Practical 9A – Basic Linux Shell Commands

Course : ITDF15 - Diploma in InfoComm & Security

Module : IT2654 – System Administration and Security

Objectives

1. To be familiar with some of the common Linux distributions.
2. To be familiar with basic RHEL features.
3. Know how to use basic Linux bash shell commands.

Lab Requirements:

Red Hat Server RHEL 6.
Login to the server as **student** or **rhel6** with the standard password **redhat**
Try out all the commands below.

1. Linux history

All modern operating systems have their roots in 1969 when **Dennis Ritchie** and **Ken Thompson** developed the C language and the **Unix** operating system at AT&T Bell Labs. They shared their source code (yes, there was open source back in the Seventies) with the rest of the world, including the hippies in Berkeley California. By 1975, when AT&T started selling Unix commercially, about half of the source code was written by others. The hippies were not happy that a commercial company sold software that they had written; the resulting (legal) battle ended in there being two versions of **Unix** in the Seventies : the official AT&T Unix, and the free **BSD** Unix.

In the Eighties many companies started developing their own Unix: IBM created AIX, Sun SunOS (later Solaris), HP HP-UX and about a dozen other companies did the same. The result was a mess of Unix dialects and a dozen different ways to do the same thing. And here is the first real root of **Linux**, when **Richard Stallman** aimed to end this era of Unix separation and everybody re-inventing the wheel by starting the **GNU** project (GNU is Not Unix). His goal was to make an operating system that was freely available to everyone, and where everyone could work together (like in the Seventies). Many of the command line tools that you use today on **Linux** or Solaris are GNU tools.

The Nineties started with **Linus Torvalds**, a Swedish speaking Finnish student, buying a 386 computer and writing a brand new POSIX compliant kernel. He put the source code online, thinking it would never support anything but 386 hardware. Many people embraced the combination of this kernel with the GNU tools, and the rest, as they say, is history.

Today more than 90 percent of supercomputers (including the complete top 10), more than half of all smartphones, many millions of desktop computers, around 70 percent of all web servers, a large chunk of tablet computers, and several appliances (dvd- players, washing machines, dsl modems, routers, ...) run **Linux**. It is by far the most commonly used operating system in the world.

Linux kernel version 4.2 was released in January 2012. Its source code grew by almost two hundred thousand lines (compared to version 4.1) thanks to contributions of over
4000 developers paid by about 200 commercial companies including Red Hat, Intel, Broadcom, Texas Instruments, IBM, Novell, Qualcomm, Samsung, Nokia, Oracle, Google and even Microsoft.

```
http://en.wikipedia.org/wiki/Dennis_Ritchie
http://en.wikipedia.org/wiki/Richard_Stallman
http://en.wikipedia.org/wiki/Linus_Torvalds
http://kernel.org http://lwn.net/Articles/472852/
http://www.linuxfoundation.org/
http://en.wikipedia.org/wiki/Linux
http://www.levenez.com/unix/ (a huge Unix history poster)
```

!!! All sections with ¤ are optional

2. Linux Distributions

This section gives a short overview of current Linux distributions.

A Linux **distribution** is a collection of (usually open source) software on top of a Linux kernel. A distribution (or short, distro) can bundle server software, system management tools, documentation

and many desktop applications in a **central secure software repository**. A distro aims to provide a common look and feel, secure and easy software management and often a specific operational purpose.

Let's take a look at some popular distributions.

2.1. Red Hat

Red Hat is a billion dollar commercial Linux company that puts a lot of effort in developing Linux. They have hundreds of Linux specialists and are known for their excellent support. They give their products (Red Hat Enterprise Linux and Fedora) away for free. While **Red Hat Enterprise Linux** (**RHEL**) is well tested before release and supported for up to seven years after release, **Fedora** is a distro with faster updates but without support. **CentOS** is similar to RHEL but is always a version or two behind it. Updates for CentOS is free though.

2.2. Ubuntu

Canonical started sending out free compact discs with **Ubuntu** Linux in 2004 and quickly became popular for home users (many switching from Microsoft Windows). Canonical wants Ubuntu to be an easy to use graphical Linux desktop without need to ever see a command line. Of course they also want to make a profit by selling support for Ubuntu. They have both desktop and server versions. The desktop versions come in tablet friendly or non-touchscreen versions to cater to many audiences.

2.3. Debian

There is no company behind **Debian**. Instead there are thousands of well organised developers that elect a Debian Project Leader every two years. Debian is seen as one of the most stable Linux distributions. It is also the basis of every release of Ubuntu. Debian comes in three versions: stable, testing and unstable. Every Debian release is named after a character in the movie Toy Story.

2.4. Other

Distributions like CentOS, Oracle Enterprise Linux and Scientific Linux are based on Red Hat Enterprise Linux and share many of the same principles, directories and system administration techniques. Linux Mint, Edubuntu and many other *buntu named distributions are based on Ubuntu and thus share a lot with Debian. There are hundreds of other Linux distributions.

2.5. Which to choose ?

```
redhat.com      #Restrictive to prevent rogue drivers+apps from crashing OS. Needs paid
                    subscription for updates+support.
ubuntu.com      #Open
debian.org      #Open
```

```
centos.org          #Open. Delayed mirror of RHEL.
distrowatch.com     #Get user reviews, ranking and links to most distros.
```

2.6. RHEL

Red Hat Enterprise Linux has many versions. To find out which version you have, type **cat /etc/redhat-release** . You can also find out which kernel is being used: **uname –a** . Red Hat maintains their own distribution, packages and even the kernel. Most other distributions rely on **kernel.org** (Linus Torvalds) to maintain the kernel.

3. Shells

Shell is a user program or an environment provided for user interaction. Shell is a command language interpreter that executes commands read from the standard input device (keyboard) or from a file. It is not part of the system kernel, but uses it to execute programs, create files etc.

3.1. Types of shells

Shells are modular in nature. You can add or remove shells to your liking. The default shell available in Red Hat is bash.

Several shells available with Linux including:

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	--
TCSH	See the man page. Type $ man tcsh	--	TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

3.2. bash

BASH = **B**ourne **A**gain **Sh**ell

Bash is a shell written as a free replacement to the standard Bourne Shell (**`/bin/sh`**) originally written by Steve Bourne for UNIX systems.

It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.

Since it is free software, it has been adopted as the default shell on most Linux systems.

3.3. Changing shell

Tip: To find all available shells in your system type following command:
`$ cat /etc/shells`
Note that each shell does the same job, but each has a different command syntax and provides different built-in functions.

You can try switching to the C shell by typing:
`$ csh`
Type "exit" to return to your previous shell. You can launch/spawn as many child shells as you like, but remember to type as many exits to return to the first shell.

To find out what shells you are running, type:
`$ ps` (use linux help to find out about ps command)

Or you can type:

`$ echo $SHELL`

In MS-DOS, the shell name is COMMAND.COM which is also used for the same purpose, but it's not as powerful as Linux Shells are!

Any of the above shell reads commands from the user (via Keyboard or Mouse) and tells the Linux kernel what the user want.

If we are giving commands from the keyboard, it is called the command line interface (Usually in-front of "$" prompt. This prompt depends on your shell and environment set by you or your System Administrator, therefore you may get different prompts).

Reading: https://www.dummies.com/programming/networking/cracking-the-unix-shell/

4. man pages

This section will explain the use of **man** pages (also called **manual pages**) on your Unix or Linux VM.

You will learn the **man** command together with related commands like **whereis**.

Most Unix files and commands have pretty good man pages to explain their use. Man pages also come in handy when you are using multiple flavours of Unix or several Linux distributions since options and parameters sometimes vary.

To search for a particular word within a man page, type "`/word`". To quit from a man page, just press the "Q" key.

4.1. man $command

Type **man** followed by a command (for which you want help) and start reading. Press **q** to quit the manpage. Some man pages contain examples (near the end).

```
student@RHELserver:~$ man pwd
Reformatting pwd(1), please wait...
```

4.2. man $configfile

Most **configuration files** have their own manual.

```
student@RHELserver:~$ man resolv.conf
Reformatting resolv.conf(5), please wait...
```

4.3. man $daemon

This is also true for most **daemons** (background programs) on your system..

```
student@RHELserver:~$ man rsyslogd
Reformatting rsyslogd(8), please wait...
```

4.4. whereis

The location of a manpage can be revealed with **whereis**.

```
student@RHELserver:~$ whereis pwd
pwd: /bin/pwd /usr/share/man/man1/pwd.1.gz
```

This file is directly readable by **man**.

```
student@RHELserver:~$ man /usr/share/man/man1/pwd.1.gz
```

Compare this with the "**which**" command.

4.5. man sections

By now you will have noticed the numbers between the round brackets. **man man** will explain to you that these are section numbers. Executable programs and shell commands reside in section one.

```
1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous (including macro packages and conventions), e.g. man(7)
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]
```

4.6. man $section $file

Therefore, when referring to the man page of the passwd command, you will see it written as **passwd(1)**; when referring to the **passwd file**, you will see it written as **passwd(5)**. The screenshot explains how to open the man page in the correct section.

```
student@RHELserver:~$ man passwd          #opens the first manual found
student@RHELserver:~$ man 5 passwd        #opens a page from section 5
```

4.7. man man

If you want to know more about **man**, then Read The Fantastic Manual (RTFM).

Unfortunately, manual pages do not have the answer to everything...

```
student@RHELserver:~$ man woman
No manual entry for woman
```

4.8. Other means of help

Many commands have simple "help" screens that can be invoked with special command flags.
These flags usually look like "`-h`" or "`--help`".
Example: **`ll --help`** **#small letter L**

For bash built-in commands, use the **help** command instead: **`help pwd`**
Examples of built-in commands: pwd, cd, echo, etc. Type **help** to view list.
Some programs, particularly those released by the Free Software Foundation, use info pages as their main source of online documentation. Info pages are similar to man page, but instead of being displayed on one long scrolling screen, they are presented in shorter segments with links to other pieces of information.
For example:

```
$ info df
```

loads the "df" info page. Press 'Q' to quit info.

5. Working with directories/folders

To explore the Linux **file tree**, you will need some basic tools.

This section is a small overview of the most common commands to work with directories : **pwd, cd, ls, mkdir, rmdir**. These commands are available on any Linux (or Unix) system.

This section also discusses **absolute** and **relative paths** and **path completion** in the **bash** shell.

5.1. pwd

The **you are here** sign can be displayed with the **pwd** command (Print Working Directory). Go ahead, try it: Open a command line interface (like gnome-terminal, konsole, xterm, or a tty) and type **pwd**. The tool displays your **current directory**.

```
student@RHELserver:~$ pwd
/home/student
```

5.2. cd

You can change your current directory with the **cd** command (Change Directory).

```
student@RHELserver$ cd /etc
student@RHELserver$ pwd
/etc
student@RHELserver$ cd /bin
student@RHELserver$ pwd
/bin
student@RHELserver$ cd /home/student
student@RHELserver$ pwd
/home/student
```

cd ~

You can pull off a trick with cd. Just typing **cd** without a target directory (argument), will put you in your home directory. Typing **cd ~** has the same effect.

```
student@RHELserver$ cd /etc
student@RHELserver$ pwd
/etc
student@RHELserver$ cd
student@RHELserver$ pwd
/home/student
student@RHELserver$ cd ~
student@RHELserver$ pwd
/home/student
```

cd ..

To go to the **parent directory** (the one just above your current directory in the directory tree), type **cd ..** .

```
student@RHELserver$ cd /usr/share/games
student@RHELserver$ pwd
/usr/share/games
student@RHELserver$ cd ..
student@RHELserver$ pwd
/usr/share
```

*To stay in the current directory, type **cd .** ;-)* We will see useful use of the **.** character representing the current directory later.

cd -

Another useful shortcut with cd is to just type **cd -** to go to the previous directory.

```
student@RHELserver$ cd
student@RHELserver$ pwd
/home/student
student@RHELserver$ cd /etc
```

```
student@RHELserver$ pwd
/etc
student@RHELserver$ cd -
/home/student
student@RHELserver$ cd -
/etc
```

5.3. Absolute and Relative paths

You should be aware of **absolute and relative paths** in the file tree. When you type a path starting with a **slash (/)**, then the **root** of the file tree is assumed. If you don't start your path with a slash, then the current directory is the assumed starting point.

The screenshot below first shows the current directory **/home/student**. From within this directory, you have to type **cd /home** instead of **cd home** to go to the **/home** directory.

```
student@RHELserver$ pwd
/home/student
student@RHELserver$ cd home
bash: cd: home: No such file or directory
student@RHELserver$ cd /home
student@RHELserver$ pwd
/home
```

When inside **/home**, you have to type **cd student** instead of **cd /student** to enter the subdirectory **student** of the current directory **/home**.

```
student@RHELserver$ pwd
/home
student@RHELserver$ cd /student
bash: cd: /student: No such file or directory
student@RHELserver$ cd student
student@RHELserver$ pwd
/home/student
```

In case your current directory is the **root directory /**, then both **cd /home** and **cd home** will get you in the **/home** directory.

```
student@RHELserver$ cd /
student@RHELserver$ pwd
/
student@RHELserver$ cd home          #relative
student@RHELserver$ pwd
/home
student@RHELserver$ cd /
student@RHELserver$ cd /home         #absolute
student@RHELserver$ pwd
/home
```

The ".." is useful in relative path addressing and when the target folder is nearby:

```
student@RHELserver$ cd
student@RHELserver$ pwd
/home/student
student@RHELserver$ cd ..            #relative
student@RHELserver$ pwd
/home
```

5.4. Path completion

The **tab key** can help you in typing a path without errors. Typing **cd /et** followed by the **tab key** will expand the command line to **cd /etc/**. When typing **cd /Et** followed by the **tab key**, nothing will happen because you typed the wrong **path** (upper case E).

You will need fewer key strokes when using the **tab key**, and you will be sure your typed **path** is correct!

5.5. ls

You can list the contents of a directory with **ls**, or with **dir** (similar to MS-DOS).

```
student@RHELserver:~$ ls
allfiles.txt  dmesg.txt  httpd.conf  stuff  summer.txt
student@RHELserver:~$ dir
allfiles.txt  dmesg.txt  httpd.conf  stuff  summer.txt
```

ls -a

A frequently used option with ls is **-a** to show all files and folders. Showing all files means including the **hidden files**. When a file name on a Unix file system starts with a dot, it is considered a hidden file and it doesn't show up in regular file listings. The same applies to folders.

```
student@RHELserver:~$ ls
allfiles.txt  dmesg.txt  httpd.conf  stuff  summer.txt
student@RHELserver:~$ ls -a
.   allfiles.txt   .bash_profile  dmesg.txt   .lesshst  stuff
..  .bash_history  .bashrc        httpd.conf  .ssh      summer.txt
student@RHELserver:~$
```

ls -l

Many times you will be using options with **ls** to display the contents of the directory in different formats or to display different parts of the directory. Typing just **ls** gives you a list of files in the directory. Typing **ls -l** (that is a letter L, not the number 1) gives you a long listing.

You may also use **ll**, an alias command, to do the same.

```
student@RHELserver:~$ ls -l
total 23992
-rw-r--r-- 1 student student 24506857 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 student student    14744 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 student student     8189 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 student student     4096 2007-01-08 12:22 stuff
-rw-r--r-- 1 student student        0 2006-03-30 22:45 summer.txt
```

ls -lh

Another frequently used ls option is **-h**. It shows the numbers (file sizes) in a more human readable format. Also shown below is some variation in the way you can give the options to ls. We will explain the details of the output later in this book.

```
student@RHELserver:~$ ls -l -h
total 24M
-rw-r--r-- 1 student student  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 student student  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 student student 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 student student 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 student student    0 2006-03-30 22:45 summer.txt
student@RHELserver:~$ ls -lh
total 24M
-rw-r--r-- 1 student student  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 student student  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 student student 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 student student 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 student student    0 2006-03-30 22:45 summer.txt
student@RHELserver:~$ ls -hl
-rw-r--r-- 1 student student  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 student student  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 student student 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 student student 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 student student    0 2006-03-30 22:45 summer.txt
student@RHELserver:~$ ls -h -l
total 24M
-rw-r--r-- 1 student student  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 student student  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 student student 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 student student 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 student student    0 2006-03-30 22:45 summer.txt
```

5.6. mkdir

Walking around the Unix file tree is fun, but it is even more fun to create your own directories with **mkdir**. You have to give at least one parameter to **mkdir**, the name of the new directory to be created. Think before you type a leading / .

```
student@RHELserver:~$ mkdir MyDir
student@RHELserver:~$ cd MyDir
student@RHELserver:~/MyDir$ ls -al
total 8
drwxr-xr-x  2 student student 4096 2007-01-10 21:13 .
drwxr-xr-x 39 student student 4096 2007-01-10 21:13 ..
student@RHELserver:~/MyDir$ mkdir stuff
student@RHELserver:~/MyDir$ mkdir otherstuff
student@RHELserver:~/MyDir$ ls -l
total 8
drwxr-xr-x 2 student student 4096 2007-01-10 21:14 otherstuff
drwxr-xr-x 2 student student 4096 2007-01-10 21:14 stuff
student@RHELserver:~/MyDir$
```

¤ mkdir -p

When given the option **-p**, then mkdir will create parent directories as needed.

```
student@RHELserver:~$ mkdir -p MyDir2/MySubdir2/ThreeDeep
student@RHELserver:~$ ls MyDir2
MySubdir2
student@RHELserver:~$ ls MyDir2/MySubdir2
ThreeDeep
student@RHELserver:~$ ls MyDir2/MySubdir2/ThreeDeep/
```

5.7. rmdir

When a directory is **empty**, you can use **rmdir** to remove the directory.

```
student@RHELserver:~/MyDir$ rmdir otherstuff
student@RHELserver:~/MyDir$ ls
stuff
student@RHELserver:~/MyDir$ cd ..
student@RHELserver:~$ rmdir MyDir
rmdir: MyDir/: Directory not empty
student@RHELserver:~$ rmdir MyDir/stuff
student@RHELserver:~$ rmdir MyDir
student@RHELserver:~$
```

¤ rmdir -p

And similar to the **mkdir -p** option, you can also use **rmdir** to recursively remove directories.

```
student@RHELserver:~$ mkdir -p dir/subdir/subdir2
student@RHELserver:~$ rmdir -p dir/subdir/subdir2
student@RHELserver:~$
```

6. Working with Files

6.1. All files are case sensitive

Linux is **case sensitive**, this means that **FILE1** is different from **file1**, and **/etc/hosts** is different from **/etc/Hosts** (the latter one does not exist on a typical Linux computer).

Let's create two files by typing:

```
$ echo "It is cold" > winter.txt
$ echo "It is very cold!" > Winter.txt
```

This screenshot shows the difference between two files, one with upper case **W**, the other with lower case **w**.

```
student@RHELserver:~$ ls
winter.txt  Winter.txt
student@RHELserver:~$ cat winter.txt
It is cold.
student@RHELserver:~$ cat Winter.txt
It is very cold!
```

The **cat** command is used to display contents of a file. Similar commands are the **more** and **less** commands. Try **`less /etc/passwd`** .

6.2. Everything is a file

A **directory** is a special kind of **file**, but it is still a (case sensitive!) **file**. Even a terminal window (**/dev/pts/4**) or a hard disk (**/dev/sdb**) is represented somewhere in the **file system** as a **file**. It will become clear throughout this course that everything on Linux is a **file**.

¤ 6.3. file

The **file** utility determines the file type. Linux does not use extensions to determine the file type. Your editor does not care whether a file ends in .TXT or .DOC. As a system administrator, use the **file** command to determine the file type. Here are some examples on a typical Linux system:

```
student@RHELserver:~$ file pic33.png
pic33.png: PNG image data, 3840 x 1200, 8-bit/color RGBA, non-interlaced
student@RHELserver:~$ file /etc/passwd
/etc/passwd: ASCII text
student@RHELserver:~$ file HelloWorld.c
HelloWorld.c: ASCII C program text
```

The **file** command uses a magic file that contains patterns to recognise file types. This file is located in **/usr/share/file/magic**. Try **`man 5 magic`** for more info.

It is interesting to point out **file -s** for special files like those in **/dev** and **/proc**.

```
root@RHELserver~# file /dev/sda
/dev/sda: block special
root@RHELserver~# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead...
root@RHELserver~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@RHELserver~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

6.4. touch

One easy way to create a file is with **touch**. (We will see many other ways for creating files later in this book.)

```
student@RHELserver:~$ cd                    #ensure you are home!
student@RHELserver:~$ mkdir test
student@RHELserver:~$ cd test
student@RHELserver:~/test$ touch file1
student@RHELserver:~/test$ touch file2
student@RHELserver:~/test$ touch file555
student@RHELserver:~/test$ ls -l
total 0
-rw-r--r-- 1 student student 0 2007-01-10 21:40 file1
-rw-r--r-- 1 student student 0 2007-01-10 21:40 file2
-rw-r--r-- 1 student student 0 2007-01-10 21:40 file555
```

¤ touch -t

Of course, touch can do more than just create files. Can you determine what by looking at the next screenshot? If not, check the manual for touch.

```
student@RHELserver:~/test$ touch -t 200505050000 ChottoMatte
student@RHELserver:~/test$ touch -t 130207111630 BigBattle
student@RHELserver:~/test$ ls -l
total 0
-rw-r--r-- 1 student student 0 1302-07-11 16:30 BigBattle
-rw-r--r-- 1 student student 0 2005-05-05 00:00 ChottoMatte
```

6.5. rm

When you no longer need a file, use **rm** to remove it. Unlike some graphical user interfaces, the command line in general does not have a *waste bin* or *trash can* to recover files. When you use rm to remove a file, the file is gone. Therefore, be careful when removing files!

```
student@RHELserver:~/test$ ls
BigBattle ChottoMatte
student@RHELserver:~/test$ rm BigBattle
student@RHELserver:~/test$ ls
ChottoMatte
```

¤ rm -i

To prevent yourself from accidentally removing a file, you can type **rm -i**.

```
student@RHELserver:~/test$ touch brel.txt
student@RHELserver:~/test$ rm -i brel.txt          #interactive mode
rm: remove regular empty file `brel.txt'? y
student@RHELserver:~/test$
```

rm -rf

By default, **rm -r** will not remove non-empty directories. However **rm** accepts several options that will allow you to remove any directory. The **rm -rf** statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with **rm -rf** (the **f** means **force** and the **r** means **recursive**) since being root implies that permissions don't apply to you. ***Warning!!! You can literally erase your entire file system by accident!***

```
student@RHELserver:~$ ls test
ChottoMatte
student@RHELserver:~$ rm test
rm: cannot remove `test': Is a directory
student@RHELserver:~$ rm -rf test
student@RHELserver:~$ ls test
ls: test: No such file or directory
```

6.6. cp

To copy a file, use **cp** with a source and a target argument. If the target is a directory, then the source files are copied to that target directory.

```
student@RHELserver:~/test$ touch FileA
student@RHELserver:~/test$ ls
FileA
student@RHELserver:~/test$ cp FileA FileB
student@RHELserver:~/test$ ls
FileA  FileB
student@RHELserver:~/test$ mkdir MyDir
student@RHELserver:~/test$ ls
FileA  FileB  MyDir
student@RHELserver:~/test$ cp FileA MyDir/
student@RHELserver:~/test$ ls MyDir/
FileA
```

cp -r

To copy complete directories, use **cp -r** (the **-r** option forces **recursive** copying of all files in all subdirectories).

```
student@RHELserver:~/test$ ls
FileA  FileB  MyDir
student@RHELserver:~/test$ ls MyDir/
FileA
student@RHELserver:~/test$ cp -r MyDir MyDirB
student@RHELserver:~/test$ ls
FileA  FileB  MyDir  MyDirB
student@RHELserver:~/test$ ls MyDirB
FileA
```

cp multiple files to directory

You can also use cp to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
cp file1 file2 dir1/file3 dir1/file55 dir2
```

¤ cp -i

To prevent cp from overwriting existing files, use the -i (for interactive) option.

```
student@RHELserver:~/test$ touch fire
student@RHELserver:~/test$ cp fire water
student@RHELserver:~/test$ cp -i fire water
cp: overwrite `water'? no
student@RHELserver:~/test$
```

¤ cp -p

To preserve permissions and time stamps from source files, use **cp -p**.

```
student@RHELserver:~$ cp /etc/passwd mypass1
```

```
student@RHELserver:~$ cp -p /etc/passwd mypass2
student@RHELserver:~$ ll my*                                    *similar to ls -l
-rw-r--r-- 1 student student  1785 2013-11-08 10:18 mypass1
-rw-r--r-- 1 student student  1785 2008-08-25 13:26 mypass2
```

6.7. mv

Use **mv** to rename a file or to move the file to another directory.

```
student@RHELserver:~/test$ touch file100
student@RHELserver:~/test$ ls
file100
student@RHELserver:~/test$ mv file100 ABC.txt
student@RHELserver:~/test$ ls
ABC.txt
student@RHELserver:~/test$
```

When you need to rename only one file then **mv** is the preferred command to use.

To move a file to another directory:

```
student@RHELserver:~/test$ touch file2move
student@RHELserver:~/test$ ls
file2move
student@RHELserver:~/test$ mv file2move ../
student@RHELserver:~/test$ ls ../
file2move
student@RHELserver:~/test$ ls
```

¤ 6.8. rename

The **rename** command can also be used but it has a more complex syntax to enable renaming of many files at once. Below are two examples, the first switches all occurrences of txt to png for all file names ending in .txt. The second example switches all occurrences of upper case ABC in lower case abc for all file names ending in .png . The following syntax will work on debian and ubuntu (prior to Ubuntu 6.10).

```
student@ubuntu:~/test$ ls
123.txt  ABC.txt
student@ubuntu:~/test$ rename 's/txt/png/' *.txt
student@ubuntu:~/test$ ls
123.png  ABC.png
student@ubuntu:~/test$ rename 's/ABC/abc/' *.png
student@ubuntu:~/test$ ls
123.png  abc.png
```

On Red Hat Enterprise Linux (and many other Linux distributions like Ubuntu 8.04), the syntax of rename is a bit different. The first example below renames all *.conf files replacing any occurrence of conf with bak. The second example renames all (*) files replacing one with ONE.

```
student@RHELserver:~/test$ ls
one.conf  two.conf
student@RHELserver:~/test$ rename conf bak *.conf
student@RHELserver:~/test$ ls
one.bak  two.bak
student@RHELserver:~/test$ rename one ONE *
student@RHELserver:~/test$ ls
```

```
ONE.bak  two.bak
student@RHELserver:~/test$
```

7. Conclusion

All operating systems provide a command shell for users to send requests to the kernel and to receive feedback. This makes the OS interactive and easier to use.

Do Practical 9B – Linux exercises after this.

END