Linux for Developers

Linux is the **kernel** of the operating system, kernel is the component that connects the hardware to the software, and manages resources such as memory, CPU, time sharing, etc.

Linux Distributions

Three major distributions are:

- Red Hat
- Debian
- SUSE

So, what does the distribution do? Well, it packages up all the software you could need in a convenient way that makes it easy to install, upgrade, remove, etc. It makes sure that all the different components of the software work well together. It's a bridge between the upstream developers, the people who actually write the code, and the end users. It makes sure that information flows in both directions. Distributions test the software under far more varied conditions that the upstream developers can do or any particular user can do, and they try to find things that are wrong and also resolve any conflicts between different software packages. Distributions also employ a lot of people to do significant work on the individual components within the distribution, the packages, the software packages, as well as the general functioning of the operating system, and even the Linux kernel itself.

Which Distribution to Choose?

You will quickly notice that technical differences are mainly about package management systems, software versions, and file locations. Once you get a grasp of those differences, it becomes relatively painless to switch from one Linux distribution to another.

Red Hat/Fedora Family

Fedora is the community distribution that forms the basis of Red Hat Enterprise Linux, CentOS, Scientific Linux and Oracle Linux. Fedora contains significantly more software than Red Hat's enterprise version. One reason for this is that a diverse community is involved in building Fedora; it is not just one company.

SUSE/openSUSE Family

The relationship between openSUSE and SUSE Linux Enterprise Server is similar to the one we just described between Fedora and Red Hat Enterprise Linux.

Debian Family

The Debian distribution is the upstream for several other distributions, including Ubuntu, Linux Mint and others. Debian is a pure open source project, and focuses on a key aspect: stability. It also provides the largest and most complete software repository to its users.

Ubuntu aims at providing a good compromise between long term stability and ease of use. Since Ubuntu gets most of its packages from Debian's unstable branch, Ubuntu also has access to a very large software repository.

Getting Help

man, info, help/--help

vi Commands

Starting, Exiting, Reading and Writing Files in vi

Command	**Description**
vi myfile	Start **vi** and edit **myfile**
vi -r myfile	Start **vi** and edit **myfile** in recovery mode from a system crash
:r file2<RET>	Read in **file2** and insert at current position
:w<RET>	Write out the file
:w myfile<RET>	Write out the file to **myfile**

:w! file2<RET>	Overwrite **file2**
:x<RET> **or** **:wq<RET>**	Exit **vi** and write out modified file
:q<RET>	Quit **vi**
:q!<RET>	Quit **vi** even though modifications have not been saved

Changing Position in vi

Command	**Description**
arrow keys	Use the arrow keys for up, down, left and right; or:
j or **<RET>**	One line down
k	One line up
h or Backspace	One character left
l or Space	One character right
0	Move to beginning of line
$	Move to end of line
w	Move to beginning of next word
b	Move back to beginning of preceding word

:0 <RET> or **1G**	Move to beginning of file
:n <RET> or **nG**	Move to line n
:$ <RET> or **G**	Move to last line in file
^f or **PageDown**	Move forward one page
^b or **PageUp**	Move backward one page
^l	Refresh and center screen

Searching for Text in vi

Command	Description
/pattern<RET>	Search forward for pattern
n	Move to next occurrence of search pattern
string<RET>	Search backward for pattern
N	Move to previous occurrence of search pattern

Changing, Adding and Deleting Text in vi

Command	Description
a	Append text after cursor; stop upon **Escape** key
A	Append text at end of current line; stop upon **Escape** key

i	Insert text before cursor; stop upon **Escape** key
I	Insert text at beginning of current line; stop upon **Escape** key
o	Start a new line below current line, insert text there; stop upon **Escape** key
O	Start a new line above current line, insert text there; stop upon **Escape** key
r	Replace character at current position
R	Replace text starting with current position; stop upon **Escape** key
x	Delete character at current position
Nx	Delete **N** characters, starting at current position
dw	Delete the word at the current position
D	Delete the rest of the current line
dd	Delete the current line
Ndd or **dNd**	Delete N lines
u	Undo the previous operation
yy	Yank (cut) the current line and put it in buffer

Nyy or **yNy**	Yank (cut) N lines and put it in buffer
p	Paste at the current position the yanked line or lines from the buffer

emacs Commands

Starting, Exiting, Reading and Writing Files in emacs

Command	**Description**
emacs myfile	Start **emacs** and edit **myfile**
Ctl-x i	Insert prompted for file at current position
Ctl-x s	Write out the file keeping current name
Ctl-x Ctl-w	Write out the file giving a new name when prompted
Ctl-x Ctl-s	Write out all files currently being worked on and exit
Ctl-x Ctl-c	Exit after being prompted if there any unwritten modified files

Changing Position in emacs

Command	**Description**
arrow keys	Use the arrow keys for up, down, left and right; or:
Ctl-n	One line down

Ctl-p	One line up
Ctl-f	One character left
Ctl-b	One character right
Ctl-a	Move to beginning of line
Ctl-e	Move to end of line
M-f	Move to beginning of next word
M-b	Move back to beginning of preceding word
M-<	Move to beginning of file
M-x goto-line n	Move to line n
M->	Move to end of file
Ctl-v or **PageDown**	Move forward one page
M-v or **PageUp**	Move backward one page
Ctl-l	Refresh and center screen

Searching for Text in emacs

Command	**Description**
Ctl-s	Search forward for prompted for pattern, or for next pattern

Ctl-r	Search backwards for prompted for pattern, or for next pattern

Changing, Adding and Deleting Text in emacs

Command	Description
Ctl-o	Insert a blank line
Ctl-d	Delete character at current position
Ctl-k	Delete the rest of the current line
Ctl-_ or **Ctl-x u**	Undo the previous operation
Ctl-space	Mark the beginning of the selected region; the end will be at the cursor position
Ctl-w	Yank (cut) the current marked region and put it in buffer
Ctl-y	Paste at the current position the yanked line or lines from the buffer

Customizing the Command Line Prompt

The default command line prompt is **$** for normal users and **#** for the root or superuser.

Customizing the command line prompt is as simple as modifying the value of the environment variable PS1. For example, to set it to display the hostname, user and current directory:

```
$ PS1="\h:\u:\w>"
c7:coop:/tmp>
```

Besides the aesthetic value of having a prettier prompt than the default value, embedding more information in the prompt can be quite useful. In the example given we have shown:

- The machine name - this is useful if you run command line windows on remote machines from your desktop; you can always tell where you are, and this can avoid many errors.
- The user name - this is particularly important if you are running as a superuser (root) and can help you avoid errors where you take what you think is a benign action and wind up crippling your system.
- The current directory - it is always important to know where you are. You certainly do not want to do something like **rm *** in the wrong directory.

Here is a table with some of the possible special characters that can be embedded in the PS1 string:

Character	**Meaning**	**Example Output**
\t	Time in HH:MM:SS	08:43:40
\d	Date in ”Weekday Month Date”	Fri Mar 12
\n	Newline	
\s	Shell name	bash
\w	Current working directory	/usr/local/bin
\W	Basename of current working directory	bin
\u	User	coop
\h	Hostname	c7
\#	Command number (this session)	43

\!	History number (in history file)	1057

Note you can embed any other string you like in the prompt.

Special Characters

A number of characters have a special meaning and cause certain actions to take place. If you want to print them directly, you usually have to prefix them with a backslash (\) or enclose them in single quotes.

Redirection Special Characters

Character	**Usage**
\#>	Redirect output descriptor (Default # = **1, stdout**)
<	Redirect input descriptor
>>	Append output
>&	Redirect **stdout** and **stderr** (equivalent to **.. > .. 2>&1**)

Compound Commands Special Characters

Character	**Usage**
|	Piping
()	Execute in a separate shell
&&	AND list
||	OR list

;	Separate commands

Expansion Special Characters

Character	**Usage**
{}	Lists
~	Usually means **$HOME**
$	Parameter substitution
‘	Back tick; used in expression evaluation (also **$()** syntax)
$(( ))	Arithmetic substitution
[]	Wildcard expressions, and conditionals

Escapes Special Characters

Character	**Usage**
****	End of line, escape sequence
’ ’	Take exactly as is
" "	Take as is, but do parameter expansion

Other Special Characters

Character	**Usage**

&	Redirection and putting task in background
#	Used for comments
***?**	Used in wildcard expansion
!	Used in history expansion

Note there are three different quoting mechanisms listed above:

- \ (as in \|; try **echo** | vs **echo** \|)
- single quotes: preserves literal value
- double quotes: same except for **$**, **‘**, and \ .

Note you can get a literal quote character by using **\’** or **\"**.

```
$ echo $HOME
/home/you
$ echo \$HOME
$HOME
$ echo '$HOME'
$HOME
$ echo "$HOME"
/home/you
```

Redirection

File descriptors:

- **0 = stdin**
- **1 = stdout**
- **2 = stderr**

less < file same as **less file** or **less 0< file**

foo > file ; redirect **stdout** (same as **foo 1> file**)

foo 2> file ; redirect **stderr**

foo >> file ; append **stdout** to **file**

foo >& file or **foo > file 2>&1**; sends **stdout** and **stderr** to a file

NOTE: **foo >>& file** does not work ; you have to do **foo >> file 2>&1.** Also Note that **foo > file 2>&1** is not the same as **foo 2>&1 > file**; the order of arguments is important.

Command Substitution and Expressions

There are two mechanisms for substituting the result of an operation into a command:

```
$ ls -l `which --skip-alias emacs`
$ ls -l $(which --skip-alias emacs)
```

The second form permits nesting, while the first form does not. Note that the first form has "backticks" (‘) not apostrophes.

Filesystem Layout

Here is a list of the main directories which should be present under /:

Main Directories

Directory	**In FHS?**	**Purpose**
/	Yes	Primary directory of the entire filesystem hierarchy
/bin	Yes	Essential executable programs that must be available in single user mode

/boot	Yes	Files needed to boot the system, such as the kernel, initrd or initramfs images, and boot configuration files and bootloader programs
/etc	Yes	System-wide configuration files
/home	Yes	User home directories, including personal settings, files, etc.
/lib	Yes	Libraries required by executable binaries in **/bin** and **/sbin**
/lib64	No	64-bit libraries required by executable binaries in **/bin** and **/sbin**, for systems which can run both 32-bit and 64-bit programs
/media	Yes	Mount points for removable media such as CD's, DVD's, USB sticks etc.
/mnt	Yes	Temporarily mounted filesystems
/opt	Yes	Optional application software packages
/proc	Yes	Virtual pseudo-filesystem giving information about the system and processes running on it; can be used to alter system parameters
/sys	No	Virtual pseudo-filesystem giving information about the system and processes running on it; can be used to alter system parameters, is similar to a device tree and is part of the Unified Device Model
/root	Yes	Home directory for the root user
/sbin	Yes	Essential system binaries

/srv	Yes	Site-specific data served up by the system; seldom used
/tmp	Yes	Temporary files; on many distributions lost across a reboot and may be a ramdisk in memory
/usr	Yes	Multi-user applications, utilities and data; theoretically read-only
/var	Yes	Variable data that changes during system operation

A system should be able to boot and go into single user, or recovery mode, with only the **/bin**, **/sbin**, **/etc**, **/lib** and **/root** directories mounted, while the contents of the **/boot** directory are needed for the system to boot in the first place.

Many of these directories (such as **/etc** and **/lib**) will generally have subdirectories associated either with specific applications or sub-systems, with the exact layout differing somewhat by Linux distribution. Two of them, **/usr** and **/var**, are relatively standardized and worth looking at.

Directories Under /usr

Directory	Purpose
/usr/bin	Non-essential binaries and scripts, not needed for single user mode; generally this means user applications not needed to start system
/usr/include	Header files used to compile applications
/usr/lib	Libraries for programs in **/usr/bin** and **/usr/sbin**
/usr/lib64	64-bit libraries for 64-bit programs in **/usr/bin** and **/usr/sbin**
/usr/sbin	Non-essential system binaries, such as system daemons

/usr/share	Shared data used by applications, generally architecture-independent
/usr/src	Source code, usually for the Linux kernel
/usr/X11R6	X Window files; generally obsolete
/usr/local	Local data and programs specific to the host; subdirectories include **bin**, **sbin**, **lib**, **share**, **include**, etc.

Directories Under /var

Directory	Purpose
/var/ftp	Used for **ftp** server base
/var/lib	Persistent data modified by programs as they run
/var/lock	Lock files used to control simultaneous access to resources
/var/log	Log files
/var/mail	User mailboxes
/var/run	Information about the running system since the last boot
/var/spool	Tasks spooled or waiting to be processed, such as print queues
/var/tmp	Temporary files to be preserved across system reboot; sometimes linked to **/tmp**

/var/www	Root for website hierarchies

Paths

When a user tries to run a program, the path is searched (from left to right) until an executable program or script is found with that name. You can see what would be found with the **which** command, as in:

```
$ which --skip-alias emacs
/usr/bin/emacs
```

Note that if there was a **/usr/local/bin/emacs**, it would be executed instead, since it is earlier in the path.

It is easy to add directories to your path, as in:

```
$ MY_BIN_DIR=$HOME/my_bin_dir
$ export PATH=$MY_BIN_DIR:$PATH
$ export PATH=$PATH:$MY_BIN_DIR
```

with the first form prepending your new directory and the second appending it to the path.

Any path which begins with / is considered absolute because it specifies the exact filesystem location. Otherwise, it is considered relative and it is implicitly assumed your current directory is prepended.

GRUB Features

Some of the most important features of GRUB are:

- You can boot into alternative operating systems. You can make this choice at boot time.
- Within a given operating system, you can choose which kernel you want to start with.
- You can use different initial ramdisk.
- You can specify different options that the system should start with.
- You can change boot parameters at boot time, without needing to change configuration files on the machine.

Monitoring and Performance Utilities

Process and Load Monitoring Utilities

Utility	Purpose
top	Process activity, dynamically updated
uptime	How long the system is running and the average load
ps	Detailed information about processes
pstree	A tree of processes and their connections
mpstat	Multiple processor usage
iostat	CPU utilization and I/O statistics
sar	Display and collect information about system activity
numastat	Information about NUMA (Non-Uniform Memory Architecture)
strace	Information about all system calls a process makes

Memory Monitoring Utilities

Utility	Purpose
free	Brief summary of memory usage
vmstat	Detailed virtual memory statistics and block I/O, dynamically updated

pmap	Process memory map

I/O Monitoring Utilities

Utility	Purpose
iostat	CPU utilization and I/O statistics
iotop	I/O statistics including per process
sar	Display and collect information about system activity
vmstat	Detailed virtual memory statistics and block I/O, dynamically updated

Network Monitoring Utilities

Utility	Purpose
netstat	Detailed networking statistics
iptraf	Gather information on network interfaces
tcpdump	Detailed analysis of network packets and traffic
wireshark	Detailed network traffic analysis

Packaging Systems

There are two main packaging systems in use in Linux systems; RPM (used for example in all variations and descendants of Red Hat Enterprise Linux, Fedora and SUSE); and deb (used for example in Debian and Ubuntu).

Operation	RPM	deb
Install a package	**rpm -i foo.rpm**	**dpkg --install foo.deb**
Install a package with dependencies from repository	**yum install foo**	**apt-get install foo**
Remove a package	**rpm -e foo.rpm**	**dpkg --remove foo.deb**
Remove a package and dependencies using a repository	**yum remove foo**	**apt-get remove foo**
Update package to a newer version	**rpm -U foo.rpm**	**dpkg --install foo.deb**
Update package using repository and resolving dependencies	**yum update foo**	**apt-get install foo**
Update entire system	**yum update**	**apt-get dist-upgrade**
Show all installed packages	**rpm -qa or yum list installed**	**dpkg --list**
Get information about an installed package including files	**rpm -qil foo**	**dpkg --listfiles foo**
Show available packages with “foo” in name	**yum list foo**	**apt-cache search foo**
Show all available packages	**yum list**	**apt-cache dumpavail foo**

What package does a file belong to?	**rpm -qf file**	**dpkg --search file**

root (super) user, su and sudo

It is possible to enter the system as the root user either for a series of operations or only for one. As a general rule, you should assume so-called root privileges only when absolutely necessary and for as short a time as necessary.

In order to temporarily sign on as another user, you can use the **su** command, as in the following examples:

```
$ su anotheruser
$ su root
$ su
```

You will be prompted for the password of the user whose name was specified. If you do not give a user name (as in the third example), root will be assumed.

The superuser session ends when you type **exit** in the shell.

If you use a naked dash as an option, as in:

```
$ su -
```

there is a subtle difference; you are signed into a login shell, which means your working directory will shift to the home directory of the account you are logging into, paths will change, etc.

If you use the **-c** option as in:

```
$ su root -c ls
$ su - root -c ls
```

you execute only one command, in this case **ls**. In the first case, this will be in the current working directory, in the second, in the root's home directory.

Suppose a normal user needs temporary root privilege to execute a command, say to put a file in a directory that requires root privilege. You can do that with the **su** command, but there is one obvious drawback; the user needs to have the root password in order to do this.

Once you have made the root password known to a normal user, you have abandoned all notions of security. While this may be an acceptable day-to-day method on a system on which you are the only normal user and you are trying to respect good system hygiene by avoiding privilege escalation except when absolutely necessary, there is a better method involving the **sudo** command.

To use **sudo** you merely have to do:

```
$ sudo -u anotheruser command
$ sudo command
```

where in the second form, the implicit user is root. While this resembles doing **su -c**, it is quite different in that the user's own password is required; **su** requires the other user's password (often that of root).

However, this will not work unless the superuser has already updated **/etc/sudoers** to grant you permission to use the **sudo** command. Furthermore, it is possible to limit exactly which subset of commands a particular user or group has access to, and to permit usage with a password prompt.

The simplest line you could add to this file would be (for user **student**):

```
student ALL=(ALL) ALL
```

which would let the user have all normal root privileges.

On all recent Linux distributions, you should not modify the file **/etc/sudoers**. Instead, there is a sub-directory **/etc/sudoers.d** in which you can create individual files for individual users.

Thus, you can simply make a short file in **/etc/sudoers.d** containing the above line, and then give it proper permissions as in:

```
$ chmod 440 /etc/sudoers.d/student
```

Note that some Linux distributions may require **chmod 400 /etc/sudoers.d/student**.

If a file named **/tmp/rootfile** is owned by root, the command:

```
$ sudo echo hello > /tmp/rootfile
```

will fail due to permission problems.

The proper way to do this would be:

```
$ sudo bash -c "echo hello > /tmp/rootfile"
```

Do you see why?

Some Linux distributions, notably Ubuntu, do not work with root user accounts in the traditional UNIX fashion.

Instead, there appears to be only a normal user account, and the same password is used to log into the system as a normal user, and to use **sudo**. In fact, there is no direct **su** command. However, the equivalent is easily accomplished through the command: **sudo su**.
