash)

## (1) & : No Logging Out :

- The & is the shell's operator used to run a process in the background. The parent in this case doesn't wait for the child's death.
- Just terminate the command line with an & ; the command will run in the background:

  **$ sort –o emp.lst emp.lst &**

  **550** ***The job's PID***

- The shell immediately returns a number –the PID of the invoked command(550) . The prompt is returned and the shell is ready to accept another command even though the previous command has not been terminated yet.
- The shell, however, remains the parent of the background process Using an & you can run as many jobs in the background as the system load permits.
- Background execution of a job is a useful feature that you should utilize to transfer time –consuming or low- priority jobs to the background, and run the important ones in the foreground.

## (2) Nohup : Logout safely:

- Background jobs ceases to run, however, when a user logs out (the C shell and Bash excepted).
- That happens because her shell is killed. And when the parent is killed, its children are also normally killed (subject to certain conditions). The UNIX system permits a variation in this default behaviour .
- The **nohup** (or no hangup) command, when prefixed to a command, permits execution of the process even after the user has logged out. You must use the & with it as well:

```
$ nohup sort emp.lst &
586
Sending output to nohup.out
```

- The shell returns the PID this time too, and some shells display this message as well.
- When the **nothup** command is run in these shells, **nohup** sends the standard output of the command to the file **nohup.out.**
- If you don't get this message, then make sure that you have taken a care of the output, using redirection if necessary.
- You can now safely log out of the system without aborting the command.
- When you use the **ps** command after using **nohup** from another terminal( and it has not been completed already), you'll notice something quite significant:

**$ ps –f -u kumar**

**UID PID PPID C STIME TTY TIME COMMAND**

- Looks what happen this time. The shell died (rather, was killed) on logging out but its child (sort) didn't ; it turned into an orphan.
- The kernel handles such situations by reassigning the PPID of the orphan (**sort**) to the system's **init** process (PID 1) – the parent of all shells. When the user logs out, **init** takes over the percentage of any process run with **nohup.**
- In this way, you can kill a parent ( the shell) without killing its child (**sort**).
- If you run more than one command in a pipeline, you should use the **nohup** command at the beginning of each command in the pipeline:

  nohup  grep 'director ' emp.lst & | nohup sort  &
- **(The *grep* command allows you to search one file or multiple files for lines that contain a pattern. Exit status is 0 if matches were found, 1 if no matches were found, and 2 if errors occurred**

# UNIX: vi Editor

## General Introduction:

The vi editor (short for visual editor) is a screen editor which is available on almost all Unix systems. vi has no menus but instead uses combinations of keystrokes in order to accomplish commands.

## Starting vi

To start using vi, at the Unix prompt type **vi** followed by a file name. If you wish to edit an existing file, type in its name; if you are creating a new file, type in the name you wish to give to the new file.

**$ vi *filename***

Then hit Return. You will see a screen similar to the one below which shows blank lines with tildes and the name and status of the file.

~

~

**"myfile" [New file]**

# vi's Modes and Moods

- **vi has two modes:** the **command mode** and the **insert mode.**
- It is essential that you know which mode you are in at any given point in time.
- When you are in command mode, letters of the keyboard will be interpreted as commands.
- When you are in insert mode the same letters of the keyboard will type or edit text. vi always starts out in command mode.
- When you wish to move between the two modes, keep these things in mind. You can type **i** to enter the insert mode.
- If you wish to leave insert mode and return to the command mode, hit the **ESC** key. If you're not sure where you are, hit **ESC** a couple of times and that

# General Command Information

vi uses letters as commands. It is important to note that in general vi commands:

- are case sensitive - lowercase and uppercase command letters do different things
- are not displayed on the screen when you type them
- generally do not require a **Return** after you type the command.

# Entering Text

- To begin entering text in an empty file, you must first change from the command mode to the insert mode.
- To do this, type the letter **i**. When you start typing, anything you type will be entered into the file.
- Type a few short lines and hit **Return** at the end of each of line. Unlike word processors, vi does not use word wrap.
- It will break a line at the edge of the screen.
- If you make a mistake, you can use the Backspace key to remove your errors.
- If the Backspace key doesn't work properly on your system, try using the Ctrl h key combination.

## A Quick Word about Customizing Your vi Environment

- There are several options that you can set from within vi that can affect how you use vi.
- For example, one option allows you to set a right margin that will then force vi to automatically wrap your lines as you type.
- To do this, you would use a variation of the **:set** command.
- The **:set** command can be used to change various options in vi. In the example just described, you could, while still in vi, type **:set wrapmargin=10** to specify that you wish to have a right margin of 10.
- Another useful option is **:set number**. This command causes vi to display line numbers in the file you are working on.

# Useful vi Commands

**Cut/Paste Commands:**

- x delete one character (destructive backspace)
- dw delete the current word (Note: ndw deletes n numbered words)
- dd delete the current line (Note: ndd deletes n numbered lines)
- D delete all content to the right of the cursor
- d$ same as above
- :u undo last command
- p, P paste line starting one line below/above current cursor location
- J combine the contents of two lines

# Cursor Relocation commands:

- :[n] goto line [n]
- shift g place cursor on last line of text
- h/l/j/k move cursor left, right, down and up
- ^f/^b move forward, backward in text, one page
- ^u/^d move up, down one half page
- $ move to end of line
- 0 move to beginning of line

# Entering the Insert Mode:

- i Begin inserting text at current cursor location
- I Begin inserting text at the beginning of the current line
- a Begin appending text, one character to the right of current cursor location
- A Begin appending text at the end of the current line
- o/O Begin entering text one line below\above current line
- ESC Exit insertion mode and return to command mode

# Applications

- **To Kill a specific background job using kill %**
- If you want to kill a specific background job use, kill %job-number. For example, to kill the job 2 use

```
# kill %2
```

- To kill a foreground jobs, 4 Ways to Kill a Process — kill, killall, pkill, xkill.

# Research

- **Running Background Job (matlab example)**
- Running a background job on the linux machines allows you to run your code on fast machines for an extended time without having to stay logged in. (Remember to adhere to our policies on number of jobs, where you can run them and breakpointing.) Though this documentation page uses matlab as an example, this would apply to C code, or R. For running multiple jobs in succession see the documentation page on shell scripts. The shell is what you type commands in, and is a complex scripting tool on its own.
- To run background job, first you must log in via ssh to a linux machine. This can be accomplished using the PuTTY client on Windows, or using the **ssh** command from a terminal on MacOS. You can also **ssh** from one linux machine to another.
- Once on a linux machine, let's analyze a line that starts a background job:
- rohan$ **matlab -nodesktop -nodisplay < file.m &> file.out &**
- or for R: rohan$ **R --no-save < file.R &> file.out &**
- "rohan$" is just the prompt with the hostname that you will see.
- **matlab** is the command to start matlab. Since we are no working graphically (you can't display graphics in a background job) we add the "-nodesktop -nodisplay" so that matlab doesn't try to start graphics and crash.