UNIT 3.
Linux Internal Commands and Builtins

Structure

3.0 Introduction
3.1 Objectives
3.2 Basic commands
ls cat, tac, rev, cp, mv, rm, rmdir, mkdir, cd, pwd, chmod, ln
3.3 Complex commands
find, `expr`, date,zdump
3.4 Time / Date commands
time, touch, cal, sleep, times
3.5 Text Processing commands
sort, tsort, diff, patch, cmp, comm, uniq, sort, join, head, tail, grep, sed, awk, wc, tr, pr, lex, yacc
3.6 File and Archiving commands
tar, locate, slocate, split, make
3.7 Communication commands
host, traceroute, ping, finger, ftp, uucp, telnet, rlogin, rsh, rcp, ssh, write, banner,
3.8 System and Administrative commands
chown, chgrp, useradd, userdel, who, su, mesg, wall, uname, du, df, hostname,halt, shutdown, reboot, finger, passwd
3.9 Networking commands
ifconfig, mount, umount, mkswap, swapon, swapoff, fdisk, fsck, e2fsck,debugfs, mkbootdisk, fdformat, umask, Aliases
3.10 Shell Related commands
echo, printf, read, let, exit, exec, true, false, help, jobs, disown, fg, bg, wait,suspend, logout,
3.11 Process commands
kill, ps,
3.12 Help commands
man, less, info, --help, apropos, whatis
3.13 Summary
3.14 Check Your Progress

3.0 Introduction

A UNIX command is a series of characters that you type. These characters consist of words that are separated by whitespace. Whitespace is the result of typing one or more Space or Tab keys. The first word is the name of the command. The rest of the words are called the command's arguments. The arguments give the command information that it might need, or specify varying behavior of the command. To invoke a command, simply type the command name, followed by arguments (if any), to indicate to the shell that you are done typing and are ready for the command to be executed, press Enter.

Standard LINUX commands make shell scripts more versatile. The power of scripts comes from coupling system commands and shell directives with simple programming constructs.

3.1 Objectives

At the end of this unit, You would be able to

- Undestand & learn about basic,complex,Time/Date,Text Processing, File and Archiving, Communication, System and Administrative, Networking, Shell Related, Process commands
- Describe the usage of Help commands

3.2 Basic Commands

Command Listing

ls

The basic file "list" command. It is all too easy to underestimate the power of this humble command.

For example, using the

`-R`, recursive option, **ls** provides a tree-like listing of a directory structure.

`-S`, sort listing by file size,

`-t`, sort by file modification time,

`-i`, show file inodes.

bash$ **ls -l** **#** long format directory listing. For each file or directory, it also shows the owner, group, size and permissions.
bash$ **ls -a** # Lists *all* the files in the directory, including hidden ones.(.)
bash$ **ls -R** # Lists the contents of each subdirectory recursively.

cat, **tac**

cat, an acronym for *concatenate*, lists a file to `stdout`. When combined with redirection (> or >>), it is commonly used to concatenate files.

```
Bash$ cat filename cat file.1 file.2 file.3 > file.33
```

`-n` option to **cat** inserts consecutive numbers before all lines of the target file(s).

`-b` option numbers only the non-blank lines.

`-v` option echoes nonprintable characters, using ^ notation.

tac

it is the inverse of *cat*, listing a file backwards from its end.

rev

reverses each line of a file, and outputs to `stdout`. This is not the same effect as **tac**, as it preserves the order of the lines, but flips each one around.

```
bash$ cat file1.txt
This is line 1.
 This is line 2.
```

```
bash$ tac file1.txt
This is line 2.
 This is line 1.
bash$ rev file1.txt
.1 enil si sihT
 .2 enil si sihT
```

cp

This is the file copy command. **`cp file1 file2`** copies `file1` to `file2`, overwriting `file2` if it already exists.


Particularly useful are the `-a` archive flag (for copying an entire directory tree) and the `-r` and `-R` recursive flags.

mv

This is the file *move* command. It is equivalent to a combination of **cp** and **rm**. It may be used to move multiple files to a directory, or even to rename a directory.


mv filename1 filename2

Renames a file, from filename1 to filename2

mv filename1 filename2 filename2 (etc) directory

Moves one or more files into the specified directory.

rm

Delete (remove) a file or files. The `-f` forces removal of even readonly files.


the recursive flag `-r`, this command removes files all the way down the directory tree.

rmdir

Remove directory. The directory must be empty of all files, including invisible "dotfiles", for this command to succeed.

mkdir

Make directory, creates a new directory. **`mkdir -p project/programs/December`** creates the named directory. The *-p* option automatically creates any necessary parent directories.

cd

The familiar **cd** change directory command finds use in scripts where execution of a command requires being in a specified directory.

```
(cd /source/directory && tar cf - . ) | (cd /dest/directory &&
tar xpvf -)
```

The `-P` (physical) option to **cd** causes it to ignore symbolic links.

cd - changes to $OLDPWD , the previous working directory.

pwd

Print Working Directory. This gives the user's (or script's) current directory. The effect is identical to reading the value of the builtin variable $PWD.

chmod

Changes the attributes of an existing file.

```
chmod +x filename
# Makes "filename" executable for all users.

chmod u+s filename
# Sets "suid" bit on "filename" permissions.
# An ordinary user may execute "filename" with same privileges
as the file's owner.
# (This does not apply to shell scripts.)
```

```
chmod 644 filename
# Makes "filename" readable/writable to owner, readable to
# others
# (octal mode).
chmod 1777 directory-name
#  Gives  everyone  read,  write,  and  execute  permission  in
directory,
# however also sets the "sticky bit".
# This means that only the owner of the directory,
# owner of the file, and, of course, root
# can delete any particular file in that directory.
```

ln

Creates links to pre-existings files. Most often used with the `-s`, symbolic or "soft" link flag. This permits referencing the linked file by more than one name and is a superior alternative to aliasing.

`ln -s oldfile newfile` links the previously existing `oldfile` to the newly created link, `newfile`.

3.3 Complex Commands

Command Listing

find

-exec *`COMMAND`* \;

Carries out *`COMMAND`* on each file that **find** scores a hit on. *`COMMAND`* terminates with \; (the ; is escaped to make certain the shell passes it to **find** literally, which concludes the command sequence). If *`COMMAND`* contains {}, then **find** substitutes the full path name of the selected file.

```
bash$ find ~/ -name '*.txt'
/home/bozo/.kde/share/apps/karm/karmdata.txt
 /home/bozo/misc/irmeyc.txt
 /home/bozo/test-scripts/1.txt
```

```
find  /etc  -exec  grep  '[0-9][0-9]*[.][0-9][0-9]*[.][0-9][0-
9]*[.][0-9][0-9]*' {} \;

# Finds all IP addresses (xxx.xxx.xxx.xxx) in /etc directory
files.
```

expr

All-purpose expression evaluator: Concatenates and evaluates the arguments according to the operation given (arguments must be separated by spaces). Operations may be arithmetic, comparison, string, or logical.

`expr 3 + 5`

returns 8

`expr 5 % 3`

returns 2

`y=`expr $y + 1``

Increment a variable, with the same effect as **`let y=y+1`** and **`y=$(($y+1))`** This is an example of ARITHMETIC EXPANSION.

`z=`expr substr $string $position $length``

Extract substring of $length characters, starting at $position.

date

Simply invoked, **date** prints the date and time to `stdout`.

zdump

Echoes the time in a specified time zone.

```
bash$ zdump EST
EST  Tue Sep 18 22:09:22 2001 EST
```

3.4 Time/Date commands

time

Outputs very verbose timing statistics for executing a command.

`time ls -l /` gives something like this:

```
0.00user 0.01system 0:00.05elapsed 16%CPU (0avgtext+0avgdata
0maxresident)k
 0inputs+0outputs (149major+27minor)pagefaults 0swaps
```

touch

Utility for updating access/modification times of a file to current system time or other specified time, but also useful for creating a new file. The command **`touch zzz`** will create a new file of zero length, named `zzz`, assuming that `zzz` did not previously exist. Time-stamping empty files in this way is useful for storing date information, for example in keeping track of modification times on a project.

cal

Prints a neatly formatted monthly calendar to `stdout`. Will do current year or a large range of past and future years.

sleep

This is the shell equivalent of a wait loop. It pauses for a specified number of seconds, doing nothing. This can be useful for timing or in processes running in the background, checking for a specific event every so often.

```
sleep 3
# Pauses 3 seconds.
```

```
The sleep command defaults to seconds, but minute, hours,
or days may also be specified.
   sleep 3 h       # Pauses 3 hours!
```

times

Gives statistics on the system time used in executing commands.

3.5 Text Processing Commands

sort

File sorter, often used as a filter in a pipe. This command sorts a text stream or file forwards or backwards, or according to various keys or character positions. Using the `-m` option, it merges presorted input files.

tsort

Topological sort, reading in pairs of whitespace-separated strings and sorting according to input patterns.

diff, **patch**

diff: flexible file comparison utility. It compares the target files line-by-line sequentially. In some applications, such as comparing word dictionaries, it may be helpful to filter the files through **sort** and **uniq** before piping them to

diff. **`diff file-1 file-2`** outputs the lines in the files that differ, with carets showing which file each particular line belongs to.

The `--side-by-side` option to **diff** outputs each compared file, line by line, in separate columns, with non-matching lines marked.

There are available various fancy frontends for **diff**, such as **spiff**, **wdiff**, **xdiff**, and **mgdiff**.

The **diff** command returns an exit status of 0 if the compared files are identical, and 1 if they differ. This permits use of **diff** in a test construct within a shell script (see below).

A common use for **diff** is generating difference files to be used with **patch** The `-e` option outputs files suitable for **ed** or **ex** scripts.

patch: flexible versioning utility. Given a difference file generated by **diff**, **patch** can upgrade a previous version of a package to a newer version. It is much more convenient to distribute a relatively small "diff" file than the entire body of a newly revised package. Kernel "patches" have become the preferred method of distributing the frequent releases of the Linux kernel.

```
patch -p1 <patch-file
# Takes all the changes listed in 'patch-file'
# and applies them to the files referenced therein.
# This upgrades to a newer version of the package.
```

Patching the kernel:

```
cd /usr/src
gzip -cd patchXX.gz | patch -p0
# Upgrading kernel source using 'patch'.
# From the Linux kernel docs "README",
# by anonymous author (Alan Cox?).
```

The **diff** command can also recursively compare directories (for the filenames present).

```
bash$ diff -r ~/notes1 ~/notes2
Only in /home/bozo/notes1: file02
 Only in /home/bozo/notes1: file03
 Only in /home/bozo/notes2: file04
```

Use **zdiff** to compare *gzipped* files.

cmp

The **cmp** command is a simpler version of **diff**, above. Whereas **diff** reports the differences between two files, **cmp** merely shows at what point they differ.

Like **diff**, **cmp** returns an exit status of 0 if the compared files are identical, and 1 if they differ. This permits use in a test construct within a shell script.

Use **zcmp** on *gzipped* files.

comm

Versatile file comparison utility. The files must be sorted for this to be useful.

comm *-options first-file second-file*

`comm file-1 file-2` outputs three columns:

- column 1 = lines unique to `file-1`
- column 2 = lines unique to `file-2`
- column 3 = lines common to both.

The options allow suppressing output of one or more columns.

- `-1` suppresses column `1`
- `-2` suppresses column `2`
- `-3` suppresses column `3`
- `-12` suppresses both columns `1` and `2`, etc.

uniq

This filter removes duplicate lines from a sorted file. It is often seen in a pipe coupled with sort.

```
cat list-1 list-2 list-3 | sort | uniq > final.list
# Concatenates the list files,
# sorts them,
# removes duplicate lines,
# and finally writes the result to an output file.
```

sort

The **`sort INPUTFILE | uniq -c | sort -nr`** command string produces a *frequency of occurrence* listing on the `INPUTFILE` file (the `-nr` options to **sort** cause a reverse numerical sort). This template finds use in analysis of log files and dictionary lists, and wherever the lexical structure of a document needs to be examined.

join

Consider this a special-purpose cousin of **paste**. This powerful utility allows merging two files in a meaningful fashion, which essentially creates a simple version of a relational database.

The **join** command operates on exactly two files, but pastes together only those lines with a common tagged field (usually a numerical label), and writes the result to `stdout`. The files to be joined should be sorted according to the tagged field for the matchups to work properly.

```
File: 1.data

100 Shoes
200 Laces
300 Socks
File: 2.data

100 $40.00
200 $1.00
300 $2.00
```

```
bash$ join 1.data 2.data
File: 1.data 2.data

 100 Shoes $40.00
 200 Laces $1.00
 300 Socks $2.00
```

head

lists the beginning of a file to `stdout` (the default is `10` lines, but this can be changed).

tail

lists the end of a file to `stdout` (the default is `10` lines). Commonly used to keep track of changes to a system logfile, using the `-f` option, which outputs lines appended to the file.

grep

A multi-purpose file search tool that uses regular expression. It was originally a command/filter in the venerable **ed** line editor, **`g/re/p`**, that is, *global - regular expression - print*.

grep *`pattern`* [*`file`*...]

Search the target file(s) for occurrences of *`pattern`*, where *`pattern`* may be literal text or a regular expression.

```
bash$ grep '[rst]ystem.$' osinfo.txt
The GPL governs the distribution of the Linux operating
system.
```

The `-i` option causes a case-insensitive search.

The `-l` option lists only the files in which matches were found, but not the matching lines.

The `-n` option lists the matching lines, together with line numbers.

```
bash$ grep -n Linux osinfo.txt
2:This is a file containing information about Linux.
6:The GPL governs the distribution of the Linux operating
system.
```

sed, **awk**

Scripting languages especially suited for parsing text files and command output. May be embedded singly or in combination in pipes and shell scripts.

sed

Non-interactive "stream editor", permits using many **ex** commands in batch mode. It finds many uses in shell scripts.

awk

Programmable file extractor and formatter, good for manipulating and/or extracting fields (columns) in structured text files. Its syntax is similar to C.

wc

wc gives a "word count" on a file or I/O stream:

```
bash $ wc /usr/doc/sed-3.02/README
20     127     838 /usr/doc/sed-3.02/README
[20 lines  127 words  838 characters]
```

`wc -w` gives only the word count.

`wc -l` gives only the line count.

`wc -c` gives only the character count.

`wc -L` gives only the length of the longest line.

Using **wc** to count how many *.txt* files are in current working directory:

```
$ ls *.txt | wc -l
#  Will  work  as  long  as  none  of  the  "*.txt"  files  have  a
```

```
linefeed in their name.
```

Using **wc** to total up the size of all the files whose names begin with letters in the range d - h

```
bash$ wc [d-h]* | grep total | awk '{print $3}'
71832
```

tr

character translation filter.

Either **tr "A-Z" "*" <filename** or **tr A-Z * <filename** changes all the uppercase letters in `filename` to asterisks (writes to `stdout`). On some systems this may not work, but **tr A-Z '[**]'** will.

The `-d` option deletes a range of characters.

```
tr -d 0-9 < filename
# Deletes all digits from the file "filename".
```

pr

Print formatting filter. This will paginate files (or `stdout`) into sections suitable for hard copy printing or viewing on screen. Various options permit row and column manipulation, joining lines, setting margins, numbering lines, adding page headers, and merging files, among other things. The **pr** command combines much of the functionality of **nl**, **paste**, **fold**, **column**, and **expand**.

pr -o 5 --width=65 fileZZZ | more gives a nice paginated listing to screen of `fileZZZ` with margins set at 5 and 65.

A particularly useful option is `-d`, forcing double-spacing (same effect as **sed -G**).

lex, **yacc**

The **lex** lexical analyzer produces programs for pattern matching. This has been replaced by the nonproprietary **flex** on Linux systems.
The **yacc** utility creates a parser, based on a set of specifications. This has been replaced by the nonproprietary **bison** on Linux systems.

3.6 File and Archiving Commands

Archiving

tar

The standard UNIX archiving utility. Originally a *Tape ARchiving* program, it has developed into a general purpose package that can handle all manner of archiving with all types of destination devices, ranging from tape drives to regular files to even `stdout`. GNU tar has long since been patched to accept gzip compression options, such as **tar czvf archive-name.tar.gz ***, which recursively archives and compresses all files in a directory tree except dotfiles in the current working directory (**$PWD**).

Some useful **tar** options:

1. `-c` create (a new archive)
2. `--delete` delete (files from the archive)
3. `-r` append (files to the archive)
4. `-t` list (archive contents)

5. `-u` update archive
6. `-x` extract (files from the archive)
7. `-z` gzip the archive

locate, **slocate**

The **locate** command searches for files using a database stored for just that purpose. The **slocate** command is the secure version of **locate** (which may be aliased to **slocate**).

`$bash locate hickson`

```
/usr/lib/xephem/catalogs/hickson.edb
```

split

Utility for splitting a file into smaller chunks. Usually used for splitting up large files in order to back them up on floppies or preparatory to e-mailing or uploading them.

make

Utility for building and compiling binary packages. This can also be used for any set of operations that is triggered by incremental changes in source files.

The **make** command checks a `Makefile`, a list of file dependencies and operations to be carried out.

3.7 Communications Commands

host

Searches for information about an Internet host by name or ip address, using dns.

traceroute

Trace the route taken by packets sent to a remote host. This command works within a LAN, WAN, or over the Internet. The remote host may be specified by an IP address. traceroute 192.168.0.1 <= ip address

ping

Broadcast an "ICMP ECHO_REQUEST" packet to other machines, either on a local or remote network. This is a diagnostic tool for testing network connections, and it should be used with caution.

```
bash$ ping localhost   or     ping 127.0.0.1
PING localhost.localdomain (127.0.0.1) from 127.0.0.1 : 56(84)
bytes of data.
 Warning: time of day goes back, taking countermeasures.
 64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=0
ttl=255 time=709 usec
 64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=1
ttl=255 time=286 usec
 --- localhost.localdomain ping statistics ---
 2 packets transmitted, 2 packets received, 0% packet loss
 round-trip min/avg/max/mdev = 0.286/0.497/0.709/0.212 ms
```

Finger

Retrieve **information about a particular user on a network**. Optionally, this command can display the user's `~/.plan`, `~/.project`, and `~/.forward` files, if present.

```
bash$ finger bozo
Login: bozo                             Name: Bozo Bozeman
 Directory: /home/bozo                   Shell: /bin/bash
 On since Fri Aug 31 20:13 (MST) on tty1    1 hour 38 minutes
idle
 On since Fri Aug 31 20:13 (MST) on pts/0   12 seconds idle
 On since Fri Aug 31 20:13 (MST) on pts/1
 On since Fri Aug 31 20:31 (MST) on pts/2    1 hour 16 minutes
idle
 No mail.
 No Plan.
```

ftp

Utility and protocol for uploading / downloading files to / from a remote host. An ftp session can be automated in a script .

uucp

UNIX to UNIX copy. This is a communications package for transferring files between UNIX servers. A shell script is an effective way to handle a **uucp** command sequence.

telnet

Utility and protocol for connecting to a remote host.

rlogin

`Remote login`, initates a session on a remote host.

Rsh

`Remote shell`, executes command(s) on a remote host.

Rcp

`Remote copy`, copies files between two different networked machines.

Ssh

`Secure shell`, logs onto a remote host and executes commands there. This secure replacement for **telnet**, **rlogin**, **rcp**, and **rsh** uses identity authentication and encryption.

write

This is a utility for terminal-to-terminal communication. It allows sending lines from your terminal (console or xterm) to that of another user. The mesg command may, of course, be used to disable write access to a terminal

```
root# write  guest     #  where guest is a login user
```

banner

Prints arguments as a large vertical banner to `stdout`, using an ASCII character (default '#'). This may be redirected to a printer for hardcopy.

3.8 System and Administrative Commands

The startup and shutdown scripts in `/etc/rc.d` illustrate the uses (and usefulness) of many of these comands. These are usually invoked by root and used for system maintenance or emergency filesystem repairs. Use with caution, as some of these commands may damage your system if misused.

chown, **chgrp**

The **chown** command changes the ownership of a file or files. This command is a useful method that *root* can use to shift file ownership from one user to another. An ordinary user may not change the ownership of files, not even her own files.

```
root# chown bozo *.txt
```

The **chgrp** command changes the *group* ownership of a file or files. You must be owner of the file(s) as well as a member of the destination group (or *root*) to use this operation.

```
root# chgrp rao roop.c
```

useradd, **userdel**

The **useradd** administrative command adds a user account to the system and creates home directory for that particular user, if so specified. The corresponding **userdel** command removes a user account from the system and deletes associated files.

```
root# useradd mahesh
```

who

Show all users logged on to the system.

```
bash$ who
bozo  tty1     Apr 27 17:45
 bozo  pts/0    Apr 27 17:46
 bozo  pts/1    Apr 27 17:47
 bozo  pts/2    Apr 27 17:49
```

The `-m` gives detailed information about only the current user. Passing any two arguments to **who** is the equivalent of **who -m**, as in **who am i** or **who The Man**. **whoami** is similar to **who -m**, but only lists the user name.

su

Runs a program or script as a *s*ubstitute *u*ser. **su rjones** starts a shell as user *rjones*. A naked **su** defaults to *root*.

mesg

Enables or disables write access to the current user's terminal. Disabling access would prevent another user on the network to write to the terminal.

wall

This is an acronym for "write all", i.e., sending a message to all users at every terminal logged into the network. It is primarily a system administrator's tool, useful, for example, when warning everyone that the system will shortly go down due to a problem.

```
bash$ wall System going down for maintenance in 5 minutes!
Broadcast message from bozo (pts/1) Sun Jul 8 13:53:27
2001...

 System going down for maintenance in 5 minutes!
```

Note: If write access to a particular terminal has been disabled with **mesg**, then **wall** cannot send a message to it.

uname

Output system specifications (OS, kernel version, etc.) to `stdout`. Invoked with the `-a` option, gives verbose system info.

```
bash$ uname -a
Linux localhost.localdomain 2.2.15-2.5.0 #1 Sat Feb 5 00:13:43
EST 2000 i686 unknown
```

du

Show (disk) file usage, recursively. Defaults to current working directory, unless otherwise specified.

```
bash$ du -ach
 1.0k    ./wi.sh
 1.0k    ./tst.sh
 1.0k    ./random.file
 6.0k    .
 6.0k    total
```

df

Shows filesystem usage in tabular form, also disk free space available.

```
bash$ df
Filesystem               1k-blocks         Used  Available  Use%
Mounted on
 /dev/hda5                273262     92607    166547  36% /
 /dev/hda8                222525    123951     87085  59% /home
 /dev/hda7               1408796   1075744    261488  80% /usr
```

hostname

Lists the system's host name. This command sets the host name in an `/etc/rc.d` setup script (`/etc/rc.d/rc.sysinit` or similar). It is equivalent to **uname -n**, and a counterpart to the $HOSTNAME internal variable.

```
bash$ hostname
localhost.localdomain
```

halt, **shutdown**, **reboot**

Command set to shut the system down, usually just prior to a power down.

finger

Where the LINUX command 'who' gives information of people currently using the system, finger is a LINUX command that provides you with more complete information concerning all with usernames for the system. If finger is used without an argument, it will give information concerning users currently logged in.

```
 [sbl@happy]:[~]$ finger
        Login     Name             Tty  Idle  Login Time
        danp      Dan Peterson      qb  1:21  Mar 28 12:26
        jon       Jonathan Little   r1    32  Mar 28 11:29
        morbius   Morbius           r6    27  Mar 28 12:27
        sbl       sammy truong      p4        Mar 28 12:25
```

passwd

The command passwd is used to change your password. The system does not print passwords, so there is a redundant check to be sure it was entered correctly. When choosing a password, the following points should be considered:

* Each password should have at least six characters
* Each password should contain at least two letters and one number or special character
* Passwords are generally case insensitive (do not use capital letters)
* Passwords should be changed on a regular (at least six weeks) basis

For example (remember characters for password will not be printed on the computer terminal)

```
sbl@asdf [~]$ passwd
              Changing password for sbl
              Old password:
              Enter the new password (minimum of 6 characters)
              New password:
              Re-enter new password:
              They don't match; try again.
              New password:
              Re-enter new password:
              Password changed.
sbl@asdf [~]$
```

3.9 Networking Commands

ifconfig

Network interface configuration and tuning utility. It is most often used at bootup to set up the interfaces, or to shut them down when rebooting.

mount

Mount a filesystem, usually on an external device, such as a floppy or CDROM. The file `/etc/fstab` provides a handy listing of available filesystems, partitions, and devices, including options, that may be automatically or manually mounted. The file `/etc/mtab` shows the currently mounted filesystems and partitions (including the virtual ones, such as `/proc`).**mount -a** mounts all filesystems and partitions listed in `/etc/fstab`, except those with a `noauto` option. At bootup, a startup script in `/etc/rc.d` (`rc.sysinit` or something similar) invokes this to get everything mounted.

$ **mount -t iso9660** /dev/cdrom /mnt/cdrom # Mounts CDROM

$ mount /mnt/cdrom # Shortcut, if /mnt/cdrom listed in /etc/fstab

umount

Unmount a currently mounted filesystem. Before physically removing a previously mounted floppy or CDROM disk, the device must be **umount**ed, else filesystem corruption may result.

$ **umount /mnt/cdrom**

You may now press the eject button and safely remove the disk.

mkswap

Creates a swap partition or file. The swap area must subsequently be enabled with **swapon**.

swapon, **swapoff**

Enable / disable swap partitition or file. These commands usually take effect at bootup and shutdown.

fdisk

Create or change a partition table on a storage device, usually a hard drive. This command must be invoked as root.

```
root# fdisk /dev/hda # where hda is the first hard disk present in the system.
```

fsck, **e2fsck**, **debugfs**

Filesystem check, repair, and debug command set.

fsck: a front end for checking a UNIX filesystem (may invoke other utilities). The actual filesystem type generally defaults to ext2.

```
root# fsck /dev/hda1 # where hda1 is the first linux native partition where kernel files exist.
```

e2fsck: ext2 filesystem checker.

debugfs: ext2 filesystem debugger.

Note: All of these commands should be invoked as root, and they can damage or destroy a filesystem if misused. If something goes wrong, you may destroy an existing filesystem.

mkbootdisk

Creates a boot floppy which can be used to bring up the system if, for example, the MBR (master boot record) becomes corrupted. The **mkbootdisk** command is actually a Bash script, written by Erik Troan, in the `/sbin` directory.

fdformat

Perform a low-level format on a floppy disk.

umask

User file creation MASK. Limit the default file attributes for a particular user. All files created by that user take on the attributes specified by **umask**. The (octal) value passed to **umask** defines the the file permissions *disabled*. For example, **umask 022** ensures that new files will have at most 755 permissions (777 NAND 022).The usual practice is to set the value of **umask** in `/etc/profile` and/or `~/.bash_profile`.

Aliases

A Bash *alias* is essentially nothing more than a keyboard shortcut, an abbreviation, a means of avoiding typing a long command sequence. If, for example, we include alias lm="ls -l | more" in the **~/.bashrc file**, then each `lm` typed at the command line will automatically be replaced by a ls -l | more. This can save a great deal of typing at the command line and avoid having to remember complex combinations of commands and options. Setting alias rm="rm -i" (interactive mode delete) it can prevent inadvertently losing important files.

```
alias ls='ls -aF --color'
alias haha='ls -aF --color'
alias sl='ls -aF --color'
```

The **unalias** command removes a previously set *alias*.

3.10 Shell Related Commands

echo

prints (to `stdout`) an expression or variable.

```
echo Hello
echo $a
```

An **echo** requires the `-e` option to print escaped characters.

Normally, each **echo** command prints a terminal newline, but the `-n` option suppresses this.

An **echo**, in combination with command substitution can set a variable.

```
a=`echo "HELLO" | tr A-Z a-z`
```

.

printf

The **printf**, formatted print, command is an enhanced **echo**. It is a limited variant of the C language `printf`, and the syntax is somewhat different.

printf *`format-string`*... *`parameter`*...

This is the Bash builtin version of the `/bin/printf` or `/usr/bin/printf` command. See the **printf** manpage (of the system command) for in-depth coverage.

Older versions of Bash may not support **printf**.

Example : printf in action

```
# printf demo

PI=3.14159265358979
DecimalConstant=31373
Message1="Greetings,"
Message2="Earthling."

echo

printf "Pi to 2 decimal places = %1.2f" $PI
echo
printf "Pi to 9 decimal places = %1.9f" $PI  # It even rounds
off correctly.
printf  "\n"  #Prints  a  line  feed,  equivalent  to  echo.
printf "Constant = \t%d\n" $DecimalConstant # Inserts tab (\t)
printf "%s %s \n" $Message1 $Message2
exit 0
```

read

"Reads" the value of a variable from `stdin`, that is, interactively fetches input from the keyboard. The `-a` option lets **read** get array variables.

Example : Variable assignment, using read

```
echo -n "Enter the value of variable 'var1': "
# The -n option to echo suppresses newline.
read var1
# Note no '$' in front of var1, since it is being set.
echo "var1 = $var1"
exit 0
```

let

The **let** command carries out arithmetic operations on variables. In many cases, it functions as a less complex version of expr.

Example : Letting let do some arithmetic.

```
#!/bin/bash

echo

let a=11          # Same as 'a=11'
let a=a+5         # Equivalent to  let "a = a + 5"
                  #  (double  quotes  and  spaces  make  it  more
readable)
```

```
echo "11 + 5 = $a"

let "a <<= 3"     # Equivalent to  let "a = a << 3"
echo "\"\$a\" (=16) left-shifted 3 places = $a"

let "a /= 4"      # Equivalent to  let "a = a / 4"
echo "128 / 4 = $a"

let "a -= 5"      # Equivalent to  let "a = a - 5"
echo "32 - 5 = $a"

let "a = a * 10"  # Equivalent to  let "a = a * 10"
echo "27 * 10 = $a"

let "a %= 8"      # Equivalent to  let "a = a % 8"
echo "270 modulo 8 = $a  (270 / 8 = 33, remainder $a)"
```

exit

Unconditionally terminates a script. The **exit** command may optionally take an integer argument, which is returned to the shell as the exit status of the script. It is a good practice to end all but the simplest scripts with an **`exit 0`**, indicating a successful run.

exec

This shell builtin replaces the current process with a specified command. Normally, when the shell encounters a command, it forks off a child process to actually execute the command. Using the **exec** builtin, the shell does not fork, and the command exec'ed replaces the shell. When used in a script, therefore, it forces an exit from the script when the **exec**'ed command terminates. For this reason, if an **exec** appears in a script, it would probably be the final command.

Example : Effects of exec

```
exec echo "Exiting \"$0\"."   # Exit from script.
```

true

A command that returns a successful (zero) exit status, but does nothing else.

```
# Endless loop
while true   # alias for ":"
do
   operation-1
   operation-2
   ...
   operation-n
   # Need a way to break out of loop.
Done
```

false

A command that returns an unsuccessful exit status, but does nothing else.

```
# Null loop
while false
do
   # The following code will not execute.
   operation-1
   operation-2
   ...
   operation-n
   # Nothing happens!
done
```

help

help COMMAND looks up a short usage summary of the shell builtin COMMAND. This is the counterpart to whatis, but for builtins.

```
bash$ help exit
exit: exit [n]
    Exit the shell with a status of N.  If N is omitted, the
exit status
    is that of the last command executed.
```

jobs

Lists the jobs running in the background, giving the job number. Not as useful as **ps**.

It is all too easy to confuse *jobs* and *processes*. Certain builtins, such as **kill**, **disown**, and **wait** accept either a job number or a process number as an argument. The **fg**, **bg** and **jobs** commands accept only a job number.

```
bash$ sleep 100 &
[1] 1384

bash $ jobs
[1]+  Running                 sleep 100 &
```

"1" is the job number (jobs are maintained by the current shell), and "1384" is the process number (processes are maintained by the system). To kill this job/process, either a **kill %1** or a **kill 1384** works.

disown

Remove job(s) from the shell's table of active jobs.

fg, **bg**

The **fg** command switches a job running in the background into the foreground. The **bg** command restarts a suspended job, and runs it in the background. If no job number is specified, then the **fg** or **bg** command acts upon the currently running job.

wait

Stop script execution until all jobs running in background have terminated, or until the job number or process id specified as an option terminates. Returns the exit status of waited-for command.

You may use the **wait** command to prevent a script from exiting before a background job finishes executing (this would create a dreaded orphan process).

suspend

This has a similar effect to **Control-Z**, but it suspends the shell (the shell's parent process should resume it at an appropriate time).

logout

Exit a login shell, optionally specifying an exit status.

3.11 Process Commands

kill

Forcibly terminate a process by sending it an appropriate *terminate* signal.

`kill -l` lists all the signals. A **`kill -9`** is a "sure kill", which will usually terminate a process that stubbornly refuses to die with a plain **kill**. Sometimes, a **`kill -15`** works. A "zombie process", that is, a process whose parent has terminated, cannot be killed (you can't kill something that is already dead), but **init** will usually clean it up sooner or later.

ps

*P*rocess *S*tatistics: lists currently executing processes by owner and PID (process id).

```
bash$   ps –a gives present status of the login with pid of different processes
```

3.12 Help Commands

man, less

Almost every command in Linux has online help available from the command line, through the "`man`" (manual) command. Less is also similar to man command.

```
bash$   man ls  # detail information about ls command
```

Info

Another source of online help is the "info" command. Some Linux commands may supply both "man" and "info" documentation. As a general rule, "info" documentation is more verbose and descriptive, like a user guide, while "man" documentation is more like a reference manual, giving lists of options and parameters, and the meaning of each.

```
bash$   info ls  # informatory information about ls with examples
```

--help

Most (but not all) programs have a `--help` option which displays a very short description of its main options and parameters. Try typing "`ls --help`" to see. This will produce more than one screenful of information, so you'll have to use the terminal's scrollbar to see what was displayed.

apropos, whatis

this is usefull to know the other kinds of related commands

```
bash$   apropos dos  # lists available commands containing the characters dos
bash$   whatis ls   # a single line information of ls command
```

3.13 Summary

These commands will work with most (if not all) distributions of Linux as well as most (?) implementations of Unix. They're the commands that everybody knows. To be able to survive in Linux, you should know these. There aren't always handy-dandy tools for X that shield you, especially if you're managing your own system, stuff often goes wrong and you're forced to work with the bare minimum.

One of the most important things to know about UNIX or any computer system is how to get help when you don't know how to use a command. Many commands will give you a usage message if you incorrectly enter the command. This message shows you the correct syntax for the command. This can be a quick reminder of the arguments and their order. For many commands, you can get the usage message by using the option -?. The usage message often does not give you any semantic information.

Use the man command and learn more about any of the commands. With this as a start, you should be comfortable with the basics.

The man command has a very useful option, especially for users who are unfamiliar with UNIX. This option is -k and is used to find all commands that have to do with a word you supply following the -k.

3.14 Check Your Progress

I. Choose the correct answer

1) tac _____
   a) concatenates 2 files
   b) listing a file backwards from its end
   c) reverses each line of file
   d) none of the above
2) rmdir remove directory iff _____.
   a) Directory must be empty of all files
   b) Directory mayn't be empty
   c) Both a) & b)
   d) None of the above
3) touch command is used for _____.
   a) Updating access/modification times of a file to current system time
   b) Creating a new file
   c) Both a) & b)
   d) None of the above
4) Diff reports _____
   a) Differences between two files
   b) At what point the files differ
   c) Removes duplicate lines
   d) None of the above.
5) egrep an acronym for _____.
   a) evaluation global regular expression print
   b) extensible global regular expression print
   c) extended global regular expression print
   d) none of the above.
6) traceroute works within _____

a) LAN
b) WAN
c) Over the internet
d) All of the above

7) If finger is used without an argument _____
   a) it gives information concerning users currently logged in
   b) it gives information about a particular user on a network
   c) it gives information about all users
   d) none of the above
8) protocol used for connecting to a remote host _____.
   a) ftp
   b) telnet
   c) tcp/ip
   d) none of the above
9) df command is used for_____ purpose.
   a) Communication
   b) System & administrative
   c) Networking
   d) File & archiving
10) Command used for shutting down the system _____
   a) halt
   b) exit
   c) quit
   d) logout
11) The following are Network related Commands _____
   a) Wall
   b) ftp
   c) host
   d) none of the above

II. Say True or False

1. mv is equivalent to a combination of cp & rm. True/False
2. mkdir –p creates a new directory with necessary parent directories. True/False
3. expr concatenates and evaluates the arguments according to the given operation. True/False
4. join pastes together only those lines with a common numerical label. True/False
5. tr a-z * < filename changes all the lowercase letters in filename to asterisks. True/False
6. exec shell builtin replaces the present process with a specified command. True/False
7. kill command resumes a process. True/False

III. Essay Type Questions

1. Write a short notes on the following with relevant options.
   a. cat & tac
   b. chmod
   c. zdump
   d. touch

    e. sort
    f. grep
2. Explain any three Text Processing Commands.
3. Write in detial about tar command with the relvant options.
4. Give the differentiation between locate & slocate.
5. Give a short note on the following communication commands.
    a. Ping
    b. Finger
6. Explain all the system & administrative commands.
7. List the differences between the following.
    a. mv & cp
    b. tac & rev
    c. expr & let
    d. cmp & diff
8. Explain any 5 impordant Network related commands.

IV. Further readings and other activities

1. Get more information using man or --help or any help command for cat, ls, cp, diff,
2. Work and expertise all the commands given in the chapter with different options and by the virtue of which know, which command is to be used for different situations of jobs.
3. Get more information on commands in the text book Principles of Unix/ Linux by Sumitabha Das.

Reference e-mails: raomvp@yahoo.com roopasindhe@lycos.com
URL / Web Site: http://www.raomvp.bravepages.com

Good-Luck