

# Linux Notes

## Lecture -1

### Linux ideas and History

Upon completion of this unit, you should be able to:

- Explain the nature of open source software
- Discuss the origins of Linux
- List the Red Hat operating system distributions
- Explain basic Linux principles

### What is open source?

- **Open source: software and source code available to all.**
  - The freedom to distribute software and source code
  - The ability to modify and create derived works
  - Integrity of author's code
- **The free Software foundation and the four freedoms**
- The software and the source code must be freely distributable
- All must be able to modify the source code and create derived works
- To maintain the integrity of the original author's work, the license may require that so the code be provided in patch form
- The license must be nondiscriminatory with respect to persons, groups, or fields of endeavor: it must be free of restrictions that can limit the license. For example, it may not require that the software be part of particular distribution: it must not restrict other non-OSS software; and it may not require the use of technology to apply the license.

### Linux origins

- **1984: The GNU project and the free software foundation**
  - Creates open source version of UNIX utilities
  - Creates the General Public License (GPL)
    - Software license enforcing open source principles
- **1991: Linus Torvalds**
  - Creates open source, UNIX like kernel, released under the GPL
  - Ports some GNU utilities, Solicits assistance Online
- **Today:**
  - Linux kernel + GNU utilities = complete, open source, UNIX Like operating system
    - Packaged for targeted audiences as distribution

The next major event in the development of Linux kernel itself. Linus torvalds, a graduate student in Finland. began developing UNIX-like kernel in the late '90's. he first announced his work in a now-famous email message on the comp.os.minix mailing list:

From: [torvalds@klaava.helsinki.fi](mailto:torvalds@klaava.helsinki.fi) (linus benedict torvalds)  
Newsgroup: comp.os.minix  
Subject: what would you like to see most in minix?  
Summary: Small poll for my new operating system  
Message-ID: <[1991aug25.205708.9441@klaava.helsinki.fi](mailto:1991aug25.205708.9441@klaava.helsinki.fi)>  
Date: 25 Aug 91 20:57:08 GMT  
Organization: university of Helsinki

Hello everybody out there using minix

I'm doing a (free) operating system (just a hobby, won't be big and professional like GNU) for 386 (486) AT clones. This has been brewing since April, and is starting to get ready. I would like any feedback on things people like/ dislike in minix, as my OS resemble it somewhat (same physical layout of the file-system (due to practical reason) among other things).

I've currently ported bash (1.08) and things seem to work this implies that I'll get something practical within a few months, and I would like to know features most people would want. Any suggestions are welcome. But I won't promise I'll implement them :-)

Linus ( [torvalds@kruuna.helsinki.fi](mailto:torvalds@kruuna.helsinki.fi) )

Ps. yes – it's free of any minix code, and it has a multi – threaded fs.  
It is not portable (uses 386 task switching etc). And it probably never will support anything other than AT-hard disk, as that's all I have: - (.

## Red Hat Distributions

- Linux Distributions are OSes based on the Linux kernel
- Red Hat Enterprise Linux
  - **Stable, thoroughly tested software**
  - **Professional support services**
  - **Centralized management tools for large network**
- The Fedora Project
  - **More, newer applications**
  - **Community supported (no official Red Hat support)**
  - **For personal system**

## **Linux principles**

- **Everything is a file (including hardware)**
- **Small, single-purpose programs**
- **Ability to chain programs together to perform complex tasks**
- **Avoid captive user interfaces**
- **Configuration data stored in text**

### ***Everything is a file***

UNIX system has many powerful utilities design to create and manipulate files. The UNIX security model is based around the security of files. By treating everything as a file. A consistency emerges. You can secure access to hardware in the same way as you secure access to a document.

### ***Small, single-purpose programs***

UNIX provides many small utilities that perform one task very well. When new functionality is required. The general philosophy is to crate a separate program – rather to extend an existing utility with new features.

### ***Ability to chain programs together to perform complex tasks***

A core design of feature of UNIX is that the output of one program can be the input for another. This gives the user the flexibility to continue many small programs together to perform a larger, more complex task.

### ***Avoid captive user interface***

Interactive commands are rare in UNIX. Most commands expect their options and arguments to be typed on the command line when the command is launched. The commands complete normally, possibly producing output or generates an error message quits. Interactivity is reserved for program where it makes sense, for example text editors (of course. There are non-interactive text editors too.)

### ***Configuration data stored in text***

Text is a universal interface. And many UNIX utilities exist to manipulate text. Storing configuration in text allows an administrator to move a configuration from one machine to another easily. There is several revision control application that enable an administrator to track which change was made on a particular day. And provide the ability to roll back a system configuration to a particular date and time

## UNIT 2

### Linux Usage Basics

#### Logging in to a Linux system

- Two types of login screens: virtual consoles (text-based) and Graphical logins (called display managers)
- Login using login name and password
- Each User has a Home Directory for personal files storage

#### For example:

Local host login: user name

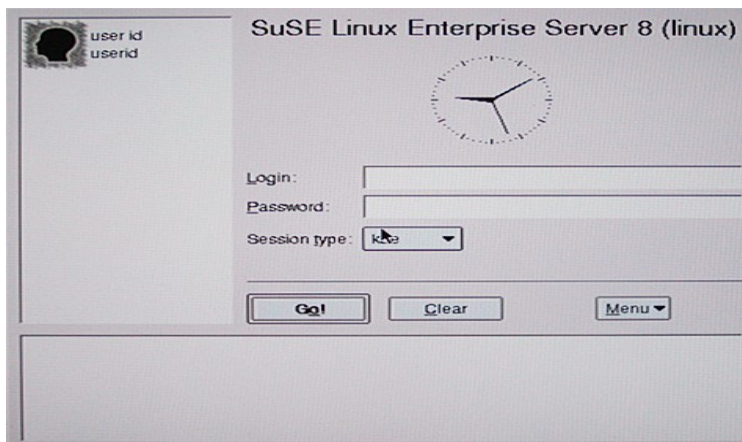
Password:

Last login: Mon Sep 15 19:30:46 On: 0

[bob@localhost bob] #

Notice that your password is not displayed when typed.

#### Graphical logins:



Enter your login user name

After that password

## **Switching between virtual consoles and the Graphical Environment**

- **A typical Linux system will run six virtual consoles and one Graphical console**
  - Server systems often have only virtual consoles
  - Desktop and workstations typically have both
- **Switch among virtual consoles by typing: ctrl-alt-F[1-6]**
- **Access the Graphical console by typing: ctrl-alt-F7**

## **Elements of the X Window system**

- **The X window system is Linux's graphical subsystem**
- **Xorg is the particular version of the X window system used by Red Hat**
  - Open source implementation of X
- **Look and behavior largely controlled by the desktop Environment**
- **Two desktop environments provided by Red Hat:**
  - **GNOME: the default desktop environment**
  - **KDE: an alternate desktop environment**

## **Starting the X server**

- **On some system, the X server starts automatically at Boot time**
- **Otherwise, if system come up in virtual consoles, users must start the X server manually**
  - **The X server Must be pre-configured by the system administrator**
  - **Log into the virtual console and run Start X**
  - **The X server appears on ctrl-Alt-F7**

## Changing your password

- Password control access to the system
  - Changing the Password the first time you log in
  - Change it regularly thereafter
  - Select a password that is hard to guess
- To change your password using GNOME, navigate to system -> preferences-> about me then click password
- To change your password from a terminal: passwd

## The root user

- The root user: a special administrative account
  - Also called the super user
  - Root has near complete control over the system
    - ...and a nearly unlimited capacity to damage it!
- Do not login as root unless necessary
  - Normal (unprivileged) users' potential to do damage is more limited

## Changing Identities

- SU-creates new shell as root
- Sudo-command runs command as root
  - Requires prior configuration by a system-administrator
- Id- shows information on the current user

## Editing text files

- The nano editor
  - Easy to learn and easy to use
  - Not as feature packed as some advanced editors
- Other Editors:
  - gedit, a simple graphical editor
  - vim, an advanced full feature editor
  - gvim, a graphical version of the vim editor

## End of Unit 2

## Lab 2

### Linux Usage and basics

**Goal:** In this lab, we will explore fundamental Linux tasks and commands, such as logging in, changing, and viewing and editing files.

### Sequence 1: logging in and basic Linux Commands

Scenario: begin exploring the use of Linux commands by logging in. changing your password .becoming superuser, using the **Cat** command to view a file and the **nano** command to change a file.

#### **Instructions:**

1. This sequence will run primarily from the first virtual console *tty1*. Switch to the first virtual console by pressing: **Ctrl-Alt-F1**
2. Log in as user student by entering student at the **stationx:** Prompt and pressing return and then entering the password student at the password: prompt.

```
StationX: student
Password :
[student@stationX ~] $
```

Note that the password does not appear on the screen as you type

3. Use the **Passwd** to set your password. The **passwd** command first asks for your current password. Enter redhat (as when logging in. the password will not appear when typed):

```
[student@stationX ~] $ Passwd
Changing password for user student.
Changing password for user student.
(Current) UNIX password
```

4. The **passwd** command checks the strength of the password you enter to ensure that it is sufficiently difficult to guess. Test this by attempting to set a particularly bad password: set the password to your user name student:

```
New UNIX password:
BAD PASSWORD: it is based on your username
New UNIX password:
```

Note that the password is rejected and you are prompted to enter a better password.

5. Try again, this time setting a complex password. Mix upper and lower case characters, use numbers and punctuation, and use at least eight characters. You will be prompted to re-enter the password. If you choose a sufficiently strong password and enter it exactly the same twice, you will succeed in changing your password and the output will look as follows:

```
New UNIX password:  
Retype new UNIX password:  
Paswrd: all authentication tokens updated successfully.
```

**If your password is rejected, keep trying until you successfully change it.**

6. Log out by running the exit command

```
[student@stationX ~] $ exit
```

Log in again using your new password

```
StationX: student  
Password:  
[student@stationX ~] $
```

7. You are now logged in as user *student*, a non-privileged user. For the next part of the lab, you will need superuser privileges to run the commands. Therefore, begin by becoming superuser, using the **su** and providing the password *redhat* when prompted for it. Note that the dash (-) is given as an argument to the **su** command. In a later unit, this will be explained; for now, just be sure to provide the dash:

```
[student@stationX ~] $ Su -  
Password:  
[root@stationX ~] #
```

Note changes to the commands prompt: the username displayed is now *root* and the final character in the prompt is a hash sign instead of a dollar sign. These are two visual indicators that you now have superuser privileges. From this point until you exit from the superuser privileges shell, all commands that you run will have full privileges. This means that it is vital that you double-check every command that you run, as some types of errors can render the system unusable.

8. Use the **passwd** command to change the *student* account password back to *redhat*:



```
[root@stationX ~] passwd student  
Changing password for user student.  
New UNIX password:  
BAD PASSWORD: it is based on a dictionary word  
Retype 'New UNIX password':  
passwd: all authentication tokens updated successfully.
```

9. Why did you need to be root to do this? What happens when you try to set this password as *student*?

10. View the `/etc/issue` file using the **cat** command.

```
[root@stationX ~] # cat /etc/issue  
RED HAT Enterprise Linux server release 5  
Kernel \r on an \m  
[root@stationX ~] #
```

The `/etc/issue` file is displayed above the login prompt. The default connect is shown above (minor variations are possible. Depending on the versions of the operating system and the current configuration) common information to add is a warning to unauthorized Person to keep out.

Note the `\r` and `\m` characters these indicate the release level and machine architecture should be displayed. To confirm this type, type Ctrl-Alt-F2 to see how these escaped characters (the backslash in this case is considered an escape. As it gives a special meaning to the next character) are converted

11. Edit the `/etc/issue` file, using the **nano** command

```
[root@stationX ~] # nano /etc/issue
```

Take a moment to examine the **nano** screen. Note the blinking cursor. Where text will be entered if you start typing. At the bottom of the page is a menu of actions that you can take. The caret (^) character indicates that you need to hold down the Ctrl key to take the action. For example, **Ctrl-X** will cause **nano** to exit

12. Add a new line at the top of `/etc/issue` to make it more friendly:

**Welcome!**

Save the change by typing **Ctrl-X** the **nano** command will ask if you want to save your changes ("Save modified buffer"). Enter Y to save your changes.

The **nano** command will suggest that you save to the `/etc/issue` file. Which is what we intend; press **Enter** to confirm. This will save the file

13. Examine the file again. You should see something like this:

```
[root@stationX ~] # cat /etc/issue
WELCOME!
RED HAT Enterprise Linux server release 5
Kernel \r on an \m
[root@stationX ~] #
```

14. You can now change the effect of your changes. But alterations to /etc/issue do not take effect until your login prompts are restarted. A quick way to force the login prompts to reset themselves is to change to each of the login prompts from **Ctrl-Alt-F2 to Ctrl-Alt-F6** and press **ctrl-d** at each prompt. This will cause the login prompt to terminate and restart. Rereading the /etc/issue file and displayed your new greeting.

15. Clean up:

Return to the first virtual console by typing the Ctrl-Alt-F1.

Type exit to exit out of the superuser shell. Note the change in your prompt.

Type exit again to log out. The login prompt should return, including your new welcome message.

Return to the graphical login by typing Ctrl-Alt-F7

## Unit 3

### Running Commands and Getting Help

Upon completion this unit, you should be able to:

- Execute command at the prompt
- Explain the purpose and usage of some simple commands
- Use the built-in help resources in Red Hat Enterprise Linux

### Running Commands

- Commands have the following syntax:
  - Command options arguments
- Each item is separated by a space
- Options modify a command's behavior
  - Single-letter options usually preceded by:
    - Can be passed as `-a -b -c` or `-abc`
  - Full-word options usually preceded by
    - Example:- **help**
- Arguments are filenames or other data needed by the command
- Multiple commands can be separated by;

For Example:

```
[student@stationX ~] $ mkdir backups; cp *. Txt backups/
```

### Some simple commands

**Date** prints the system time and date the format is configurable via an optional formatting string (see options with **date—help**). The Example below demonstration this feature

- **date** – display date and time
- **cal** – display calendar

For Example:

```
[student@stationX ~] $ Date
Sun may 5 14:57:05 EDT 2002
[student@stationX ~] $ Date +"today is %A , %B, %D, %Y, %nIt, is %r %Z."
Today is Friday, January 06, 2006.
It is 12:05:41 pm, EST.
```

Cal prints an ASCII calendar of the current month. When given a single numeric argument. Cal will give a calendar for the given year. Use a four digit year; however, as the command cal 06 will give a calendar for the year 06, not the year 2006

Given a month and year as arguments, cal will display the calendar for that particular month. For example:

```
[student@stationX ~] $ cal 9 2010
```

September 2010

Su	Mo	Tu	Wed	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	32

Try displaying the calendar for January, 1752

### Getting help

- Don't try to memorize everything !
- Many levels of Helps
  - `whatis`
  - `command-help`
  - `man` and `info`
  - `/user/share/doc/`
  - Red Hat documentation

### The `whatis` Command

- Displays short descriptions of commands
- Uses a database that is up sate nightly
- Often not available immediately after install

```
$ whatis cal
```

```
cal      (1)  - displays a calendar
```

### The `-help` option

- Display usage summary and argument List
- Used by most, but not all, command

```
$ date -help
```

Usage: date [OPTION]... [+FORMATE] or:

```
date [-u|--utc|--universal] [MMDDhhmm[ [CC]YY] [.SS] ]
```

Display the current time in given FORMATE, or set the system date.

...argument list omitted...

## **Reading Usage Summaries**

- Printed by `–help`, `man` and others
- Used to describe the syntax of a command
  - Arguments in `[]` are optional
  - Arguments in CAPS or `<>` are variables
  - Text followed by `...` represents a list
  - `X|Y|Z` means “X or Y or Z”
  - `-abc` means “any mix of `–a` `–b` `–c`”

## **The man Command**

- Provides Documentation for commands
- Almost Every command has a man “page”
- Pages are grouped into “chapters”
- Collectively referred to as the Linux Manual
- **`man [<chapter>] <command>`**

## **Navigating man Pages**

- While viewing the man pages
  - Navigate with arrows, `pgup`, `pgDn`
  - `/text` searches for text
  - `n/N` goes to next/previous match
  - `q` quits
- Searching the Manual
  - `Man –k keyword` lists all matching pages
  - Uses **whatis** database

## **The info Command**

- Similar to `man`, but often more in-depth
- Run `info` without args to list all page
- `info` pages are structured like a web site
  - Each page is divided into “nodes”
  - Link to nodes are preceded by `*`
  - `info [command]`

## **Navigating info pages**

- While viewing an **info** page
  - Navigate with arrows, pgup, pgDn
  - **Tab** move to next link
  - Enter follows the selected link
  - n/p/u goes to the next/previous/up-one node
  - s **text** searches for text (default: last search)
  - q quits **info**

## **Extended Documentation**

- The /usr/share/doc directory
  - Subdirectories for most installed packages
  - Location of docs that do not fit elsewhere
    - Example configuration files
    - HTML/PDF/PS documentation
    - License details

## **Red Hat Documentation**

- Available on docs CD or Red Hat website
  - Installation Guide
  - Deployment Guide
  - Virtualization Guide

## **End of Unit 3**

- Question and answer
- Summary
  - Running Commands
  - Getting Help

## Lab 3

**Goal:** become familiar with the resources available to you for answering questions about redhat enterprise Linux commands

**System setup:** A working installed red hat Enterprise Linux system with an unprivileged user account named student with a password of student

### **Sequence 1:** Using the Help Tools

#### **Instructions:**

1. Look at the command line options for the **man** command. What man options can be used to search name of every manual page for a keyboard and list the matches (the same behavior as **whatis**)

---

2. What man options can be used to search the name and short description of every manual page for a keyboard and list the matching page?

---

3. What **man** option can be used to search the entire text (not just the name and short descriptions) of the manual for a keyboard, displaying pages one at a time?

---

4. Suppose you wanted to view the man pages for the base name function of the C programming language. As apposed to that of the **basename** command. How might you do that?

**HINT:** C functions are discussed in chapter 3 of the manual

---

5. What command line options might you use to cause **Ls** to display a long listing of files with human-readable size description (i.e.6.8M instead of 6819407)?

**HINT:** you will need two command-line options.

---

6. Given the usage description below, which of the following would be a syntactically valid invocation of the command **foo**?

*Foo -x ! -y {-abcde} filename.....*

- a. `foo -x -y one.txt`
- b. `foo`

- c. `foo -y -abc one.txt two.txt`
- d. `foo -abc one.txt two.txt three.txt`

## Sequence 2: Solving problems with man

**Scenario:** you want to change `/etc/issue` again (see previous lab). This time you want to see if there is an escape to make it print the system's hostname.

### Instruction:

1. Begin by consulting the **man** page for `/etc/issue`  
[student@stationx ~] **\$ man issue**

Note that it says escape characters are dependent upon **mingetty**. Note also the reference To *mingetty* (8) in the see also section.

Exit the man page by pressing **q**

2. Continue your search by looking at the **mingetty** man page. Do you need to specify chapter 8 when requesting the page? Why or why not?
- 

3. Jump to the section on escapes by typing  
`/ escape`

Your cursor should be on the heading ISSUE ESCAPES. If it is not, press **n** to move between matches until it is. This section discusses the escape sequences introduced in a previous exercise.

4. Which escape represents the system's hostname?
- 

5. Open `/etc/issue` in **nano**  
[root@stationx ~] **# nano /etc/issue**

6. Change the welcome message you added previously to include the hostname escape

`/etc/issue` should now look similar to this.

```
Welcome to \n!  
Red hat Enterprise Linux server release 5  
Kernel \r on an \m
```

7. Save the file and exit **nano** by pressing **Ctrl-X**
8. Log out by closing your shell with the **exit** command. This should drop you back to a new and improved login prompt! Try accessing other virtual terminals and seeing how the prompt changes. Remember that may need to press enter to see an updated prompt.



## Sequence 1 Solutions

1. **man -f** keyword returns a list of all man pages that contain keyword in the name, which is the same thing that **whatis keyword** would do.
2. **man -k keyword** will list all manual pages that contain keyword. in the name of short description.
3. **man -k keyword** will display every man page that contains keyword. for each matching page the title will be displayed and you will be asked whatever or not you would like to read it
4. **man 3 basename** would display the man page for the basename ( ) function from chapter 3 of the red hat enterprise Linux manual. Just running **man basename** would have displayed the page for the basename command, since it appears before basename ( ) in chapter 1.
5. **Ls -lh filename** would display a long listing (-l) with human-readable (-h) sizes.
6. This usage summary requires exactly one of -x or -y, followed by zero or more of -a, -b, -c, -d, followed by one or filenames. Thus,

foo -y -abc one.txt two.txt

Is the correct form of the command?

## Sequence 2 Solutions

2. You do not need to specify the chapter because in this case chapter 8 contains the first (and only) match for the name **mingetty**. You can verify this by running **whatis mingetty** or by running **man mingetty** and noting the *mingetty* (8) heading in the upper-left.
4. The **\n** escape represents system's hostname.

## Lecture- 4 Linux File System Hierarchy

### Browsing the File system

#### Objectives

Upon completion of this unit, you should be able to:

- Describes important elements of the filesystem hierarchy
- Copy, move, and remove files
- Create and view files
- Manage files with Nautilus

### Linux file Hierarchy Concept

- Files and directories are organized into a single-rooted inverted tree structure
- Filesystem begins at the root directory, represented by a lone / (forward slash) character
- Names are case-sensitive
- Paths are delimited by /

### Some Important Directories

- Home Directories: /root, /home/ username
- User Executables : /bin, /usr /bin, /usr /local /bin
- System Executables: /sbin, /usr/sbin, /usr/local/sbin
- Other Mountpoints: /media, /mnt
- Configuration: /etc
- Temporary Files: /tmp
- Kernels and Boot loader: /boot
- Server Data: /var, /srv
- System Information: /proc, /sys
- Shared Libraries: /lib, /usr/lib, /usr/lib/usr/local/lib

### Current Working Directory

- Each shell and system process has a current working directory (CWD)

- **pwd**
  - Displays the absolute path to the shell's cwd

For example. [student@stationX games] \$ pwd

/user/local/games

## File and Directory Names

- Names may be up to 255 characters
- All characters are valid, except the forward-slash
  - It may be unwise to use certain special characters in file or Directory names
  - Some characters should be protected with quotes when referencing them
- Names are case-sensitive
  - Example: MAIL, Mail, mail, and mAiL
  - Again, possible, but may not be wise

To access a file whose name contains special characters, enclose the filename in quotes.  
For example:


```
[student@stationX ~] $ ls -l "file name with spaces.txt"
-Rw-rw-r-- 1 student student 0 Dec 14 21:48 file name with spaces.txt
```

## Absolute and Relative Pathnames

- Absolute pathnames
  - Begin with a forward slash
  - Complete "road map" to the location
  - Can be used anytime you wish to specify a file name
- Relative Pathnames
  - Do not begin with a slash
  - Specify location relative to your current working directory
  - Can be used as a shorter way to specify a file name

Some example of relative pathnames. Relative to particular directories. Follow. In each case, the file being referenced is /user/share/doc/HTML/index.html.

Current directories	relative path to index.html
/usr/share/doc/HTML	index.html
/usr/share/doc/	HTML/index.html
/usr/share/	doc/HTML/index.html
/usr/	share/doc/HTML/index.html
/	usr/share/doc/HTML/index/html



/usr/share/doc/HTML/en/                      ../index.html  
/usr/share/doc/nautilus-2.1.91/              ../HTML/index.html

## Changing Directories

- **cd** change directories
  - To an absolute and relative path:
    - **cd** /home/joshua/work
    - **cd** project /docs
  - To a directory one level up:
    - **cd** ..
  - To your home directory
    - **cd** ~
  - To your previous working directory:
    - **cd** -

## Listing Directory Contents

- Lists the contents of the current directory or a specified directory
  - Usage:
    - **ls** [**options**] [**files\_or\_dirs**]
- Ls without arguments list the file and directory names in the current directory.

[student@stationX ~] \$ **ls** work

- Example:
  - **ls -a** (include hidden files)
  - **ls -l** (display extra information)
  - **ls -R** (recurse through directories)
  - **ls -ld** (directory and symlink information)

### **For example:**

1. **ls -a** includes so-called “hidden” files and directories whose names begin with a dot.

```
[student@stationX ~] $ ls -a
.bash_history      .bash_loguot      .bash_profile     work
```

Ls list another file or directory if given as an argument:

```
[student@stationX ~] $ ls /
```

```
Bin  dev  home  lib  misc  opt  root  tftpboot  usr
Boot  etc  initrd  lost+found  mnt  proc  sbin  tmp  var
```

2. Use `ls -l` for a more detailed “long” listing:

```
[student@stationX ~] $ ls -l /usr
```

Total 204

```
drwxr-xr-x  2 root  root  61440 jul 29 10:07 bin
drwxr-xr-x  2 root  root   4096 feb  6 1996 dict
drwxr-xr-x  3 root  root   4096 jul 29 09:00 doc
drwxr-xr-x  2 root  root   4096 feb  6 1996 etc
```

.... Output truncated...

3. `ls -R` recurses through subdirectories. Listing their contents too. `ls -d` directory names, not their contents. It has no effect when filename are passed as arguments. This option is also useful with `-l`:

```
[student@stationX ~] $ ls -ld /usr
```

```
Drwxr-xr-x 18 root  root  4096 feb  24 11:36 /usr
```

The `ls` command has many other options. All options can be used in combination with other `ls` options

## Copying Files and Directories

- **cp** –copy file and directories
- Usage:
  - **cp [options] file destination**
  - More than one file may be copied at a time if the destination is a directory:
    - **cp [options] file1 file2 destination**

A few common options include:

**-I** (interactive): ask before overwriting a file  
**-r** (recursive): recursively copy an entire directory tree  
**-p** (preserve): preserve permission, ownership, and time stamps  
**-a** (archive): copies file and directories recursively (like `-r`) while preserving permission (like `-p`)

Example:

```
[student@stationX ~] $ ls /home/student
testfile
```

```
[student@stationX ~] $ cp ~student /testfile /tmp/student_test_file
[student@stationX ~] $ ls /tmp
```

Student\_test\_file

## Moving and Renaming Files and Directories

- **mv**- move and/or rename files and directories
- Usage:
  - **mv [options] file destination**
- More than one file may be moved at a time if the destination is a directory
  - **mv [options] file1 file2 destination**
- Destination work like **cp**

For Example:

```
[student@stationX ~] $ ls ~student
Testfile
[student@stationX ~] $ mv ~student/testfile /tmp/student_test_file
[student@stationX ~] $ ls ~student
[student@stationX ~] $ ls /tmp/
Student_test_file
.....other output omitted....
```

## Creating and Removing Files

- **touch** – create empty files or update file timestamps
- **rm** – remove files
- Usage:
  - **rm -l [options] <file>...**
- Example:
  - **rm -l file (interactive)**
  - **rm -r directory (recursive)**
  - **rm -f file (force)**

## Creating and Removing Directories

- **mkdir** -creates directories
- **rmdir** -remove empty directories
- **rm -r** -recursively removes directory trees

Directories are created with **mkdir**. For example, to create a directory called work:

```
[student@stationX ~] $ mkdir work
```

To remove an empty directory. Use **rmdir work** for example:

```
[student@stationX ~] $ rmdir work
```

Rmdir will only remove empty directories. To remove a directory and its contents, use **rm -r**

## Using Nautilus

- **Gnome graphical file system browser**
- **Can run in spatial or browser mode**
- **Accessed via...**
  - **Desktop icons**
    - Home: your home directory
    - Computer: Root file system, network resources and removable media
  - **Applications -> System tools-> file browser**

## Moving and Coping in Nautilus

- Drag and Drop
  - Drag: move on same file system, copy on different file system
  - Drag +ctrl: always copy
  - Drag +alt: ask whether to copy, move or create symbolic link (alias)
  - Context menu
    - Right-click to rename, cut, copy or paste

## Lab 4

### Browsing the filesystem

**Goal:** become familiar with the functions, syntax and use of several essential file and directory manipulation commands.

**System Setup:** A working, installed Red Hat Enterprise Linux system with an unprivileged user account named student with a password of student.

### **Sequence 1: Directory and file Organization**

**Scenario:** A few files have accumulated in your home directory, and you have decided that it is time to organize things. You plan to create several new subdirectories, and to copy and move your files around to fit in to your new scheme. Additionally, you have several files that are not are not needed at all. Which must be deleted.

**Deliverable:** A more organized home directory, with files placed in to the appropriate

Subdirectories.

### Instructions:

1. Log in as user student with password student. If you are using the graphical environment, start a terminal by clicking Applications->Accessories->terminal
2. Immediately, after logging into the system, you should be in your home directory. Verify this using the ‘print working directory’ command.

```
[student@stationX ~] $ Pwd  
/home/student
```

3. Check to see if you have any files in your home directory using the following commands:

**Ls**

**Ls -a**

**Ls -al**

Why do the first and second commands return different numbers of the files?

What is the file of the largest file currently in your home directory as reported by the third command?

4. You will now use touch to create the files needed for this sequence. The details of how the expansion used in the following command works will be covered in a later unit. For now, simply type the following line exactly as you see it (with the curly braces { } included, and an underscore between the first two groups of sets):

```
[student@stationX ~] $ touch {report, graph} _{jan, feb.mar}
```

5. Use the ls command and examine the result of the last command. You should find that the following six new, empty file in your home directory.

```
[student@stationX ~] $ ls  
Graph_feb graph_jan graph_mar report_feb report_jan report-mar
```

These files represent the data files that you will use in the remainder of this sequence. If for some reason you do not see these files, asks the instructor for assistance; without these files, the remainder of this will not work.

6. In order to organize your files you must first create some new directories, below be sure to check that your working directory is as expected.

```
[student@stationX ~] $ ls mkdir projects
```



```
[student@stationX ~] $ ls mkdir projects/graphs
```

```
[student@stationX ~] $ ls cd projects
```

```
[student@stationX ~] $ mkdir reports
```

```
[student@stationX ~] $ cd reports
```

```
[student@stationX ~] $ mkdir ../backups
```

Use **ls** to examine your work

```
[student@stationX reports] $ cd
```

```
[student@stationX ~] $ ls -l
```

Total 4

```
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_feb
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_jan
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_mar
Drwxrwxr-x 5 student student 4096 sep 30 21:09 projects
-rw-rw-r-- 1 student student 0 sep 30 21:08 report_feb
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_jan
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_mar
```

```
[student@stationX ~] $ ls projects
```

Backups graphs reports

7. Begin by moving all of the graph files into the graphs subdirectory of the projects directory. Do this in two steps: in the step, move one file; in the second step, move two files:

```
[student@stationX ~] $ mv graph_jan projects/graphs
```

```
[student@stationX ~] $ mv graph_feb graph_mar projects /graphs
```

```
[student@stationX ~] $ ls -l projects /graphs/
```

```
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_feb
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_jan
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_mar
```

8. Next, move two of the “**report**” files into the reports subdirectory of the projects directory. Move the files in one command.

```
[student@stationX ~] $ mv report_jan report_feb projects/reports
```

```
[student@stationX ~] $ ls -l projects/reports
```

```
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_feb
-rw-rw-r-- 1 student student 0 sep 30 21:08 graph_jan
```

9. Remove the remaining report file:

```
[student@stationX ~] $ rm report_mar
```

```
[student@stationX ~] $ ls
```

## Projects

10. Change into the backup's directory and copy the January files into this directory. Copy one using an absolute pathname and the other using a relative pathname:

```
[student@stationX ~] $ cd projects/backups
[student@stationX backups] $ pwd
/home/student/projects/backups
[student@stationX backups] $ cp ../home/student/projects/graphs/graph_jan.
[student@stationX backups] $ ls -l
Total 2
```

```
-rw-rw-r-- 1 student student      0 sep 30 21:08 graph_jan
-rw-rw-r-- 1 student student      0 sep 30 21:08 report_jan
```

The trailing dot is the destination: the present working directory.

11. Log out or close your graphical terminal by running the exit command

## Sequence 2: managing files with Nautilus

**Scenario:** A co-worker, bob would like to see a copy of your graphs. You will take advantage of the /tmp directory to make these files available to him using nautilus. The instructions below use Nautilus' "spatial" mode and the copy/paste method for copying files. Nautilus is a very flexible tool and there are multiple ways to approach most problems. If you have time left over after completing this sequence, try discovering some of these other methods. For example you can try doing the sequence again using nautilus; 'browser interface to see which you prefer. To launch the nautilus browser, select filesystem Browser from the Application menu. Be sure to take advantage of the file system tree feature if you do this (select tree from the dropdown in the left slide bar).

**Deliverable:** Bob can now access copies of your graphs by retrieving them from **/tmp/stuff\_for\_Bob**

### Instruction:

1. You have not already, log into the graphical environment.
2. Double-click on the student's home directory icon on your desktop. This will open a nautilus window displaying your home directory.

3. Double click on **projects**, then **graphs**. Note that each directory opens in a new window.
4. With the **graph** window selected, press **Ctrl-shift-w** to close the parent directory window.
5. Press **Ctrl-a** to select all three graph.
6. Press Ctrl-c to copy the files. A message will be displayed at the bottom of the windows informing you that three items have been copied.
7. Press **Ctrl-l** (the letter l, not the number 1) to display at the open location dialog. Type / **tmp** (note that you can use tab- competition) and press enter to open a new window displaying **/tmp**.
8. Press **Ctrl-shift-n** or right click on window background and select create Folder to create a new directory.
9. Type in **stuff\_for\_Bob** as the new directory's name and double-click the directory to open it
10. Press **Ctrl-V** to paste your graph into **/tmp/stuff\_for\_Bob**.
11. press-q to close all nautilus window

### Sequence 3: Backing-up your system configuration

**Scenario:** To prevent the loss of important configuration files. You have decided to create a backup of the **/etc/sysconfig** directory. Note that **/etc/sysconfig** is not the only directory that houses important configuration files. But because it will take less time then backing to everything. We will content to ourselves with it for this exercise.

**Deliverable:** A backup copy of your **//etc/sysconfig** directory, with the original attributes and owners

#### Instruction:

1. Log in on tty1 as user root with password of redhat.
2. In root's home directory create a directory named backups.

```
[roo@stationx ~] # mkdir ~/backups
```

3. Devise a cp command to recursively copy **/etc/sysconfig** to a subdirectory of **~/backups** in a way that meets the following requirements:

- The permission and ownership of files being copied must be preserved. Remember that cp does not do this by default.
- To avoid conflicting with later backups, files should be copy to a directory called **~/backups/sysconfig-yyyymmdd**, where **yyyymmdd** is a datestamp. For example 20071231 if it is the 31<sup>st</sup> of December, 2007. Because cp does not have an option for doing this automatically, you will need to type in the datestamp yourself. This technique will be improved upon in later labs.
- The name of each file should be printed out as it is copied.

You can consult the solutions section for this exercise to check your answer

3. When you are confident that you have the right command line. Execute it to create your backup.
4. Why is using a unique destination so important when copying directories? What would happen if you ran the command you devised for the previous step twice in a row?
5. What about a third time?

### **Sequence 1 Solutions**

3. `ls` returns fewer files than `ls -a` option includes files whose names begin with a period such files are usually used for storing configuration information and are not included in directory listing by default.

The fifth columns of `ls -l`'s output displays the file's size

### **Sequence 3 solution**

3. Long answer (assuming the 31<sup>st</sup> of December, 2007, for the date):

```
cp-rpv /etc/sysconfig ~/backups/sysconfig - 20071231
```

**Shorter answer:**

```
Cp -av /etc/sysconfig ~/backups/sysconfig - 20071231
```

5. The first time the command is run, the destination directory, `~/backups/sysconfig - 20071231` will be created and the files in `/etc/sysconfig-bak/sysconfig/` will be copied in to it

The second time, because the destination directories already exist, `/etc/sysconfig/~/backups/sysconfig-bak/sysconfig/`.

6. if the command was run a third time, the files would again be copied to `~/backups/sysconfig-bak/sysconfig` and newer files would overwrite the older ones. If u were logged in as root, you would be prompted for confirmation before overwriting. For anyone else the files would be overwritten without confirmation.

## Unit 5

### Users, Groups and permissions

#### Users

- Every user is assigned a unique user ID number (UID)
  - UID 0 identifies root
- Users' names and UIDs are stored in /etc/passwd
- Users are assigned a home directory and a program that is when they log in (usually a shell)
- Users cannot read, write or execute each others' files without permission

#### **Groups**

- Users are assigned to groups
- Each group is assigned a unique group ID number (gid)
- GIDs are stored in /etc/group
- Each user is given their own private group
  - Can be added to other groups for additional access
- All users in a group can share files that belong to the group

#### **Groups:**

Sometimes users need to collaborate. This can be accomplished by having users assigned to groups and setting appropriate group permission for files or directories. Every user is a member. Gids. The group names and GIDs are stored in the */etc/group file*

#### **User private Group Scheme:**

By default a user belongs to a group that is named the same as their username. That is user Digby is a member of group Digby and, by default is the only member of that group. This system can be abandoned by system administrators when they set up accounts and so this may not be the case at your location.

## Primary Group:

A user's primary group is defined in the `/etc/passwd` file and secondary groups are defined in the `/etc/group` file the primary group is important because files created by the user will inherit that group affiliation. The primary group can temporarily be changed by running ***newgrp groupname***, where group name is one of the user's secondary groups. The user can return to their original group by typing **exit**.

## Linux File Security:

- Every file is owned by a UID and a GID
- Every process runs as a UID and one or more GIDs
  - Usually determined by who run process
- Three access categories:
  - Processes running with the same UID as the file (user)
  - Processes running with the same GID as the file (group)
  - All other processes (other)

## Permission Precedence:

- **If UID matches, user permission apply**
- **Otherwise, if GID matches, group permission apply**
- **If neither match, other permission apply**

## Permission Types:

- Four symbols are used when displaying permissions
  - `r`: permission to read a file or list a directory contents
  - `w`: permission to write to a file or create and remove files from a directory
  - `X`: permission to execute a program or change into directory and do a long listing of the directory
  - `-`: no permission (in place of the *r*, *w*, or *x*)

## Examining Permissions:

- File permission may be viewed using `ls -l`  
`$ ls -l /bin/login`  
`-rwxr-xr-x 1 root root 19080 apr 1 18:26 /bin/login`
  - File type and permission represented by a 10-character string

## Interpreting Permissions:

- `-rwxr-x --- 1 andersen trusted 2948 oct 11 14:07 myscript`
  - Read, write and execute for the owner, andersen
  - Read and execute for members of the *trusted* group
  - No access for all others

## *Interpreting permission*

The file shown at the top of this slide can be read, written and executed (rwx) by the user Anderson, read and executed (r – x ) by member of the group trusted and has grants no permissions (---) to anyone else. Some more examples are shown below.

Given the following file permission:

```
-rwxr-xr--    1 fred   fred   26807   mar  8  22:55  penguin
-rw-r--r--    1 mary   admin  1601   mar  5  22:36  redhat
-rw-rw-r--    1 root    staff  8671   mar  8  22:59  tuxedo
```

And the following file permission:

User	Primary group	Secondary Group
fred	fred	staff
mary	mary	staff.admin

The following will be true:

- Penguin can be read, written and executed by fred, but only red by mary;
- Redhat can be read and written by mary, but only read by fred :
- Tuxedo can be read and written by both Mary and Fred.

## **Changing File Ownership:**

- Only root can change a file's Owner
- Only root or the owner can change a file's group
- Ownership is changed with chown:
  - Chown [-R] user\_name file\directory
- Group-Ownership is changed with chgrp:
  - chgrp [-r] group\_name file\directory

File ownership can be changed with the **chown** command. For example. To grant ownership of this file foofile to student, the following command could be used:

```
[root@station ~] # chown student foofile
```

Chown can be used with the **-R** option to recursively change ownership of an entire directory tree. The following command would grant ownership of foodir and all files and subdirectories within it to student

```
[root@station ~] # chown -R student foodir
```

Only root can change the ownership of a file. Group ownership, however can be set by root or this file's owner. Root can grant ownership to any group. While non-root users can grant ownership only to groups they belong to. Changing the group ownership of a file is done with **the chgrp** command. The syntax is identical to that of chown. Including the use of **-r** to affect entire directory trees.

### Changing permissions – symbolic Method:

- To change access modes:
  - **Chmod [-R] mode file**
- Where *mode* is:
  - U,g or o for user, group and other
  - + or – for grant or deny
  - r,w or x for read, write and execute
- Examples:
  - Ugo+r: grant access to all
  - o-wx: deny write and execute to others

**For example:**

```
[root@station ~] # chmod u+r filename
```

```
[root@station ~] # chmod g+rw filename
```

```
[root@station ~] # chmod o+x filename
```

### Changing Permission - Numeric Method:

- Uses a three-digit mode number
  - First digit specified owner's permissions
  - Second digit specified group's permissions
  - Third digit represented other's permissions
- Permissions are calculated by adding:
  - 4(for read)
  - 2 (write)
  - 1 (for execute)
- Example:
  - **chmod 640 myfile**

Numerical mode setting



To change all the permission on a file at once, it is often easier and quick to use the numeric method.

In this method, the first argument to the chmod command is a three digit number representing the permission that are set. In this method, the read permission has value of four, the write permission has a value of two, and the execute permission has a value of one. Add together the permission you wish to set for user, group, and others.

### Examples:

**Chmod 664 file** grants read/write to the owner and groups, read-only to others.

**Chmod 660 file** grants read/write to the owner and groups; no permission for others

**Chmod 600 file:** grants read/write to the owner; no permission set for the group and others

**Chmod 444 file:** grants read only to all

With directory permission when you set the read permission. You almost always want to set execute permission.

Example:

**Chmod 755 dir.** Grants full permission for the owner read and execute to group and other.

**Chmod 770 dir.** Grants full permission to the owner and group, no permission to others

**Chmod 700 dir.** Grants full permission to the owner. No permission to the group or others

**Chmod 555 dir.** Grants read and execute permission set to all.

### Changing Permissions – Nautilus:

- Nautilus can be used to set the permissions and group membership of files and directories.
  - In a Nautilus window, right click on a file
  - Select properties from the context menu
  - Select the **permission** tab

### End of Unit 5:

- Question and Answer
- Summary
  - All files are owned by one user and one group
  - The mode of a file is made up of three permissions: those of the user, the group and all other
  - Three permissions may be granted or denied: read, write and execute

## Lab 5

### File permission

---

**Goal:** become familiar with the functions, syntax, and use of several essential file permission modifications command. And how you can combine these commands together in useful ways to accomplish common user tasks.

**System setup:** A working, installed red hat enterprise Linux system with an unprivileged User account named student with a password of student

### **Sequence 1** **determining file permission**

#### **Instructions:**

1. What is the symbolic representation (e.g. `rwxr-xr-x`) of each of the following numeric permission?

644 -----  
755 -----  
000 -----  
711-----  
700-----  
777-----  
555-----  
111-----  
600-----  
731-----

2. Given a file with permission 755, what commands would change the permission to `r-xr--r--`

---

—

3. You have been just downloaded a file that you know is trustworthy from the internet. And you want to execute it. What step must you perform before you can run it, list two different ways to perform that stem?

---

—

4. How could root change the ownership of a file so that it is associated with the user *joi* and group *apache*?

## Sequence 1 Solution:

1. **644** `rw-r--r--`  
**755** `rxr-xr-x`  
**000** `-----`  
**700** `rx-----`  
**777** `rxrxrxrx`  
**555** `r-xr-xr-x`  
**111** `-x--x--x`  
**600** `rw-----`  
**731** `rx-wx--x`

2. there are several possible solutions:

**Chmod** 544 filename

**Chmod** u-w,go-x filename

**Chmod** u=rx,go=r filename

3. The command must be made executable. Both one of the following commands will accomplish this (and other commands will also work):

**Chmod +x cmdname**

**Chmode 755 cmdname**

4. `chown Joe filename ; chgrp apache filename`

Or

**Chown Joe: apache filename**

## Unit 6

### Using the bash Shell

### Objectives

Upon completion of this unit, you should be able to:

- Use command-line shortcuts
- Use command-line expansion
- Use History and editing tricks
- Use the **gnome-terminal**

#### **Command Line Shortcuts File Globing:**

- **Globing is wildcard expansion:**
  - **\*** - matches zero or more character
  - **?** – matches any single character
  - **[0-9]** –matches a range of numbers
  - **[abc]** – matches any of the character in the list
  - **[^abc]**- matched all except the character in the list
  - **Predefined character classes** can be used

#### **Typing the following command:**

```
[student@stationx ~] $ rm *mp3
```

Is the same as typing:

```
[student@stationx ~] $ rm gonk.mp3 zonk.mp3
```

The result is that all files in the directory that have names ending in mp3 (in the case, just the two listed) will be removed.

**Echo** can be used to test the expansion of metacharacters before using them in a destructive command like **rm**:

```
[student@stationx ~] $ echo ?o*  
Joshua.txt gonk.mp3 zonkmp3
```

Uses of character ranges, as those used for numbers are considered harmful. They are not portable and may result in unexpected list after expansion.

You would rather use character classes. Which are also used in other applications, and only in bash.

The syntax for a character class is. [: keyword:], where keyword can be: alpha, upper, lower, digit, alnum, punct, space

If you want to match only numbers, you might use:

**[[:digit:]]**

If you want to match anything but alphanumeric characters:

**[A[:alnum:]]**

## Command Line Shortcuts the *Tab* key:

- Type *Tab* to complete command lines:
  - For the command name, it will complete a command name
  - For an argument, it will complete a file name
- Examples:

```
$ xte<tab>
```

```
$ xterm
```

```
$ ls myf<tab>
```

```
$ ls myfile.txt
```

## Command Line Shortcuts History:

- **bash** stores a history of commands you've entered, which can be used to repeat commands
- Use **history** command to see list of "remembered" commands

```
$ history
```

```
14 cd /tmp
```

```
15 ls -l
```

```
16 cd
```

```
17 cp /etc/passwd
```

```
18 vi passwd
```

```
...output truncated
```

## More History Tricks:

- Use the up and down keys to scroll through previous commands
- Type ctrl-r to search for a command in command history

- (reverse-i-search)
- To recall last argument from previous command:
  - Esc:- (the escape key followed by a period)
  - Alt:- (hold down the alt key while pressing the period)

## History:

Using your history is great productivity-enhancing tool. Linux users who develop a habit of using their history can streamline and speed their use of the shell. Try playing with keystrokes listed above.

You can ignore repeated duplicate commands and speed their use of the shell. Try playing with the keystrokes listed above.

You can ignore repeated duplicate commands and repeated lines that only differ in repeated spaces by adding the following to your .bashc.

```
[student@stationX ~] export histcontrol=ignoreboth
```

## Command Line Expansion the Tilde:

- Tilde ( ~ )
  - May refer to your home directory
- ```
$ cat ~/.bash_profile
```
- May refer to another user's home directory
- ```
$ ls ~julie/public_html
```

## Command Line Expansion Commands and braced sets:

- Command Expansion: \$ ( ) or ``
    - Prints Output of one command as an argument to another

```
$ echo "this system's name is $ ( hostname )"
This system's name is server1.example.com
```
  - Brace Expansion: { }
- ```
$ echo file {1,3,5}
File1 file3 file5
$ rm -f file {1,3,5}
```

The bash shell has some special features relating to expansion which significantly improve the power of working at the command line.

Curly braces are useful for generating patterned strings. For Example without the curly braces, the **mkdir** command below would take almost two hundred keystrokes to execute.

```
[student@stationX ~] $ mkdir -p work/ {inbox, outbox, pending} /  
{normal,urgent.important}
```

Use of the backquotas is called command substitution. In command substitution. The command in backquotas is executed and the output of the command is placed on the command line as if the user had typed it in. an alternative syntax for the backquotas is to place the command in parentheses preceded by a dollar sign \$ ( ).

### **Command Editing Tricks:**

- **Ctrl-s** moves to beginning of line
- **Ctrl-e** moves to end of the line
- **Ctrl-u** deletes to beginning of line
- **Ctrl-k** delete to end of line
- **Ctrl-arrow** moves left or right by word

### **Gnome-Terminal:**

- Applications◇ accessories◇ terminal
- Graphical terminal emulator that support multiple “tabbed” shells
  - Ctrl-sift-t creates a new tab
  - Ctrl-pgUp/pgDn switches to next/prev tab
  - Ctrl-sift-c copies selected text
  - Ctrl-sift-v pastes text to the prompt

### **Scripting Basics:**

- Shell script are text files that contain a series of commands or statements to be executed
- Shell scripts are useful for:
  - Automating commonly used commands
  - Performing system administration and troubleshooting
  - Creating simple applications
  - Manipulation of text or files

### **Creating Shell scripts:**

- Step 1: use such as vi to create a text file containing commands
  - First line contain the magic shebang sequence: # !
  - #!/bin/bash
- Comments your scripts !
  - Comments start with a #

Shell script are text file that generally contain one command per line, but you can have multiple commands on a line if you separate them with semicolons (;) in order to continue command on to the next line you use the line continuation character. For the

Bourne shell (/bin/sh) and its derived shells such as bash, this is a backslash followed by a new line. You can enter this by pressing the \ key followed by the enter key on most keyboards. This will enable you to enter one command that spans multiple lines.

The first line a shell scripts should contain 'magic' which is commonly referred to as the shebang. This tells the operating system which interpreter to use in order to execute the script. Some examples are.

Shebang use

- **#!/bin/bash**-used for bash scripts (most common on Linux)
- **#!/bin/sh**-used for Bourne shell scripts (common on all UNIX-like systems)
- **#!/bin/csh**-used for C shell scripts (common on BSD derived systems)
- **#!/user/bin/perl**-used for perl (an advanced scripting and programming language)
- **#!/user/bin/python**-used for python scripts (an object oriented programming language)

Commenting shell scripts

It is extremely important to put comments in your shell scripts. Anything following the # symbol is comment and therefore ignored by the interpreter. It is good practice to make a shell script self documenting. Self documenting. Means that someone with almost no scripting knowledge can read your comments in the script and reasonably understand what the script does. The easiest way to do this is to write the comments. Before you actually write the code. This has an additional benefit of clarifying to yourself, what your objective is. In addition, this practice makes the script easier to maintain for both yourself and others. As time progresses, you may not recall what you were trying to do at the time and comments may help clarify the original objective.

## Creating Shell Scripts:

- Step 2: Make the scripts Executable  
\$ chmod u+x myscript. Sh
- To execute the new script:
  - Place the script file in a directory in the executable path –OR–
  - Specify the absolute or relative path to the script on the command line

An example of making a script you own executable:

```
[student@stationX ~] $ Chmod u+x scriptfile
```

Or

```
Chmod 750 scriptfile
```



Ensure that the script is located in a directory listed by the PATH environment variable. To do this. Enter the following command:

```
[student@stationX ~] $ echo #PATH
```

If the script is not in a directory listed in the PATH variable. Either move the script to a directory that is (such as \$HOME/bin) or specify the absolute or relative path on the command line when executing the script:

```
[student@stationX ~] $ /home/user/mytestscript
```

Or

```
[student@stationX ~] $. /mytestscript
```

You can create a bin directory in your directory and store your own script here. Since this directory is normally added in your PATH environment variable. You can run any of your scripts without having to type in the full path.

### **Sample Shell Script:**

```
#!/bin/bash
```

```
# This script display some information about your environment
```

```
Echo "Greetings. The date and time are $(date)".
```

```
Echo "your working directory is: $(pwd)"
```

### **End of Unit 6:**

- Question and answer
- Summary
  - Command expansion:\$ ( )
  - History recall: !string, !num
  - Inhibition: “,\

## **Lab 6**

### **Exploring the bash Shell**

**Goal:** Become familiar with the functions, syntax and use of several essential file and directory manipulation commands. Practice combining these commands together in useful ways to accomplish common user tasks.

**System Setup:** A working, installed red hat Enterprise Linux system with an unprivileged user account named student with a password of student.

### **Sequence 1: Directory and file Organization**

**Scenario:** Once again files have managed to accumulate in your home directory and you have decided that it is time to organize things. This time you will use your knowledge of the bash shell to perform more complex file management tasks.

**Deliverable:** A more organized home directory, with files placed into the appropriate Subdirectories, and some files backed up to /tmp/archive.

**Instructions:**

1. Log in as user student with the password student. If you are the graphical environment, start a terminal by clicking Application->Accessories->terminal
2. Immediately after logging into the system, you should be in your home directory. Verify this with **pwd**.

```
[student@stationx ~] $ Pwd  
/home/student
```

Note that you can also tell you are in your home directory by noting the ~ in your command prompt.

3. You will now use touch to create the files needed for this sequence. The details of how the expansion used in the following command works will be covered in a later Unit. For now, simply type the following line exactly as you see it (with the curly braces { } included, and an underscore character between the first few groups of sets). Have another nearby student or the instructor verifies the accuracy of your command before you press enter:

```
[student@stationx ~] $ touch  
{report,memo,graph}_{Sep,Oct,nov,dec}_{a,b,c}_{1,2,3}
```

4. Use the **ls** command to examine the results of the last command. You should find that it created 108 new, empty files (you do not need to count) in your home directory. These file represent the data files that you will use in the remainder of this sequence. If for some reason you do not see these files, ask the instructors for assistance; without these files, the remainder of this lab will not work.
5. In order to organize your files you must first create new directories. Use **mkdir** to create some subdirectories directly inside your home directory:

```
[student@stationx ~] $ mkdir a_reports  
[student@stationx ~] $ mkdir September October December
```

Again, use **ls** to examine your work

6. Create some additional subdirectories inside one of your new directories using the following commands.

```
[student@stationx ~] $ cd a_reports
```

To change to the directory. Then:

```
[student@stationor,; a_reports] $ mkdir one two three
```

Use ls to verify that you have three new directories named one, two, and three under you're a\_reports subdirectory.

7. Begin by moving all of the "b" reports out of your home directory and grouping them by month. When working with accomplished wildcard patterns. It is a good idea to pre-verify the operation to ensure you are operating on the correct files. One way to do this is to replace your command with a harmless command using the intended wildcard pattern.

```
[student@stationx a_reports] $ cd
```

```
[student@stationx ~] $ ls -l *dec_b_?
```

Use should see the 9 "December", b" files listed. Move one of them to the December directory:

```
[student@stationX ~] $ mv graph_dec_b_?december/
```

Now move the reset of them with:

```
[student@stationX ~] $ mv *dec_b_? December/
```

List the contents of the December subdirectory to verify the move operation was successful:

```
[student@stationX ~] $ ls -l December/
```

Total 9

|            |   |         |         |   |     |    |       |                |
|------------|---|---------|---------|---|-----|----|-------|----------------|
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | graph_dec_b_1  |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | graph_dec_b_2  |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | graph_dec_b_3  |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | memo_dec_b_1   |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | memo_dec_b_2   |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | memo_dec_b_3   |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | report_dec_b_1 |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | report_dec_b_2 |
| -rw-rw-r-- | 1 | student | student | 0 | Oct | 16 | 22:17 | report_dec_b_3 |

8. Move all of the remaining "b" reports into their respective directories

```
[student@stationX ~] $ mv *nov_b_? November/
```

```
[student@stationX ~] $ mv *Oct_b_? October/
```

```
[student@stationX ~] $ mv *sep_b_? September/
```

9. You will now collect the ‘a’ reports into their respective corresponding numbered directories. Notice the use of ~ as shorthand for ‘ your home directory’ the combination of the wildcard and the pattern specifies all files that end in\_a1 in your home directory.

```
[student@stationX ~] $ cd a_reports  
[student@stationX a_reports] $ mv ~/*_a_1 one/
```

The ‘ September’ al’ files are old and no longer needed. Use ls to make sure you have created a pattern that matches only these file, then delete them, and verify that the other ‘al’ files were moved properly

```
[student@stationX a_reports] $ cd one  
[student@stationX one] $ ls *sep*  
[student@stationX one] $ rm *Sep*  
[student@stationX one] $ ls  
Report_dec_a_1 graph_oct_a_1 memo_no_a_1 report_dec_a_1  
Report_oct_a_1 graph_nov_a_1 memo_dec_a_1 memo_oct_a_1  
Report_nov_a_1
```

9. Move the final ‘a\_2’ and ‘a\_3’ reports into their respective directories. To make life interesting, we will move them from the current directory, using both relative and absolute pathnames. First, use pwd to identify the current directory.

```
[student@stationX one] $ pwd  
/home/student/a_reports/one
```

Verify the pattern that references the ‘a\_2’ files with ls, then move them using absolute pathnames:

```
[student@stationX one] $ ls /home/student/*a_2*  
[student@stationX one] $ mv /home/student/*a_2/home/student/a_reports/two/
```

Even though your current directory is /home/student/a\_reports/one, you can move files from /home/student to /home/student/a\_reports/two because you specified the files’ using relative pathnames – in this case, absolute pathnames.

Now move the ‘a\_3’ files using relative pathnames. Again. First verify the pattern references the correct files.

```
[student@stationX one] $ ls ../a_3*  
[student@stationX one] $ mv ../a_3* ../three/
```

10. Return to your home directory, and use ls to verify that the only files remaining in this directory are the ‘c’ files (i.e.. graph\_dec\_c\_1, graph\_dec\_c\_2, ...)

11. the ‘c1’ and ‘c2’ report files for each month are important, and you want to make backup of them in another directory:

```
[student@stationX ~] $ mkdir /tmp/archive  
[student@stationX ~] $ cp reports*{1,2} /tmp/archive/
```

Additional, all the report files for the month of December should be backed up to the /tmp/archive directory. Note the use of the **-I** option to have **cp** prompt before overwriting any files.

```
[student@stationX ~] $ cp -I report_dec* /tmp/archive/  
Cp: overwrite ./tmp/archive/report_dec_c_1,? n  
Cp: overwrite ./tmp/archive/report_dec_c_2,? n
```

12. Now that you have backed up the few ‘c’ files that are important to you, you want to delete all of the files still remaining in your home directory. Examination of the remaining files reveals that the wildcard **\*c\*** will match all of them. However, to ensure that you do not accidentally delete other files, try out the following commands, examining the output:

```
[student@stationX ~] $ ls *c*  
...output omitted...  
[student@stationX ~] $ ls -fd *c*  
...output omitted...
```

13. Delete the remaining ‘c’ files in your home directory. Once more we’ll use **ls** before issuing a destructive command.

```
[student@stationX ~] $ ls *c*_[1-3]  
...output omitted...  
[student@stationX ~] $ rm *c*_[1-3]  
[student@stationX ~] $ ls  
A_reports December November October projects September
```

## Sequence 2: Automating tasks with shell scripts

**Scenario:** You would like to make it easier to automate the backup command you devised in the previous lab. The shell scripts you create will be built-upon in later labs

**Deliverable:** A script that prints information about and performs backups to a datesamped directory.

### Instructions:

1. Consider the command you devised in a previous lab for creating a backup of /etc/sysconfig into a date\_pecific directory. It should have looked something like:

**Cp\_av/etc/sysconfig ~/backups/sysconfig-yyyymmdd**

This lab will have you create a simple shell script for replicating this command. Your script will have the advantage of determining the datestamp automatically. So that a destination does not have to be explicitly stated.

2. First you will need to devise a command that automatically generates a datestamp appropriate format. Consult the format section of man date and write the command you will need below. See the solutions sections for this exercise if you have trouble.

---

3. Now you are ready to create the script. Begin by logging in as root and creating a directory in your home directory called bin. this is the standard location for user's custom scripts.

```
[root@stationX ~] #mkdir ~/bin
```

4. Use a text editor, such as **nano** or **vi**, to create a new file called. ~/bin/backup-sysconfig.sh. Because this file is a shell script, be sure to begin it with a "shbang" and a comment describing the script's purpose:

```
# ! /bin/bash  
# This script creates a backup of /etc/sysconfig  
# Into a datestanped subdirectory of ~/backups/
```

5. Next, add a line that performs the **cp** command from your previous lab. Instead of typing an explicit date, use the \$0 command substitution operator to run the date from the previous step each time backups-sysconfig.sh is executed.

```
Cp -av /etc/sysconfig ~backups/sysconfig-$ (date '+%y%m%d' _
```

6. Finally, add a line that prints some information about the backup that was just completed:

**each “Backup of /etc/sysconfig completed at: \$(date) “**

7. Save the file. Your completed script should look like this:

```
# ! /bin/bash  
# This script creates a backup of /etc/sysconfig  
# Into a dated timestamped subdirectory of ~ /backups/  
  
Cp -av /etc/sysconfig ~ /backups/sysconfig-$ (date `+%y%m%d`)  
Echo “backup of /etc/sysconfig completed at: $ (date)”
```

8. If you already have a backup with today’s datestamp. You should remove it before creating a new backup directory of the same name:

```
[root@stationX ~] # rm -rf ~ /backups/sysconfig-$(date `+%y%m%d`)
```

9. Before you can execute the script, you will need to make it executable. Run the following command to do this:

```
[root@stationX ~] chmod u+x ~ /bin/backup-syscnfig.sh
```

10. You are now be ready to try out your script; you should see something like the following:

```
[root@stationX ~] # backup-sysconfig.sh  
` /etc/sysconfig` -> ` /home/brad/backups/sysconfig-200070129`  
` /etc/sysconfig/irqbalance` -> ` /home/brad/backups/sysconfig-200070129/grub`  
...output truncated  
Backup of /etc/sysconfig completed at: Mon Jan 18:13:22 EST 2007
```

If you have problems, double-check your script and try using **bash -x** in your shbang for diagnostic output

## Tolls for Extracting Text:

- File contents: **less** and **cat**
- File Excerpts: **head** and **tail**
- Extract by column: **cut**
- Extract by keyword: **grep**

## Viewing File Contents Less and Cat:

- **cat**: dump one or more files to STDOUT
  - Multiple files are concatenated together
- **Less**: view file or STDIN one page at a time
  - Useful commands while viewing:
    - **/text** searches for text
    - **n/w** jumps to the next/previous match
    - **V** opens the file in text editor
  - **less**: is the pager used by **man**

## Viewing File Excerpts head and tail:

- **head**: Display the first 10 lines of a file
  - Use **-n** to change number of lines displayed
- **tail**: Display the last 10 lines of a file
  - Use **-n** to change number of lines displayed
  - Use **-f** to change “follow” subsequent additions to the file
    - Very useful for monitoring log files!

The head command is used to display just the first few lines of a file. The default is 10 lines.

```
[student@stationX ~] $ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
ip:x:4:7:ip:/var/spool/1pd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
```



**-n** specifies the number of lines to display:

```
[student@stationx ~] $ head -n 3 /etc/passwd
Root:x:0:0:root:/root:/bin/bash
Bin:x:1:1:bin:/bin:
Daemon:x:2:2:daemon:/sbin
```

**tail** is used to display the last few lines of a file. The default is 10. **tail** is often used by the system administrator to read the most recent entries in log files.

```
[root@stationX ~] # tail -n 3 /var/log/cron
Root (10/13-18:01:00-658) cmd (run-parts /etc/cron.hourly)
CRON (10/15-13:50:00-781) STARTUP (fork ok)
Root (10/15-13:50:00-781) cmd (/sbin/rmmod-as)
```

Using **-f** causes **tail** to continue to display the file in “real time” showing additions to the end of the files as they occur. This is very useful for watching growing files. Such as the output of the make command. System administrator uses this feature to keep an eye on the system log using the following command:

```
[root@stationX ~] # tail -f /var/log/messages
Tail-f will continue to show updates to the file until Ctrl-c is pressed.
```

## Extracting Text by Keyword grep:

- Prints lines of files or STDIN where a pattern is matched
- ```
$ grep 'john' /etc/passwd
$ date -help |grep year
```
- Use **-i** to search case-insensitively
  - Use **-n** to print line numbers of matches
  - Use **-Ax** to print include the x lines after each match
  - Use **-Bx** to include the x lines before each match

## Extracting Text by Column cut:

- Display specific columns of file or STDIN data
- ```
$ cut -d: -f1 /etc/passwd
$ grep root /etc/passwd | cut -d: -f7
```
- Use **-d** to specify the column delimiter (default is TAB)
  - Use **-f** to specify the column to print
  - Use **-c** to cut by characters
- ```
$ cut -c2 -5 /usr/share /dict /words
```

## Tools for Analyzing Text:

- Text stats: **wc**

- Sorting Text: **sort**
- Comparing files: **diff** and **patch**
- Spell check: **aspell**

## Gathering Text Statistics **wc** (word count):

- Counts words, lines, bytes and characters
- Can act upon a file or STDIN

```
$ wc story.txt
```

```
39  237 1901 story.txt
```

Use **-l** for only line count

Use **-w** for only word count

Use **-c** for only byte count

Use **-m** for character count (not displayed)

If you specify more than one file, we also reports totals for whatever values it has been told to count.

```
[student@stationX ~] $ wc .bash*
```

```
3      3      24  .bash_logout
```

```
13     29     191  .bash_profile
```

```
20     55     337  .bashrc
```

```
36     87     552  total
```

**wc** can also accept data on the standard input. This can be useful for counting the number of lines in a command's output. For example:

```
[student@stationX ~] $ ls /tmp ! wc -l
```

Shows us that **ls/tmp** produces 32 lines of output (even though by default your terminal will probably display them in columns). Therefore there are 32 files and directories in /tmp.

## Modifying a File Insert Mode:

- I begins insert mode at the cursor
- Many other options exists
  - A append to end of line
  - I insert at beginning of line
  - O insert new a line (below)
  - O insert new line (above)

## Editing text in insert mode:

Many commands will take you into insert mode. The slide above lists six common ways to do so.

The I command will insert your data before the cursor. The letter I will not appear in your document, but all other characters that type will appear, until you exit insert mode, hit the esc key.

The command will place you in insert mode. Allowing you to append after the cursor.

Again, exit insert mode by hitting the esc key.

The o command opens a line below the current line, placing you in insert mode.

Note the relationship between the lower case **a**, **I**, and **o**, and the upper case **A**, **I**, whereas the **a** appends after the cursor, the **A** appends at the end of the line. The **I** insert before the cursor; the **I** inserts at the beginning of the line. The **o** opens a line below the current line; the **O** opens a line above the current line. Patterns such as these permeate the **vi** and **vim** commands.

## **Saving File and Exiting vim Ex Mode:**

- **Enter Ex Mode with:**
  - **Creates a command prompt at bottom-left of screen**
- **Common Write/quit Commands:**
  - **:w** writes (saves) the file to disk
  - **:wq** writes and quits
  - **:q** quits, even if changes are lost

## **Using Command Mode:**

- Default mode of vim
- Keys describe movement and text manipulation commands
- Example
  - Right arrow moves right one character
  - 5, Right arrow Moves right five characters

## **Moving Around Command Mode:**

- Move by Character: arrow Keys, h, j, k, l
  - Non- arrow keys useful for remote connections to older systems
- Move by word: **w**, **b**
- Move by sentences: **)** **‘**(
- Move by paragraph: **}** **{**
- Jump to line X: **xg**
- Jump to end: **G**

## **Search and Replace Command mode:**

- Search as in **less**
  - **/**, **n**, **N**

- Search/Replace as in **Sed**
  - Affects current line by default
  - Use x, y ranges or % for whole file
    - **:1, 5s/cat/dog/**
    - **:%s/cat/dog/gi**

### Manipulating Text Command mode:

	Change (replace)	Delete (Cut)	Yank (copy)
Line	cc	dd	yy
Letter	cl	dl	yl
Word	cw	dw	yw
Sentences ahead	c)	d)	y)
Sentences behind	c(	d(	y(
Paragraph above	c{	d{	y{
Paragraph below	c}	d}	y}

## Lab 9

## Using vim

**Goal:** Familiarity with using vim for system administration tasks

### Sequence 1: Navigating with vim

**Scenario:** You wish to familiarize your self with vim by navigating a copy of a familiar document, /etc/passwd.

### Instructions:

1. Log in as user student with the password student. If you are using the graphical environment. Start a terminal by clicking applications->Accessories->Terminal
2. create your own copy of /etc/passwd in your home directory:

```
[student@stationX ~] $ cp /etc/passwd ~
```

Note that if you intended to actually modify this file you would have to be logged in as root and it would be considered best-practice to use the **vipw** command instead of running vim directly. **Vipw** uses prevents other instances of **vim** from opening the file at the same time, That way to administrators cannot accidentally overwrite each other's changes.

3. Open your copy of passwd in **vim**

```
[student@stationX ~] $ vim ~/passwd
```

4. First, try moving around using standard keys such as the arrows, PgUp, PgDn, Home and End. They should all work as expected. Note, however, that on older systems they might not.
  5. Now try moving from word to word with the w and b keys. Note that the keys for moving by sentence (the parentheses) and paragraph (the curly braces) simply move from one end of the file to the other. Why is this?
- 

6. Try combining numbers with movement keys. For example:

**5w**

**2 Down Arrow**

7. Next you will enter insert mode and make some changes. Remember that since you are working as a non-root user on a copy of the real passwd file, there is no danger of actually damaging the system.

Press the **i** key. Note that the word INSERT appears at the bottom of your screen that the arrow keys still move your cursor, but the other keys now modify the document, as they would in an ordinary text editor.

8. Once you have changed some text, exit insert mode by pressing **Esc**. Note that the INSERT disappears from the bottom of your screen.

## Managing Ethernet connections:

- Network interfaces are named sequentially: **eth0**, **eth1**, etc
  - Multiple addresses can be assigned to a device with aliases
  - Aliases are labeled eth0:1, eth0:2, etc
  - Aliases are treated like separate interfaces

## View interface configuration with **ifconfig [ethx]**

If you want to see your LAN card configuration you can see by this command  
For example:

[student@stationX ~] \$ **Ifconfig ethx** (x is the name of your LAN card)

**For enabling and disabling the LAN card**

**For enable:** [student@stationX ~] \$ **ifup ethx**

**For disable:** [student@stationX ~] \$ **ifdown ethx**

Network connection names consist of a prefix. Based on the device type, and a number to distinguish a particular device from others of its type. For example, all Ethernet devices have the prefix eth. The first detected Ethernet card is assigned the name eth0. the second eth1 and so forth every system also has a special network device called the lo, which represent the ‘ ‘ localhost’ ’ or ‘ ‘ loopback’ ’ device with address 127.0.0.1 you can view the basic settings of a network device by running the **ifconfig** command. By default will print information on all active devices. If given a device name as an argument. It will print information about that device only:

[student@stationX ~] \$ **Ifconfig eth0**

```
Eth0      link encap:Ethernet HWaddr 00:09:6b:cd:2b:87
          Inet addr: 192.168.0.254 Bcast: 192.168.0.255 mask: 255.255.255.0
          Inet6 addr: fe80:: 209:6bff: fecd:2b87/64 scope: link
          UP BROADCAST RUNNING MULTICAST MTU: 1500 metric: 1
          RX packets:851525 errors:0 dropped :0 overruns:0 frame:0
          TX packets:1132322 errors:0 dropped:0 overruns:0 carrier:0
          Collisions:0 txqueuelen: 1000
          RX bytes:211140434 (201.3 mib) TX bytes: 1113058956 (1.0 gib)
```

## Graphical Network Configuration sstem-config-network:

- System->Administration->Network
  - Activate/deactivate interfaces
  - Assign IP Addresses/DHCP
  - Modify DNS settings
  - Modify gateway address

## Network Configuration Files Ethernet Devices:

- **Device configuration store in text files**
  - /etc/sysconfig/network-scripts/ifcfg-ethx
- **Complete list of options in**
  - /usr/share/doc/initscripts-\* /sysconfig.txt

Dynamic configuration	Static configuration
-----------------------	----------------------

DEVICE=ethx HWADDR=0:02:8a:A6:30:45 BOOTPROTO=dhcp ONBOOT=YES TYPE=ethernet	DEVICE=ethx HWADDR=0:02:8a:A6:30:45 IPADDR=192.168.0.254 NETMASK=255.255.255.0 GATEWAY=192.168.2.254 ONBOOT=YES TYPE=ehternet
---	---

## Network Configuration File other Global Network Settings:

- Global setting in /etc/sysconfig/network
  - **Many may be provided by DHCP**
  - **Gateway can be overrrdden in if cfg file**

```
NETWORKING =YES
HOSTNAME=server1.example.com
GATEWAY= 192.168.2.254
```

Some network settings are defined by globally. Rather than on a per-interface basis. These global network settings are defined in the /etc/sysconfig/network file. Some of the more important settings that can be defined in this file include:

Setting	meaning
Networking	Whether to enable networking at all. Should normally be set to yes.
GATEWAY	The IP Address of the system or devices to send message destined for hosts on another networks. It is the responsibility of the gateway to determine how to contact the destination host. Specifying this is only necessary when not using DHCP. This can also be set in an ifcfg, the gateway defined in the most recently activated ifcfg file used
HOSTNAME	The system's hostname. This should be the DNS name that Resolves to its primary IP address. If you are using DHCP, it Is probably not necessary to define this. If you do not define this and your system it is not using DHCP then it will ask DNS what name is associated with your IP address and use that. If DNS does not have a name associated with your IP, your system will be assigned the name localhost.localhost in

## Network Configuration Files DNS Configuration:

- Domain Name Service translates hostnames to network addresses
- Server address is specified by dhcp or in /etc/resolv.conf

```
Search example.com cracker.org
name server 192.168.0.254
name server 192.168.1.254
```

## **Printing in Linux:**

- Printer may be local or networked
- Print requests are sent to queues
- Queued jobs are sent to the printer on a first come first served basis
- Jobs may be canceled before or during printing

### **Printing:**

Having created a file, you will no doubt want to print it. The printing system in red Hat enterprise Linux is very flexible. Printers may be parallel, serial, or networked. Support is included for printing to remote CPU's IPP, ipd (common Linux and UNIX printing subsystem), windows, Netware, and jetdirect printers.

### **Queues**

One or more queues is associated with each printer. Print jobs are sent to a queue, not to a printer, directly. Different queues for the same printer may have differing priority or output options. A setting up print queues is the responsibilities of the system administrator; individual users do not create print queues.

### **Jobs:**

Once a file has been sent to a queue for printing, it is called a job, jobs may be canceled while they are printing. Or when they are in the waiting to be printed.

## **System –Config-Printer:**

- **System◇ administration◇ Printing**
- **Supported printer connections:**
  - Local (parallel or used)
  - Unix/Linux print server
  - Window print server
  - Netware print server
  - HP jet direct



- **Configuration stored in** /etc/cups/printers. Conf

### Printing Commands:

- **lpr** sends a job to the queue to be printed
  - Accepts ASCII, post scripts
- **lpq** views the contents of the queue
- **lprm** removes a job from the queue
- System V printing commands such as lp, lpstat and cancel are also supported

### Using the print utilities:

The **lpr** command is used to send a job to the printer. The Linux printing system will print files in ASCII. Postscript, PDF, and other formats. Most applications under Linux output post script.

The **-p** option is used to select a queue other than the default and **-#** is used to specify the number of copies. For example, to print 5 copies. Of the file reports on the accounting printer:

```
[student@stationX ~] $ lpr -p accounting -#5 report.ps
```

**lpq** views the contents of the queue

When entered without options. **lpq** lists the jobs in the default queue. As with **lpr**, **-p** is used to specify a queue other than the default. For example:

```
[student@stationX ~] $ lpq
```

Printer: ps@localhost

Queue: no printable jobs in queue

Server: no server active

Satus: job 'jay@localhost+916' removed at 12:16:03

Rank owner/id	Class job files	size time
---------------	-----------------	-----------

Done jay@localhost+185	A 185 results	2067 08:38:04
------------------------	---------------	---------------

To remove a job from the print queue, use **lprm** followed by the job number, specify a non-default print queue if necessary. For example:

```
[student@stationX ~] $ lprm 916
```

In this example. **Lprm** responds with the name of the queue from which the job was removed. Note that a user may only remove his own print jobs from the queue.

### Printing Utilities:

- **evince** views PDF documents
- **lpstat -a** lists configured printers
- **enscripts** and **a2ps** convert text to postscripts
- **ps2pdf** convert postscripts to PDF
- **mpage** prints multiple pages per sheet

### *Tools to assist in printing tasks:*

Several utilities are included with red Hat Enterprise Linux to create output for the printer and interact with postscript files.

### **Enscript, a2ps:**

These command convert text to postscript and send it to the print queue or a file. They are often useful to send the output of a command to the printer via a pipe.

### **evince:**

the evince utility is used to view PDF files.

### **Ps2pdf:**

This utility creates PDF files. You can use any program that will create a postscript file, and then use ps2pdf to convert it to a pdf file. There are a number of versions of this program called ps2pdf12, ps2pdf13, ps2pdf14, respectively creating PDF version 1, 2, 3 and 1, 4 output files.

### **Pdf2ps:**

This utility converts PDF files to postscripts, which makes it easy to print PDF documents right from the command line. There is also a pdftotext which converts pdf documents to plain text documents.

### **Mpage**

Prints ASCII or postscript input with text reduced in size, so that multiple pages of input appear on a single sheet of paper.

## **Setting the system's Date and time:**

- **GUI: system-config -date**
  - System ◇ administration ◇ date & time
  - Can set date/times manually or use NTP
  - Addition NTP servers can be added
  - Can use local time or UTC

- **CLI: date [MMDDhhmm[[cc]yy][.ss]]**
  - # date 01011330
  - date 010113302007.05

The most basic way to alter your system's date is to use the date command date --help reports that the following syntax should be used:

[mmddhhmm [[cc]yy] [.ss]]

This means that you must at least provide two-digit values for the month, day hour and minute you wish to set, with an optional two or four digit year and an optional number of seconds, preceded, by a period some examples:

```
# date 12312359      # 31st of dec, at 11:59pm. No change to the year.
# date 123123592007  # as above, but also sets the year to 2007
# date 01010101.01   # 1st of jan, at 01:01:01am. No change to the year.
```

While less intuitive than graphical tools. Date has the advantages of not relying on a graphical environment and being easy to use in a shell script.

**System-config-date** is used to graphically configuration your system's date and time setting. It can be accessed by selecting system->administration->Date & time

Occasionally. all the controls on the first tab are unelectable (grayed-out). This mean that the system's time is being set from another server via the Network time protocol (NTP) and cannot be set manually. To regain manual control click the second tab. **Network Time protocol** and deselect the **Enable network Time protocol** checkbox.

On **time Zone** tab the system clock can be set for local time or for UTC (Greenwich Mean Time). This is controlled by selecting the **system clock uses UTC** checkbox.

## Lab 10:

### Understanding the Configuration Tools

---

**Goal:** Become familiar with some of Red Hat Enterprises Linux's system administration tools.

**System Setup:** A working. Installed Red Hat Enterprise Linux system with an unprivileged user account named student with a password of student. Root's password should be redhat

### Sequence 1: Configuration the Network with system-config-network

**Scenario:** you wish to use system-config-network to add a second IP address to eth0 the address for this interfaces will be 192.168.50.x where x is your station number.

### Instruction:

1. Select system->Admission->Network.  
Enter the root password, **redhat**, if prompted.
3. Click New, highlight Ethernet Connection, and click forward.
4. Highlight the device associated with eth0 and click forward.
5. Enter 192.168.50.x where x is your station number in the Address field.
6. Enter 255.255.255.0 in the subnet mask field. Leave the gateway field blank.
7. Click forward, then apply.
8. You should see a new profile attached to eth0 with the nickname''eth0:''', Note that it is inactive by default. Highlight the new interface and click **Activate**. If you are prompted to save your changes, click **yes**
9. When your neighbor has also completed this exercise you cant test each other's configurations with the ping. For example. If your neighbor is at station1. you can run:

[student@stationX ~] \$ **Ping -c4 192.168.50.1** to test

### Sequence 2: Configuring the network with interface configuration files

**Scenario:** you will begin by examining the **ifcfg** file created by the previous exercise then you will create a new interface using only a text editor.

### Instruction:

1. Open a terminal and run **/sbin/ifconfig**. Among the output you should see an entry for your new interface:

```
Eth0:1          link encap:Ethernet      Hwaddr  00::oc:29:2e:be:8a
                Inet  addr:192.168.50.x    bcast:192.168.50.255
                Mask: 255.255.255.0
```

```
UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
    Interrupt: 169 Base  Address:0x1080
```

2. Why do need to type the full path to **/sbin/ifconfig** if you are not logged in as root?
3. **Cd** to the **/etc/sysconfig/network-scripts** directory and examine the **ifcfg-eth: 1** file. You should see something like this

```
[student@stationX network-scripts]$ cat ifcfg-eth0:1
# please read /usr/share/doc/initscripts-*/sysconfig.txt
# For the documentation of these parameters.
```

```
Type=Ethernet
DEVICE=eth0:1
BOOTPROTO=none
```

```
NETMASK=255.255.255.0
IPADDR=192.168.50.x
USERCTL=no
IPV6INIT=no
PEERDNS=yes
```

4. You can now use this file as a template to create new interface configurations. Begin by creating a copy of eth: 1's configuration called eth0:2:  
[student@stationX network-scripts]\$ **sudo cp ifcfg-eth0:1 ifcfg-eth:2**

5. Open ifcfg-eth0:2 and the following changes:

- Change DEVICE to **eth0:2**
- Change IPADDR to **192.168.51.x**, where x is your station number
- Change USERCTL to **yes**

If you are using vim, this can be accomplished with the following command sequence:

```
3Dwn 4w r2
3Down 2w Right r1
Down Left cw yes
Esc: wq
```

6. Bring your new interface up: Note that because of USERCTL, being set to yes, you do not need to be root to do this:

```
[student@stationX network-scripts] $ /sbin.ifup eth: 2
```

7. Use **ifconfig** to verify that your new interfaces have the expected settings. You should see something like:

```
Eth0:2      link encap: Ethernet      Hwaddr  00:0C:29:2e: BE: 8a
            Inet addr: 192.168.51.x Bcast: 192.168.51.255
            Mask: 255.255.255.0
            UP BROADCAST RUNNING MULTICAST MTU: 1500 Metric: 1
            Inrupt: 169 Base addresses: 0x1080
```

8. Work with a partner to use to ping to verify the connection, as in the previous exercise.
9. clean up by brining both interfaces down:

```
[student@stationX network-scripts] $ sudo ifdown eth: 1
[student@stationX network-scripts] $ Ifdown eth: 2
```

10. Will these interfaces be re-enabled if the system is rebooted? Why or not?

### Sequence 3: Configuring Printers

**Scenario:** you wish to configure a new printer for the system using system-config-printer. For the purposes of this lab, assume that a generic postscript printer is connected to your system's parallel port (LPT).

**Instruction:**

1. Select System->administration->printing.
2. Select new to begin the configuration wizard.
3. Enter fakeprinter as the name of your printer and click forward.
4. Select LPT #1 and click forward.
5. Select generic and click forward.
6. Select postscript printer and click Forward
7. Click Apply
8. You should now be presented with a list of printers configured on this system. Select your new printer. Note that there are many more options that can be configured after the printer's initial configuration, including buttons to print a test page and to make this the system's default printer.
9. Close **system-config-printer**

### **Sequence 4: Configuration Date and using date**

**Scenario:** Use the following problems to become proficeantwith using the date command to configure your system's date and time from the command-line.

**Instructions:**

1. What date command would be used to set the system's date and/or time to the following values. Remember the usage summary [mmddhhmm[[cc]yy] [.ss]].

The second of January at 3:00pm\_\_\_\_\_

The first of January, 2008 at thirty seconds after midnight

### **Sequence 5: Synchronizing Date and Time with a network Time Server**

**Scenario:** Ensure that your system's date are synchronized with the date and time on server1 by using system-config-date to configure the Network Time protocol (NTP)

**Instruction:**

1. Select system->Administration->date&time menu
2. Select the **Network Time protocol** tab.

3. Check Enable Network Time protocol.
4. Enter server1.example.com in the Server box.
5. Click OK.

## Listing Processes:

- View process information with **ps**
  - Shows processes on all terminals by default
  - -a includes processes on all terminals
  - -x includes processes not attached to terminals
  - -u prints process owner information
  - -f prints process parentage
  - -o **property**,... prints custom information:
  - Pid, comm, %cpu, %mem, state, tty, euser, ruser

## Finding Processes:

- **Most flexible: ps options | other commands**  
ps axo comm, tty | Grep ttyso
  - By predefined patterns: pgrep  
\$ pgrep -u root  
\$ pgrep -G student
  - By exact program name: pidof  
\$ pidof bash

## Viewing Specific process Information

Since there may be hundreds of processes on a common technique to locate a specific process is to send output from **ps** to **grep**

```
[student@stationX ~] $ ps axo pid, comm | grep 'cups'
```

2734 cupsd  
3502 eggcups

Compare the above output with that from pgrep

```
[student@stationX ~] $ pgrep cups
```

2734  
3502

A more exact method of obtaining PID's for processes is the **pidof**. Which matches exactly on the program name specified and therefore requires that you know the specific program name?

```
[student@stationX ~] $ pidof cupsd  
2734
```

## Signals:

- Most fundamental inter process communication
  - **Sent directly to processes, no user interface required**
  - **Program associate actions with each signal**
  - **Signals are specified by name or number when sent:**
    - Signal15, TERM (default) –terminate cleanly
    - Signal 9, KILL, -terminate immediately
    - Signal 1 ,HUP –RE-read configuration files
    - Man 7 signal show complete list

## Signals

Signals are simple message that can be communicated to process with command like **kill**. Which will be discussed on the next page? The advantage of signals is that they can be sent to a process even if it is not attached to a terminal and display an interface. So a web server or a graphical program whose interface has “frozen” may still be shut down by sending it the appropriate signal.

Though in most cases software developers can associate any action with the reception of a particular signal. Almost all signals have standard meanings. Signal can be specified by their name. such as kill. Or by their number. Such as 9. A called detailed list of signals including names. Numbers and meanings can be displayed with **man 7 signal**.

## Sending signals to processes:

- By PID: **kill** [signal] pid...
- By Name: **Killall** [signal] comm...
- By pattern: **pkill** [-signal] pattern

The following are all identical and will send the default term signal to the process with PID number 3428:

```
[student@stationX ~] $ kill 3428
```



```
[student@stationX ~] $ kill -15 3428
[student@stationX ~] $ kill -SIGTERM 3428
[student@stationX ~] $ Kill -TERM 3428
```

### **Scheduling Priority:**

- Scheduling Priority determines access to CPU
- Priority is affected by a process' nice value
- Values range from -20 to 19 but default to 0
  - Lower nice value means higher CPU priority
- Viewed with **ps -0 comm, nice**

### **Schedule priorities**

Every running process has a scheduling priority: a ranking among running process determining which should get the attention of the processor. The formula for calculating This priority is complex, but users can affect the priority by setting the "niceness" value, one element of this complex formula. The niceness value defaults to zero but can be set from -20 (least nice, highest priority) to 19 (most nice, lowest priority).

### **Altering Scheduling Priority:**

- Nice values may be altered...
  - When starting a process:  
\$ nice -n 5 command
  - After starting :  
\$ renice 5 PID
- Only root may decrease nice values

### ***Adjusting priorities for programs***

To set the niceness value to a specific value when starting a process, use **nice-n**

```
[student@stationX ~] $ nice -n 15 myprog
```

Non-privileged users may not set niceness value to less than zero: that is, they not request a higher than normal priority for their processes. Only root may allocate additional CPU clock cycles to a process.

### ***Altering priorities of running programs***

All users may raise the niceness (reduce the priority) of their own jobs using the renice command:

```
[student@stationX ~] $ renice 15 -p pid
```

Note that non-privileged users may start a process at any positive nice value but cannot lower it once raised. Root is permitted to reduce the niceness (raise the priority) of currently running process

```
[root@stationX ~] $ renice -15 -p pid
```

## Interactive Process Management Tools:

- CLI: **top**
- GUI: **gnome-system-monitor**
- Capabilities
  - Display real-time process information
  - Allow sorting, killing and re-nicing

### *The top command*

Running **top** present a list of the process running on your system, updated every 5 seconds. You can use keystrokes to kill, renice and change the sorting order of process. That are in the running state are highlighted and you can colorize processes and create multiple windows to view more than one sorted list of process at a time, press the? key while in top to view the complete list of hotkeys. You can exit top by pressing the q key.

### *The gnome-system-monitor command*

The gnome-system-monitor, which can be run from the cc:: sole or by selecting Application system Toolsystem Monitor from the menu system, is a graphical tool that also offers the ability to view sorted lists of process and to kill or renice those process.

By default the tool only display processes that are in the “Running” state, However this can be changed by selecting All process or my process from the dropdown menu in the upper-right. The preferences dialog (Editpreferences) allows you to customize which fields are displayed and you can sort by a particular field by clicking on its heading. You can send a process the TERM signal by right-clicking on it and selecting End process or the kill-9) by selecting kill process. A process can be re-niced by right-clicking on it and selecting Change priority.

## Scheduling a process To Execute Later:

- One-time jobs use at, recurring jobs use **crontab**

Create	At time	<b>crontab -e</b>
List	At-l	<b>crontab -l</b>
Details	At -c jobnum	N/A
Remove	At -d jobnum	<b>crontab -r</b>
Edit	N/A	<b>crontab -e</b>

- Non-redirected output is mailed to the user
- Root can modify jobs for other users

### ***Schedule a one-time job with at***

Commands are entered, one per line, and terminated with a Ctrl-d on a line by itself. In the following example, you would like to compare network services on your machine to a baseline that is stored on non-writable media:

```
[student@stationX ~] $ at 0200
```

```
□ netstat -tulpn | diff - /media/cdrom/netstat_baseline
```

```
□ Ctrl-d
```

```
Job 1 at 2007-01-08 22:45
```

```
[student@stationX ~] $ at -1
```

```
1          2007-01-09 02:00 a student
```

**At-1** (an alias to **atq**) lists the job currently pending, **at -c jobnum** cats the full environment for the specified job number, and **at-d jobnum** deletes the job.

root can modify other user's jobs by first getting a login shell as that user (**su-username**).

The time argument has many formats which are illustrated by the following example.

**At 8:00pm December 7**

**At midnight + 23 minutes**

**at 7 am Thursday**

**at now + 5 minutes**

See **man at** for information.

### Schedule recurring jobs with **crontab**

The cron mechanism is controlled by a process named **crond**. This process runs every minute and determines if an entry in users cron tables need to be executed. If the time has passed for an entry to be started, it is started. A cron job can be scheduled as often as once a minute or as infrequently as once a year.

root can modify recurring jobs for any user with **crontab -u username** and any of the other options. Such as **-e**

See **man crontab** for more information.

### **Crontab File Format:**

- Entry consists of five space-delimited fields followed by a command line
  - One entry per line, no limit to line length
- Field are minut, hour, day of month, month and day of week

- Comment lines begin with #
- See **man 5 crontab** for details

### ***Crontab examples***

Entry fields can be separated by any number of tabs or spaces. Valid field values are as follows:

Minute	0-59
Hour	0-23
Day of month	0-31
Month	0-12
Day of week	0-6 (0=Sunday)

Multiple values may be separated by commas. An asterisk in a field represents all valid values. A user's crontab may look like the following:

#	Min	Hour	DoM	Month	DoW	Command
	0	4	*	*	1, 3, 5	find ~ -name core   xargs rm -f { }
	0	0	31	10	1, 3, 5	mail -s 'boo' \$LOGNAME < boo.txt
	0	2	*	*	*	netstat -tulpin   diff

~/media/cdrom/baseline

### **Grouping commands**

- Two ways to group commands:
  - Command: `date;who |wc-l`
    - Commands run back to back
  - Subshell: `(date; who | wc-l) >> /tmp/trace`
    - All output is sent to a signal STDOUT and STDERR

---

## **LAB 11** **Process Control**

---

**Goal:** Practice using the process control related commands.

**System Setup:** A working. Installed Red Hat Enterprise Linux system with an unprivileged user account named student with a password of *student*.

**Lab Setup:** Some parts of this lab will require you to check your local email. To do this. You will need to run the mutt command, which provides a powerful. Non-graphical mail client. To use mutt effectively. You should familiarize yourself with the following commands:

- Arrow keys to move between messages
- Enter to view the selected message

- While viewing a message:
  - Enter to scroll down
  - Backspace to scroll up
  - **q** to return to the message list
- **d** to delete the selected message
- **q** to quit **mutt**

## Sequence 1: Job Control

**Scenario:** Explore job control.

**Instruction:**

1. Bring up a root shell by switching to root:

```
[student@stationX ~] $ su -
```

2. Begin some jobs in the background:

```
[root@stationX ~] # tail -no -f /var/log/message &
[root@stationX ~] # updatedb &
```

3. The **-n0** option passed to **tail** cause no lines from **/var/log/messages** to be printed at first. The following command will cause some lines sent to the system logfile and hence to be printed by **tail**. Note that even when a program is in the background, it can still print to the screen.

```
[root@stationX ~] # service syslog restart
... output omitted
```

4. Now that you have a few jobs running in your shell, request a list of them with the **jobs** command.

```
[root@stationX ~] # jobs
[1]-  Running                  tail -no -f /var/log/messages &
[2]+  Running                  updatedb &
```

5. Kill the **tail** process by referencing its job id (probably 1). Remember to prepend a **%** so you refer to a job ID and not a process ID!

```
[root@stationX ~] # kill %1
```

```
[root@stationX ~] # jobs
```

```
[1]-  terminated
```

```
[2]+  Running
```

```
tail -no -f /var/log/message &  
updatedb &
```

6. Next, start an instance of **vim**

```
[root@stationX ~] # vim
```

7. While in vim, press Ctrl-z to suspend the current program. You should find yourself back at a bash prompt.

8. Run **jobs** again and note **vim**'s job **ID**.

### Sequence 3: Scheduling One-Time jobs

**Scenario:** In this chapter you will schedule a job for one-time execution at a specified time using the tool you developed in the previous sequence.

#### Instructions:

1. Start in a shell, either on a virtual console or a graphical shell, such as gnome-terminal. You should be signed in as student.
2. Schedule your **reach.sh** tool to check all stations five minutes from now:

```
[student@stationX ~] $ at now+5min
```

```
at> for X in $(seq 1 40) ; do
```

```
at> reach.sh station$X
```

```
at> Ctrl-d
```

```
Job 7 at 2007-01-23 08:40
```

Note: Since your script only emits output when there is a problem, you do not have to worry about redirecting STDOUT or STDERR in regular usage. Your job will only notify you of unreachable stations!

3. The system responded with a job number, but list your scheduled jobs to see any others that you may have created (or that root may have created for you!):

```
[student@stationX ~] $ at -l
```

```
Job 7 at 2007-01-23 08:40 a student
```

4. For detailed information, cat the job:

```
[student@stationX ~] $ at -c 7
```

Read this output, observing that each **at** job stores the environment for the user that created the job. The job's command is listed near the bottom of the output.

5. Optionally, watch the job list until your job executes.

```
[student@stationX ~] $ watch -n1 'at -1'
```

6. Check your mail after the job executes to review its output.

```
[student@stationX ~] $ mutt
```

## Sequence 5: Recurring jobs

**Scenario:** In this sequence you will take the command you developed in the previous sequence and adapt it for use in a recurring job. You would like the output mailed to student's mail complete with column headings.

### Instructions:

1. Review the man page for crontab to check the field order:

*Man crontab*

2. Oops! There are two man pages for crontab, so open up the one in section 5. which deals with configuration files:

*Man 5 crontab*

3. Use information in the man page to determine how would you write a crontab entry that should run every five minutes?

---

4. Use your answer to the previous question to add a crontab entry that runs the **ps** command from earlier five minutes.

You can do this either by running **crontab-e** and using a text editor or by piping directly to **crontab** like this:

```
Each '*/*5 * * * * ps axo pid,comm.,pcpu --sort=pcpu | head -n2 '
crontab
```

5. Once you have added the job, list your crontab to confirm:

**Crontab -l**

6. Now add lines to run your reach.sh command on server1 and station 100 every two minutes. Because **/usr/local/bin** is not in the PATH used by cron, you will need to use an absolute path to the script.

Your crontab should now look like this:

```
* /5 * * * *      ps axo pid,comm,pcpu -- sort=.pcpu | head -n2
*/2 * * * *       /usr/local/bin/reach.sh server1
*/2 * * * *       /usr/local/bin/reach.sh station100
```

6. Once a few minutes have passed. Check your mail with the **mutt** command to see the output of your jobs. See the instructions at the beginning of this lab if you are unfamiliar with **mutt**

Some observations about what you should see:

- ☐ You did not receive any mail regarding the reach ability of server1: your script correctly avoids output on successful completion
- ☐ You received at least one message regarding the failure to reach station 100

8. For cleanup, remove your crontab:

```
Crontab -r ; cr -1
```

## Bash Variables:

- **Variables are named values**
  - **Useful for storing data or command output**
- **Set with VARIABLE=VALUE**
- **Referenced with \$VARIABLE**

```
$HI="HELLO, and welcome to $(hostname)"
```

```
$ echo $HI
```

```
HELLO, and welcome to stationx
```

Sometimes it is helpful to be able to store information and reference it later. In bash. This is done with variable. Bash recognizes that you are trying to set a variable when it sees the pattern text=text. Note that there can be no spaces on either side of the = or you will get an error.

So, to set the variable FILES to the output of ls/etc, you would type

```
[student@stationX ~] $ FILES=$(ls /etc/)
```



You can access the value of a variable by prepending a \$ \$ to its name. This is called dereferencing or expanding the variable.

```
[student@stationX ~] $ echo $FILES
```

```
A2ps.cfg a2ps.site.cfg acpi adjtime alchemist aliases aliases.db allegro.rc also alternatives
anacrontab ant.conf ant.d apt asound.state at.deny audit
Autofs_ldap_auth.conf auto.master auto.misc auto.net auto.smb avahi bashrc
... Output truncated ...
```

You will find variables very useful for writing more complex shell scripts and for configuring some programs. Which look for the values of certain variables rather than reading a configuration file. The latter will be discussed when Environment variables are introduced on the next page.

## **Environment Variables**

- Variables are local to a shell by default
- Environment variables are inherited by child shells
  - Set with **Export VARIABLE=VALUE**
  - Accessed by some programs for configurations

Some variables, like \$HOME, are intended to be referenced but not set by the user. However, users can also create their own variables:

```
[student@stationX ~] $ HI='Hello' pleased to meet you,'
[student@stationX ~] $ echo $HI
Hello, pleased to meet you.
```

When setting variables at the command line, do not include spaces between the variable, the equals sign and the value. Entering something like **HI = "Hello, pleased to meet you"** would probably generate a syntax error because **bash** will try to execute a command called **HI**, which is unlikely to exist.

Two types of variables exist: local variables, also called shell variables, and environment variables. The difference between the two is that the shell will pass the environment variable on to commands that it calls, but it will not pass on local variables. As a result, local variables are used to configure the shell itself, while environment variables are used to configure other commands.

The **set**, **env**, and **echo** commands can be used to display all variables, environment variables, and a single variable value, respectively. Examples:

```
[student@stationX ~] $ set I less
[student@stationX ~] $ env I less
```

```
[student@stationX ~] $ echo $HOME  
/home/student
```