Frequently used commands

http://www.thegeekstuff.com/2010/11/50-linux-commands/

http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds1/cp.htm

Table of Contents

ls command

Open Last Edited File Using ls -t

To open the last edited file in the current directory use the combination of ls, head and vi commands as shown below.

ls -t sorts the file by modification time, showing the last edited file first. **head -1** picks up this first file.

```
$ vi first-long-file.txt
$ vi second-long-file.txt

$ vi `ls -t | head -1`
```

[**Note**: This will open the last file you edited (i.e second-long-file.txt)]

Display One File Per Line Using ls -1

To show single entry per line, use -1 option as shown below.

```
$ ls -1
bin
boot
cdrom
dev
etc
home
initrd
initrd.img
lib
```

Display All Information About Files/Directories Using ls -l

To show long listing information about the file/directory.

```
 $ ls -l
-rw-r----- 1 ramesh team-dev 9275204 Jun 13 15:27 mthesaur.txt.gz
```

-

1st Character – File Type: First character specifies the type of the file.
In the example above the hyphen (-) in the 1st character indicates that this is a normal file. Following are the possible file type options in the 1st character of the ls -l output.

Field	Explanation
-	normal file
d	directory
s	socket file
l	link file

•

Field 1 – File Permissions: Next 9 character specifies the files permission. Each 3 characters refers to the read, write, execute permissions for user, group and world In this example, -rw-r —– indicates read-write permission for user, read permission for group, and no permission for others.
Field 2 – Number of links: Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file.
Field 3 – Owner: Third field specifies owner of the file. In this example, this file is owned by username 'ramesh'.
Field 4 – Group: Fourth field specifies the group of the file. In this example, this file belongs to "team-dev' group.
Field 5 – Size: Fifth field specifies the size of file. In this example, '9275204′ indicates the file size.
Field 6 – Last modified date & time: Sixth field specifies the date and time of the last modification of the file. In this example, 'Jun 13 15:27′ specifies the last modification time of the file.
Field 7 – File name: The last field is the name of the file. In this example, the file name is mthesaur.txt.gz.

Display File Size in Human Readable Format Using ls -lh

Use **ls -lh** (h stands for human readable form), to display file size in easy to read format. i.e M for MB, K for KB, G for GB.

```
$ ls -l
-rw-r----- 1 ramesh team-dev 9275204 Jun 12 15:27 arch-linux.txt.gz*

$ ls -lh
-rw-r----- 1 ramesh team-dev 8.9M Jun 12 15:27 arch-linux.txt.gz
```

Display Directory Information Using ls -ld

When you use "ls -l" you will get the details of directories content. But if you want the details of directory then you can use -d option as., For example, if you use ls -l /etc will display all the files under etc directory. But, if you want to display the information about the /etc/ directory, use -ld option as shown below.

```
$ ls -l /etc
total 3344
-rw-r--r--   1 root root   15276 Oct  5  2004 a2ps.cfg
-rw-r--r--   1 root root    2562 Oct  5  2004 a2ps-site.cfg
drwxr-xr-x   4 root root    4096 Feb  2  2007 acpi
-rw-r--r--   1 root root      48 Feb  8  2008 adjtime
drwxr-xr-x   4 root root    4096 Feb  2  2007 alchemist

$ ls -ld /etc
drwxr-xr-x 21 root root 4096 Jun 15 07:02 /etc
```

Order Files Based on Last Modified Time Using ls -lt

To sort the file names displayed in the order of last modification time use the -t option. You will be finding it handy to use it in combination with -l option.

$ **ls -lt**
total 76
drwxrwxrwt 14 root root 4096 Jun 22 07:36 tmp
drwxr-xr-x 121 root root 4096 Jun 22 07:05 etc
drwxr-xr-x 13 root root 13780 Jun 22 07:04 dev
drwxr-xr-x 13 root root 4096 Jun 20 23:12 root
drwxr-xr-x 12 root root 4096 Jun 18 08:31 home
drwxr-xr-x 2 root root 4096 May 17 21:21 sbin
lrwxrwxrwx 1 root root 11 May 17 20:29 cdrom -> media/cdrom
drwx------ 2 root root 16384 May 17 20:29 lost+found
drwxr-xr-x 15 root root 4096 Jul 2 2008 var

Order Files Based on Last Modified Time (In Reverse Order) Using ls -ltr

To sort the file names in the last modification time in reverse order. This will be showing the last edited file in the last line which will be handy when the listing goes beyond a page. This is my default ls usage. Anytime I do ls, I always use ls -ltr as I find this very convenient.

$ **ls -ltr**

total 76
drwxr-xr-x 15 root root 4096 Jul 2 2008 var
drwx------ 2 root root 16384 May 17 20:29 lost+found
lrwxrwxrwx 1 root root 11 May 17 20:29 cdrom -> media/cdrom
drwxr-xr-x 2 root root 4096 May 17 21:21 sbin
drwxr-xr-x 12 root root 4096 Jun 18 08:31 home
drwxr-xr-x 13 root root 4096 Jun 20 23:12 root
drwxr-xr-x 13 root root 13780 Jun 22 07:04 dev
drwxr-xr-x 121 root root 4096 Jun 22 07:05 etc
drwxrwxrwt 14 root root 4096 Jun 22 07:36 tmp

Display Hidden Files Using ls -a (or) ls -A

To show all the hidden files in the directory, use '-a option'. Hidden files in Unix starts with '.' in its file name.

$ **ls -a**
[rnatarajan@asp-dev ~]$ ls -a
. Debian-Info.txt
.. CentOS-Info.txt
.bash_history Fedora-Info.txt
.bash_logout .lftp
.bash_profile libiconv-1.11.tar.tar
.bashrc libssh2-0.12-1.2.el4.rf.i386.rpm

It will show all the files including the '.' (current directory) and '..' (parent directory). To show the hidden files, but not the '.' (current directory) and '..' (parent directory), use option -A.

$ **ls -A**

```
Debian-Info.txt              Fedora-Info.txt
CentOS-Info.txt              Red-Hat-Info.txt
.bash_history                SUSE-Info.txt
.bash_logout                 .lftp
.bash_profile                libiconv-1.11.tar.tar
.bashrc                      libssh2-0.12-1.2.el4.rf.i386.rpm
```
[**Note**: . and .. are not displayed here]

Display Files Recursively Using ls -R

```
$ ls  /etc/sysconfig/networking
devices  profiles

$ ls  -R /etc/sysconfig/networking
/etc/sysconfig/networking:
devices  profiles

/etc/sysconfig/networking/devices:

/etc/sysconfig/networking/profiles:
default

/etc/sysconfig/networking/profiles/default:
```

To show all the files recursively, use -R option. When you do this from /, it shows all the unhidden files in the whole file system recursively.

Display File Inode Number Using ls -i

Sometimes you may want to know the inone number of a file for internal maintenance. Use -i option as shown below to display inone number. Using inode number you can remove files that has special characters in it's name as explained in the example#6 of the find command article.

```
$ ls -i /etc/xinetd.d/
279694 chargen      279724 cups-lpd  279697 daytime-udp
279695 chargen-udp  279696 daytime   279698 echo
```

Hide Control Characters Using ls -q

To print question mark instead of the non graphics control characters use the -q option.

```
ls -q
```

Display File UID and GID Using ls -n

Lists the output like -l, but shows the uid and gid in numeric format instead of names.

```
$ ls -l ~/.bash_profile
-rw-r--r--  1 ramesh ramesh 909 Feb  8 11:48 /home/ramesh/.bash_profile
$ ls -n ~/.bash_profile
-rw-r--r--  1 511 511 909 Feb  8 11:48 /home/ramesh/.bash_profile
```

[Note: This display 511 for uid and 511 for gid]

Visual Classification of Files With Special Characters Using ls -F

Instead of doing the 'ls -l' and then the checking for the first character to determine the type of file. You can use -F which classifies the file with different special character for different kind of files.

```
$ ls -F
Desktop/  Documents/  Ubuntu-App@  firstfile  Music/  Public/  Templates/
```

Thus in the above output,

/ – directory.
nothing – normal file.
@ – link file.
* – Executable file

Visual Classification of Files With Colors Using ls -F

Recognizing the file type by the color in which it gets displayed is an another kind in classification of file. In the above output directories get displayed in blue, soft links get displayed in green, and ordinary files gets displayed in default color.

```
$ ls --color=auto
Desktop  Documents Examples firstfile Music  Pictures  Public  Templates  Videos
```

Useful ls Command Aliases

You can take some required ls options in the above, and make it as aliases. We suggest the following.

Long list the file with size in human understandable form.

```
alias ll="ls -lh"
```

Classify the file type by appending special characters.

```
alias lv="ls -F"
```

Classify the file type by both color and special character.

```
alias ls="ls -F –color=auto"
```

cd command examples

The **cd** command sets the current working directory of a process. The user must have execute (search) permission in the specified directory.

To change the current working directory to the login (home) directory

```
cd
```

To change to an arbitrary directory

```
cd /usr/include
```

This changes the current directory to /usr/include.

To go down one level of the directory tree

```
cd sys
```

If the current directory is /usr/include and it contains a subdirectory named sys, then /usr/include/sys becomes the current directory.

To go up one level of the directory tree

```
cd ..
```

The special file name, .. (dot-dot), refers to the directory immediately above the current directory.

Use “cd -” to toggle between the last two directories

Use “shopt -s cdspell” to automatically correct mistyped directory names on cd

Use cd alias to navigate up the directory effectively

When you are navigating up a very long directory structure, you may be using cd ..\..\ with multiple ..\’s depending on how many directories you want to go up as shown below.

```
# mkdir -p /tmp/very/long/directory/structure/that/is/too/deep

# cd /tmp/very/long/directory/structure/that/is/too/deep
# pwd
/tmp/very/long/directory/structure/that/is/too/deep

# cd ../../../../
# pwd
/tmp/very/long/directory/structure
```

Instead of executing cd ../../../.. to navigate four levels up, use one of the following alias methods:

Navigate up the directory using ..n : In the example below, ..4 is used to go up 4 directory level, ..3 to go up 3 directory level, ..2 to go up 2 directory level. Add the following alias to the .bash_profile and re-login.

```
alias ..="cd .."
alias ..2="cd ../.."
alias ..3="cd ../../.."
alias ..4="cd ../../../.."
alias ..5="cd ../../../../.."

# cd /tmp/very/long/directory/structure/that/is/too/deep
#..4
[Note: use ..4 to go up 4 directory level]
# pwd
/tmp/very/long/directory/structure/
```

Navigate up the directory using only dots: In the example below, ….. (five dots) is used to go up 4 directory level. Typing 5 dots to go up 4 directory structure is really easy to remember, as when you type the first two dots, you are thinking "going up one directory", after that every additional dot, is to go one level up. So, use …. (four dots) to go up 3 directory level and .. (two dots) to go up 1 directory level. Add the following alias to the .bash_profile and re-login for the ….. (five dots) to work properly.

```
alias ..="cd .."
alias ...="cd ../.."
alias ....="cd ../../.."
alias .....="cd ../../../.."
alias ......="cd ../../../../.."

# cd /tmp/very/long/directory/structure/that/is/too/deep
# .....
[Note: use ..... (five dots) to go up 4 directory level]
# pwd
/tmp/very/long/directory/structure/
```

Navigate up the directory using cd followed by consecutive dots: In the example below, cd….. (cd followed by five dots) is used to go up 4 directory level. Making it 5 dots to go up 4 directory structure is really easy to remember, as when you type the first two dots, you are thinking "going up one directory", after that every additional dot, is to go one level up. So, use cd…. (cd followed by four dots) to go up 3 directory level and cd… (cd followed by three dots) to go up 2 directory level. Add the following alias to the .bash_profile and re-login for the above cd….. (five dots) to work properly.

```
alias cd..="cd .."
alias cd...="cd ../.."
alias cd....="cd ../../.."
alias cd.....="cd ../../../.."
alias cd......="cd ../../../../.."

# cd /tmp/very/long/directory/structure/that/is/too/deep
# cd.....
[Note: use cd..... to go up 4 directory level]
# pwd
/tmp/very/long/directory/structure
```

Perform mkdir and cd using a single command

Sometimes when you create a new directory, you may cd to the new directory immediately to perform some work as shown below.

```
# mkdir -p /tmp/subdir1/subdir2/subdir3
# cd /tmp/subdir1/subdir2/subdir3
# pwd
/tmp/subdir1/subdir2/subdir3
```

Wouldn't it be nice to combine both mkdir and cd in a single command? Add the following to the .bash_profile and re-login.

```
function mkdircd () { mkdir -p "$@" && eval cd "\"\$$#\""; }
```

Now, perform both mkdir and cd at the same time using a single command as shown below:

```
# mkdircd /tmp/subdir1/subdir2/subdir3
[Note: This creates the directory and cd to it automatically]
# pwd
/tmp/subdir1/subdir2/subdir3
```

Use "cd -" to toggle between the last two directories

You can toggle between the last two current directories using cd – as shown below.

```
# cd /tmp/very/long/directory/structure/that/is/too/deep
# cd /tmp/subdir1/subdir2/subdir3

# cd -
# pwd
/tmp/very/long/directory/structure/that/is/too/deep

# cd -
# pwd
/tmp/subdir1/subdir2/subdir3

# cd -
# pwd
/tmp/very/long/directory/structure/that/is/too/deep
```

pwd command

pwd is Present working directory. What else can be said about the good old pwd who has been printing the current directory name for ages.

man command

Display the man page of a specific command.

```
$ man crontab
```

When a man page for a command is located under more than one section, you can view the

man page for that command from a specific section as shown below.

```
$ man SECTION-NUMBER commandname
```

Following 8 sections are available in the man page.

1. General commands
2. System calls
3. C library functions
4. Special files (usually devices, those found in /dev) and drivers
5. File formats and conventions
6. Games and screen savers
7. Miscellaneous
8. System administration commands and daemons

For example, when you do whatis crontab, you'll notice that crontab has two man pages (section 1 and section 5). To view section 5 of crontab man page, do the following.

```
$ whatis crontab
crontab (1)         - maintain crontab files for individual users (V3)
crontab (5)         - tables for driving cron

$ man 5 crontab
```

which command

To find out if a command name is located in your command path:

```
which ue
/usr/local/bin/ue
```

This returns the information that the MicroEMACS editor (ue) is available on this system as the file /usr/local/bin/ue.

To find out if the Korn shell is available on your system:

```
which ksh
/bin/ksh
```

This returns the information that the Korn shell is available on your system in the directory /bin.

To find out which (if any) of several commands are available:

```
which delete talk ps
No delete in /usr/local/utils/bin:/bin:/usr/bin:
/usr/local/ucb/bin:/usr/local/gnu/bin:
/usr/local/public/bin:/usr/bin/X11:.
/usr/bin/talk
/bin/ps
```

This returns the information that the file for the command **delete** does not exist in any of the directories that are specified in this user's PATH environment variable.

Pathnames for the files of the commands **talk** and **ps** are displayed, so these commands are

available on this system.

whereis command

When you want to find out where a specific Unix command exists (for example, where does ls command exists?), you can execute the following command.

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
```

When you want to search an executable from a path other than the whereis default path, you can use -B option and give path as argument to it. This searches for the executable lsmk in the /tmp directory, and displays it, if it is available.

```
$ whereis -u -B /tmp -f lsmk
lsmk: /tmp/lsmk
```

whatis command

Whatis command displays a single line description about a command.

```
$ whatis ls
ls              (1)  - list directory contents

$ whatis ifconfig
ifconfig (8)         - configure a network interface
```

uname command

Uname command displays important information about the system such as — Kernel name, Host name, Kernel release number,
Processor type, etc.,

Sample uname output from a Ubuntu laptop is shown below.

```
$ uname -a
Linux john-laptop 2.6.32-24-generic #41-Ubuntu SMP Thu Aug 19 01:12:52 UTC 2010 i686 GNU/Linux
```

mkdir command

The **mkdir** command creates one or more new directories specified by the *Directory* parameter. Each new directory contains the standard entries **.** (dot) and **..** (dot-dot). You can specify the permissions for the new directories with the **-m** *Mode* flag. You can use the umask subroutine to set the default mode for the **mkdir** command.

The owner-ID and group-ID of the new directories are set to the process's effective user-ID and group-ID, respectively. The setgid bit setting is inherited from the parent directory. To change the setgid bit, you can either specify the **-m** *Mode* flag or issue the **chmod** command after the creation of the directory.

Note: To make a new directory you must have write permission in the parent directory.

Flags

-e Creates directories with encryption inheritance.
-m *Mode* Sets the permission bits for the newly-created directories to the value specified by the *Mode* variable. The *Mode* variable takes the same values as the *Mode* parameter for the chmod command, either in symbolic or numeric form.
When you specify the **-m** flag using symbolic format, the op characters + (plus) and - (minus) are interpreted relative to the assumed permission setting a=rwx. The + adds permissions to the default mode, and the - deletes permissions from the default mode. Refer to the **chmod** command for a complete description of permission bits and formats.

-p Creates missing intermediate path name directories. If the **-p** flag is not specified, the parent directory of each-newly created directory must already exist.

Intermediate directories are created through the automatic invocation of the following **mkdir** commands:

```
mkdir -p -m $(umask -S),u+wx $(dirname Directory) &&
mkdir [-m Mode] Directory
```

where the [-m Mode] represents any option supplied with your original invocation of the **mkdir** command.

The **mkdir** command ignores any *Directory* parameter that names an existing directory. No error is issued.

To create a new directory in the current working directory,

```
mkdir Test
```

The Test directory is created with default permissions.

To create a new directory with permissions

To create a new directory called Test with rwxr-xr-x permissions in the previously created /home/demo/sub1 directory

```
mkdir -m 755 /home/demo/sub1/Test
```

To create a new directory with default permissions in subdirectory

To create a new directory called Test with default permissions in the /home/demo/sub2 directory

```
mkdir -p /home/demo/sub2/Test
```

The **-p** flag creates the /home, /home/demo, and /home/demo/sub2 directories if they do not already exist.

rmdir command

To empty and remove a directory

```
rm mydir/* mydir/.*
rmdir mydir
```

This command removes the contents of the **mydir** file and then removes the empty directory. The **rm** command displays an error message about trying to remove the directories . (dot) and .. (dot, dot), and then the **rmdir** command removes them.

Note that the **rm mydir/* mydir/.*** command first removes files with names that do not begin with a dot, and then removes those with names that do begin with a dot. You may not realize that the directory contains file names that begin with a dot because the **ls** command does not usually list them unless you use the **-a** flag.

To remove the subdirectory

To remove the **/home**, **/home/demo**, and **/home/demo/mydir** directories, type:

```
rmdir -p /home/demo/mydir
```

This command removes first the **/mydir** directory and then the **/demo** and **/home** directories, respectively. If a directory is not empty or does not have write permission when it is to be removed, the command terminates.

touch command

The **touch** command updates the access and modification times of each file specified by the *File* parameter of each directory specified by the *Directory* parameter. If you do not specify a value for the *Time* variable, the **touch** command uses the current time. If you specify a file that does not exist, the **touch** command creates the file unless you specify the -c flag.

The return code from the **touch** command is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

-a Changes the access time of the file specified by the *File* variable. Does not change the modification time unless **-m** is also specified. **-c** Does not create the file if it does not already exist. No diagnostic messages are written concerning this condition. **-f** Attempts to force the touch in spite of read and write permissions on a file. **-m** Changes the modification time of *File*. Does not change the access time unless **-a** is also specified. **-r** *RefFile* Uses the corresponding time of the file specified by the *RefFile* variable instead of the current time. *Time* Specifies the date and time of the new timestamp in the format *MMDDhhmm*[YY], where:

MM
: Specifies the month of the year (01 to 12).

DD
: Specifies the day of the month (01 to 31).

hh
Specifies the hour of the day (00 to 23).
mm
Specifies the minute of the hour (00 to 59).
YY
Specifies the last two digits of the year. If the *YY* variable is not specified, the default value is the current year.

-t *Time* Uses the specified time instead of the current time. The *Time* variable is specified in the decimal form [[*CC*]*YY*]*MMDDhhmm*[**.** *SS*] where:

CC
Specifies the first two digits of the year.
YY
Specifies the last two digits of the year.
MM
Specifies the month of the year (01 to 12).
DD
Specifies the day of the month (01 to 31).
hh
Specifies the hour of the day (00 to 23).
mm
Specifies the minute of the hour (00 to 59).
SS
Specifies the second of the minute (00 to 59).

To update the access and modification times of a file,

touch program.c

This sets the last access and modification times of the program.c file to the current date and time. If the program.c file does not exist, the **touch** command creates an empty file with that name.

To avoid creating a new file

touch -c program.c

- To update only the modification time, enter:

touch -m *.o
This updates the last modification times (not the access times) of the files that end with a .o extension in the current directory. The **touch** command is often used in this way to alter the results of the **make** command.

To explicitly set the access and modification times

touch -c -t 02171425 program.c
This sets the access and modification dates to 14:25 (2:25 p.m.) February 17 of the current year.

To use the time stamp of another file instead of the current time

touch -r file1 program.c
This gives the program.c file the same time stamp as the file1 file.

To touch a file using a specified time other than the current time

touch -t 198503030303.55 program.c
This gives the program.c file a time stamp of 3:03:55 a.m. on March 3, 1985.

rm command

To delete a file

rm myfile

If there is another link to this file, then the file remains under that name, but the name myfile is removed. If myfile is the only link, the file itself is deleted.

To delete a file without first receiving a confirmation prompt

rm -f core
No confirmation prompt is issued before the **rm -f** command attempts to remove the file named core. However, an error message displays if the core file is write-protected and you are not the owner of the file or you do not have root authority. No error message displays when the **rm -f** command attempts to remove nonexistent files.

To delete files one by one

rm -i mydir/*
After each file name is displayed, enter y to delete the file, or press the Enter key to keep it.

- To delete a directory tree, enter:

rm -ir manual

This command recursively removes the contents of all subdirectories of the manual directory, prompting you regarding the removal of each file, and then removes the manual directory itself, for example:

You: **rm -ir manual**
System: rm: Select files in directory manual? Enter y for yes.
You: y
System: rm: Select files in directory manual/draft1? Enter y for yes.
You: y
System: rm: Remove manual/draft1?
You: y
System: rm: Remove manual/draft1/chapter1?
You: y
System: rm: Remove manual/draft1/chapter2?
You: y
System: rm: Select files in directory manual/draft2? Enter y for yes.
You: y
System: rm: Remove manual/draft2?

You: y
System: rm: Remove manual?
You: y

Here, the **rm** command first asks if you want it to search the manual directory. Because the manual directory contains directories, the **rm** command next asks for permission to search manual/draft1 for files to delete, and then asks if you want it to delete the manual/draft1/chapter1 and manual/draft1/chapter2 files. The **rm** command next asks for permission to search the manual/draft2 directory. Then asks for permission to delete the manual/draft1, manual/draft2, and manual directories.

If you deny permission to remove a subdirectory (for example, manual/draft2), the **rm** command does not remove the manual directory. Instead, you see the message: rm: Directory manual not empty.

cp command

Copy file1 to file2 preserving the mode, ownership and timestamp.

$ cp -p file1 file2

Copy file1 to file2. if file2 exists prompt for confirmation before overwritting it.

$ cp -i file1 file2

To make a copy of a file in the current directory,

cp prog.c prog.bak

This copies prog.c to prog.bak. If the prog.bak file does not already exist, the **cp** command creates it. If it does exist, the **cp** command replaces it with a copy of the prog.c file.

To copy a file in your current directory into another directory,

cp jones /home/nick/clients

This copies the jones file to /home/nick/clients/jones.

To copy a file to a new file and preserve the modification date, time, and access control list associated with the source file

cp -p smith smith.jr
This copies the smith file to the smith.jr file. Instead of creating the file with the current date and time stamp, the system gives the smith.jr file the same date and time as the smith file. The smith.jr file also inherits the smith file's access control protection.

To copy all the files in a directory to a new directory,

cp /home/janet/clients/* /home/nick/customers

This copies only the files in the clients directory to the customers directory.

To copy a directory, including all its files and subdirectories, to another directory

cp -R /home/nick/clients /home/nick/customers

Note:A directory cannot be copied into itself.This copies the clients directory, including all its files, subdirectories, and the files in those subdirectories, to the customers/clients directory.

To copy a specific set of files to another directory

cp jones lewis smith /home/nick/clients

This copies the jones, lewis, and smith files in your current working directory to the /home/nick/clients directory.

To use pattern-matching characters to copy files,

cp programs/*.c .

This copies the files in the programs directory that end with .c to the current directory, signified by the single . (dot). You must type a space between the c and the final dot.

To copy a file to a new file and preserve the ACL and EA associated with the source file,

cp -U smith smith.jr

mv command

Rename a file

Rename file1 to file2. if file2 exists prompt for confirmation before overwritting it.

$ mv -i file1 file2

$mv /home/dir/file1 /home/dir/file2

mv -f is just the opposite, which will overwrite file2 without prompting.

This command renames file1 to file2. If a file named file2 already exists, its old contents are replaced with those of appendix.

mv -v will print what is happening during file rename, which is useful while specifying shell metacharacters in the file name argument.

$ mv -v file1 file2

To move a directory

mv book manual

This command moves all files and directories under book to the directory named manual, if manual exists. Otherwise, the directory book is renamed manual.

To move a file to another directory and give it a new name

```
mv intro manual/chap1
```

This command moves intro to manual/chap1. The name intro is removed from the current directory, and the same file appears as chap1 in the directory manual.

To move a file to another directory, keeping the same name

```
mv chap3 manual
```

This command moves chap3 to manual/chap3

Note: Examples 1 and 3 name two files, example 2 names two existing directories,and example 4 names a file and a directory.

To move several files into another directory

```
mv chap4 jim/chap5 /home/manual
```

This command moves the chap4 file to the /home/manual/chap4 file directory and the jim/chap5 file to the /home/manual/chap5 file.

To use the mv command with pattern-matching characters

```
mv manual/* .
```

This command moves all files in the manual directory into the current directory . (period), retaining the names they had in manual. This move also empties manual. You must type a space between the asterisk and the period.

Note: Pattern-matching characters expand names of existing files only. For example, the command mv intro man*/chap1 does not work if the file manual/chap1 does not exist.

cat command

To display a file at the workstation

```
cat notes
```

This command displays the data in the notes file. If the file is more than one less than the number of available display lines, some of the file scrolls off the screen. To list a file one page at a time, use the pg command.

To concatenate several files

```
cat section1.1 section1.2 section1.3 >section1
```

This command creates a file named section1 that is a copy of section1.1 followed by section1.2 and section1.3.

To suppress error messages about files that do not exist

```
cat  -q section2.1 section2.2 section2.3 >section2
```

If section2.1 does not exist, this command concatenates section2.2 and section2.3. The result is the same if you do not use the **-q** flag, except that the **cat** command displays the error message:

```
cat: cannot open section2.1
```

You may want to suppress this message with the **-q** flag when you use the **cat** command in shell procedures.

To append one file to the end of another,

```
cat section1.4 >> section1
```

The >> (two carets) appends a copy of section1.4 to the end of section1. If you want to replace the file, use the > (caret).

To add text to the end of a file

```
cat >>notes
Get milk on the way home
Ctrl-D
```

This command adds Get milk on the way home to the end of the file called notes. The **cat** command does not prompt; it waits for you to enter text. Press the Ctrl-D key sequence to indicate you are finished.

To concatenate several files with text entered from the keyboard

```
cat section3.1 - section3.3 >section3
```

This command concatenates the file section3.1 with text from the keyboard (indicated by the minus sign), and the file section3.3, then directs the output into the file called section3.

You can view multiple files at the same time. Following example prints the content of file1 followed by file2 to stdout.

```
$ cat file1 file2
```

While displaying the file, following cat -n command will prepend the line number to each line

of the output.

```
$ cat -n /etc/logrotate.conf
    1  /var/log/btmp {
    2      missingok
    3      monthly
    4      create 0660 root utmp
    5      rotate 1
    6  }
```

head command

1. Print the first N number of lines

To view the first N number of lines, pass the file name as an argument with -n option as shown below.

```
$ head -n 5 flavours.txt
Ubuntu
Debian
Redhat
Gentoo
Fedora core
```

Note: When you simply pass the file name as an argument to head, it prints out the first 10 lines of the file.

2. Print N number of lines by specifying N with -

You don't even need to pass the -n option as an argument, simply specify the N number of lines followed by '-' as shown below.

```
$ head -4 flavours.txt
Ubuntu
Debian
Redhat
Gentoo
```

3. Print all but not the last N lines

By placing '-' in front of the number with -n option, it prints all the lines of each file but not the last N lines as shown below,

```
$ head -n -5 flavours.txt
Ubuntu
```

4. Print the N number of bytes

You can use the -c option to print the N number of bytes from the initial part of file.

```
$ head -c 5 flavours.txt
Ubuntu
```

Note : As like -n option, here also you can pass '-' in front of number to print all bytes but not the last N bytes.

5. Passing Output of Other command to Head Input

You may pass the output of other commands to the head command via pipe as shown below,

```
$ ls | head
bin
boot
cdrom
dev
etc
home
initrd.img
lib
lost+found
media
```

Display the first 5 lines of several files

head -5 *.xdh

This displays, one after the other, the first 5 lines of each file with the extension .xdh in the current directory.

The first line of each file is shown as: ==> *filename* <==

tail command

Tail prints the last N number of lines from given input. By default, it prints last 10 lines of each given file.

1. Print the last N lines

To view the last N number of lines from file, just pass the file name with -n option as shown below.

```
$ tail -n 5 flavours.txt
Debian
Redhat
Gentoo
Fedora core
```

Note: When you simply pass the filename, it prints out the last 10 lines of the file.

2. Print the appended lines as and when the file grows

You can use -f option to output the appended lines of file instantly as shown below,

```
$ tail -f /var/log/messages
```

Note: This is very useful to monitor the log files.

3. Terminate the tail command once PID dies

Using –pid with -f option, you can terminate the tail command when the specific process gets over or killed as shown below.

```
$ tail -f /tmp/debug.log --pid=2575
```

In the above tail gets terminated immediately when the pid 2575 vanishes.

4. Keep on trying to tail the file even if it is non-existent

Sometimes, the file intended to tail may not be available when you run the tail command and it may get created later or the files becomes inaccessible . By this time, you can use the –retry option to keep on trying to open the file as shown below.

```
$ tail -f /tmp/debug.log --retry
tail: warning: --retry is useful mainly when following by name
tail: cannot open `/tmp/log' for reading: No such file or directory
```

After giving the above warnings, it is trying to open the file.

less command

Viewing a Text File: more or less

Often, you will want to look at a text file, without bothering to load it into an editor--you just want to read it, perhaps before deleting it, to make sure it was what you thought it was. UNIX provides several ways of doing this. You may see people use the command "cat" but we're not going to recommend that here because of its relative uselessness for this purpose. Instead, we're going to give you two other commands, more or less...

No, really: those are the commands; more and less. Pick either one; they're very similar. We'll discuss "less" here.

The format of the command is:

less **filename**

This will cause the file to be displayed to your screen, one screen-full at a time. There is immediate help for more, accessible via the h subcommand (when you're "in" less). Pressing the <**Spacebar**> moves you forward one screen through the file. You can use the subcommand b to move backward one screen. When you're ready to quit, q (for "quit") exits the file and returns you to the UNIX $ prompt.

There are other commands to let you do things like move forward or backward a half-screen, or a particular number of lines or screens. Type h (in less or more) to see a complete list.

More Uses for less or more

The less and more commands are also useful for paging through long output of other types; like, for example, if your ls -la listing runs off the screen, and you want to be able to view it one-screen-full at a time.

In general,

command | less

...will invoke less to handle output so you can page forward and backward through it. For example, for a long directory listing,

ls -la | less

would let you view the listing using less to control paging forward and backwards through the listing.

The | symbol (called the "vertical bar," or "pipe") is found as <**Shift**><**backslash**> on the right side of most keyboards, above the <**Enter**> key. Basically, it is a UNIX mechanism which allows you to "string together" multiple commands, such that the output from each command is fed to the input of the next command, and so on, allowing you to create more complex "combination commands." In the case of ls-la | less, we are "piping" the output of ls -la through less.

tar command

a. Creating an archive using tar command

Creating an uncompressed tar archive using option cvf

This is the basic command to create a tar archive.

```
$ tar cvf archive_name.tar dirname/
```

In the above command:

c – create a new archive
v – verbosely list files which are processed.
f – following is the archive file name

Creating a tar gzipped archive using option cvzf

The above tar cvf option, does not provide any compression. To use a gzip compression on the tar archive, use the z option as shown below.

```
$ tar cvzf archive_name.tar.gz  dirname/
```

z – filter the archive through gzip

Note: .tgz is same as .tar.gz

Note: I like to keep the 'cvf' (or tvf, or xvf) option unchanged for all archive creation (or view, or extract) and add additional option at the end, which is easier to remember. i.e cvf for archive creation, cvfz for compressed gzip archive creation, cvfj for compressed bzip2 archive creation etc., For this method to work properly, don't give – in front of the options.

Creating a bzipped tar archive using option cvjf

Create a bzip2 tar archive as shown below:

```
$ tar cvfj archive_name.tar.bz2 dirname/
```

j – filter the archive through bzip2

gzip vs bzip2: bzip2 takes more time to compress and decompress than gzip. bzip2 archival size is less than gzip.

Note: .tbz and .tb2 is same as .tar.bz2

b. Extracting (untar) an archive using tar command

Extract a *.tar file using option xvf

Extract a tar file using option x as shown below:

```
$ tar xvf archive_name.tar
```

x – extract files from archive

Extract a gzipped tar archive (*.tar.gz) using option xvzf

Use the option z for uncompressing a gzip tar archive.

```
$ tar xvfz archive_name.tar.gz
```

Extracting a bzipped tar archive (*.tar.bz2) using option xvjf

Use the option j for uncompressing a bzip2 tar archive.

```
$ tar xvfj archive_name.tar.bz2
```

Note: In all the above commands v is optional, which lists the file being processed.

c. Listing an archive using tar command

View the tar archive file content without extracting using option tvf

You can view the *.tar file content before extracting as shown below.

```
$ tar tvf archive_name.tar
```

View the *.tar.gz file content without extracting using option tvzf

You can view the *.tar.gz file content before extracting as shown below.

```
$ tar tvfz archive_name.tar.gz
```

View the *.tar.bz2 file content without extracting using option tvjf

You can view the *.tar.bz2 file content before extracting as shown below.

```
$ tar tvfj archive_name.tar.bz2
```

d. Listing out the tar file content with less command

When the number of files in an archive is more, you may pipe the output of tar to less. But, you can also use less command directly to view the tar archive output.

e. Extract a single file from tar, tar.gz, tar.bz2 file

To extract a specific file from a tar archive, specify the file name at the end of the tar xvf command as shown below. The following command extracts only a specific file from a large tar file.

```
$ tar xvf archive_file.tar /path/to/file
```

Use the relevant option z or j according to the compression method gzip or bzip2 respectively as shown below.

```
$ tar xvfz archive_file.tar.gz /path/to/file
```

```
$ tar xvfj archive_file.tar.bz2 /path/to/file
```

f. Extract a single directory from tar, tar.gz, tar.bz2 file

To extract a single directory (along with it's subdirectory and files) from a tar archive, specify the directory name at the end of the tar xvf command as shown below. The following extracts only a specific directory from a large tar file.

```
$ tar xvf archive_file.tar /path/to/dir/
```

To extract multiple directories from a tar archive, specify those individual directory names at the end of the tar xvf command as shown below.

```
$ tar xvf archive_file.tar /path/to/dir1/ /path/to/dir2/
```

Use the relevant option z or j according to the compression method gzip or bzip2 respectively as shown below.

```
$ tar xvfz archive_file.tar.gz /path/to/dir/
```

```
$ tar xvfj archive_file.tar.bz2 /path/to/dir/
```

g. Extract group of files from tar, tar.gz, tar.bz2 archives using regular expression

You can specify a regex, to extract files matching a specified pattern. For example, following tar command extracts all the files with pl extension.

```
$ tar xvf archive_file.tar --wildcards '*.pl'
```

Options explanation:

–wildcards *.pl – files with pl extension

h. Adding a file or directory to an existing archive using option -r

You can add additional files to an existing tar archive as shown below. For example, to append a file to *.tar file do the following:

```
$ tar rvf archive_name.tar newfile
```

This newfile will be added to the existing archive_name.tar. Adding a directory to the tar is also similar,

```
$ tar rvf archive_name.tar newdir/
```

Note: You cannot add file or directory to a compressed archive. If you try to do so, you will get “tar: Cannot update compressed archives” error as shown below.

```
$ tar rvfz archive_name.tgz newfile
tar: Cannot update compressed archives
Try `tar --help' or `tar --usage' for more information.
```

i. Verify files available in tar using option -W

As part of creating a tar file, you can verify the archive file that got created using the option W as shown below.

```
$ tar cvfW file_name.tar dir/
```

If you are planning to remove a directory/file from an archive file or from the file system, you might want to verify the archive file before doing it as shown below.

```
$ tar tvfW file_name.tar
Verify 1/file1
1/file1: Mod time differs
1/file1: Size differs
Verify 1/file2
Verify 1/file3
```

If an output line starts with Verify, and there is no differs line then the file/directory is Ok. If not, you should investigate the issue.

Note: for a compressed archive file (*.tar.gz, *.tar.bz2) you cannot do the verification.

Finding the difference between an archive and file system can be done even for a compressed archive. It also shows the same output as above excluding the lines with Verify.

Finding the difference between gzip archive file and file system

```
$ tar dfz file_name.tgz
```

Finding the difference between bzip2 archive file and file system

```
$ tar dfj file_name.tar.bz2
```

j. Estimate the tar archive size

The following command, estimates the tar file size (in KB) before you create the tar file.

```
$ tar -cf - /directory/to/archive/ | wc -c
20480
```

The following command, estimates the compressed tar file size (in KB) before you create the tar.gz, tar.bz2 files.

```
$ tar -czf - /directory/to/archive/ | wc -c
508

$ tar -cjf - /directory/to/archive/ | wc -c
428
```

locate command

Using locate command you can quickly search for the location of a specific file (or group of files). Locate command uses the database created by updatedb.

The example below shows all files in the system that contains the word crontab in it.

```
$ locate crontab
/etc/anacrontab
/etc/crontab
/usr/bin/crontab
/usr/share/doc/cron/examples/crontab2english.pl.gz
/usr/share/man/man1/crontab.1.gz
/usr/share/man/man5/anacrontab.5.gz
/usr/share/man/man5/crontab.5.gz
/usr/share/vim/vim72/syntax/crontab.vim
```

grep command examples

a. Search for the given string in a single file

The basic usage of grep command is to search for a specific string in the specified file as shown below.

```
Syntax:  grep "literal_string" filename
$ grep "this" demo_file
this line is the 1st lower case line in this file.
Two lines above this line is empty.
```

b. Checking for the given string in multiple files.

```
Syntax: grep "string" FILE_PATTERN
```

This is also a basic usage of grep command. For this example, let us copy the demo_file to demo_file1. The grep output will also include the file name in front of the line that matched the specific pattern as shown below. When the Linux shell sees the meta character, it does the expansion and gives all the files as input to grep.

```
$ cp demo_file demo_file1

$ grep "this" demo_*
demo_file:this line is the 1st lower case line in this file.
demo_file:Two lines above this line is empty.
demo_file:And this is the last line.
demo_file1:this line is the 1st lower case line in this file.
demo_file1:Two lines above this line is empty.
demo_file1:And this is the last line.
```

c. Case insensitive search using grep -i

Syntax: grep -i "string" FILE

This is also a basic usage of the grep. This searches for the given string/pattern case insensitively. So it matches all the words such as “the”, “THE” and “The” case insensitively as shown below.

```
 $ grep -i "the" demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.
And this is the last line.
```

d. Match regular expression in files

Syntax: grep "REGEX" filename

This is a very powerful feature, if you can use use regular expression effectively. In the following example, it searches for all the pattern that starts with “lines” and ends with “empty” with anything in-between. i.e To search “lines[anything in-between]empty” in the demo_file.

```
$ grep "lines.*empty" demo_file
Two lines above this line is empty.
```

From documentation of grep: A regular expression may be followed by one of several repetition operators:

? The preceding item is optional and matched at most once.
* The preceding item will be matched zero or more times.
+ The preceding item will be matched one or more times.
{n} The preceding item is matched exactly n times.
{n,} The preceding item is matched n or more times.
{,m} The preceding item is matched at most m times.
{n,m} The preceding item is matched at least n times, but not more than m times.

e. Checking for full words, not for sub-strings using grep -w

If you want to search for a word, and to avoid it to match the substrings use -w option. Just doing out a normal search will show out all the lines.

The following example is the regular grep where it is searching for “is”. When you search for “is”, without any option it will show out “is”, “his”, “this” and everything which has the substring “is”.

```
$ grep -i "is" demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.
Two lines above this line is empty.
And this is the last line.
```

The following example is the WORD grep where it is searching only for the word “is”. Please note that this output does not contain the line “This Line Has All Its First Character Of The Word With Upper Case”, even though “is” is there in the “This”, as the following is looking only for the word “is” and not for “this”.

```
$ grep -iw "is" demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
Two lines above this line is empty.
And this is the last line.
```

f. Displaying lines before/after/around the match using grep -A, -B and -C

When doing a grep on a huge file, it may be useful to see some lines after the match. You might feel handy if grep can show you not only the matching lines but also the lines after/before/around the match.

Please create the following demo_text file for this example.

```
$ cat demo_text
4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

 * e - go to the end of the current word.
 * E - go to the end of the current WORD.
 * b - go to the previous (before) word.
 * B - go to the previous (before) WORD.
 * w - go to the next word.
 * W - go to the next WORD.

WORD - WORD consists of a sequence of non-blank characters, separated with white space.
word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word
```

* 192.168.1.1 - single WORD
* 192.168.1.1 - seven words.

f.1 Display N lines after match

-A is the option which prints the specified N lines after the match as shown below.

Syntax:
grep -A <N> "string" FILENAME

The following example prints the matched line, along with the 3 lines after it.

$ grep -A 3 -i "example" demo_text
Example to show the difference between WORD and word

* 192.168.1.1 - single WORD
* 192.168.1.1 - seven words.

f.2 Display N lines before match

-B is the option which prints the specified N lines before the match.

Syntax:
grep -B <N> "string" FILENAME

When you had option to show the N lines after match, you have the -B option for the opposite.

$ grep -B 2 "single WORD" demo_text
Example to show the difference between WORD and word

* 192.168.1.1 - single WORD

f.3 Display N lines around match

-C is the option which prints the specified N lines before the match. In some occasion you might want the match to be appeared with the lines from both the side. This options shows N lines in both the side(before & after) of match.

$ grep -C 2 "Example" demo_text
word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

* 192.168.1.1 - single WORD

g. Highlighting the search using GREP_OPTIONS

As grep prints out lines from the file by the pattern / string you had given, if you wanted it to highlight which part matches the line, then you need to follow the following way.

When you do the following export you will get the highlighting of the matched searches. In the following example, it will highlight all the this when you set the GREP_OPTIONS environment variable as shown below.

```
$ export GREP_OPTIONS='--color=auto' GREP_COLOR='100;8'

$ grep this demo_file
this line is the 1st lower case line in this file.
Two lines above this line is empty.
And this is the last line.
```

h. Searching in all files recursively using grep -r

When you want to search in all the files under the current directory and its sub directory. -r option is the one which you need to use. The following example will look for the string “ramesh” in all the files in the current directory and all it’s subdirectory.

```
$ grep -r "ramesh" *
```

i. Invert match using grep -v

You had different options to show the lines matched, to show the lines before match, and to show the lines after match, and to highlight match. So definitely You’d also want the option -v to do invert match.

When you want to display the lines which does not matches the given string/pattern, use the option -v as shown below. This example will display all the lines that did not match the word “go”.

```
$ grep -v "go" demo_text
4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

WORD - WORD consists of a sequence of non-blank characters, separated with white space.
word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

* 192.168.1.1 - single WORD
* 192.168.1.1 - seven words.
```

j. display the lines which does not matches all the given pattern.

```
Syntax:
grep -v -e "pattern" -e "pattern"

$ cat test-file.txt
a
b
c
d
```

```
$ grep -v -e "a" -e "b" -e "c" test-file.txt
d
```

k. Counting the number of matches using grep -c

When you want to count that how many lines matches the given pattern/string, then use the option -c.

Syntax:

```
grep -c "pattern" filename
```

```
$ grep -c "go" demo_text
6
```

When you want do find out how many lines matches the pattern

```
$ grep -c this demo_file
3
```

When you want do find out how many lines that does not match the pattern

```
$ grep -v -c this demo_file
4
```

l. Display only the file names which matches the given pattern using grep -l

If you want the grep to show out only the file names which matched the given pattern, use the -l (lower-case L) option.

When you give multiple files to the grep as input, it displays the names of file which contains the text that matches the pattern, will be very handy when you try to find some notes in your whole directory structure.

```
$ grep -l this demo_*
demo_file
demo_file1
```

m. Show only the matched string

By default grep will show the line which matches the given pattern/string, but if you want the grep to show out only the matched string of the pattern then use the -o option.

It might not be that much useful when you give the string straight forward. But it becomes very useful when you give a regex pattern and trying to see what it matches as

```
$ grep -o "is.*line" demo_file
is line is the 1st lower case line
is line
is is the last line
```

n. Show the position of match in the line

When you want grep to show the position where it matches the pattern in the file, use the following options as

Syntax:
grep -o -b "pattern" file

$ cat temp-file.txt
12345
12345

$ grep -o -b "3" temp-file.txt
2:3
8:3

Note: The output of the grep command above is not the position in the line, it is byte offset of the whole file.

o. Show line number while displaying the output using grep -n

To show the line number of file with the line matched. It does 1-based line numbering for each file. Use -n option to utilize this feature.

$ grep -n "go" demo_text
5: * e - go to the end of the current word.
6: * E - go to the end of the current WORD.
7: * b - go to the previous (before) word.
8: * B - go to the previous (before) WORD.
9: * w - go to the next word.
10: * W - go to the next WORD.

find command examples

1. Find Files Using Name

This is a basic usage of the find command. This example finds all files with name — MyCProgram.c in the current directory and all it's sub-directories.

find -name "MyCProgram.c"
./backup/MyCProgram.c
./MyCProgram.c

2. Find Files Using Name and Ignoring Case

This is a basic usage of the find command. This example finds all files with name — MyCProgram.c (ignoring the case) in the current directory and all it's sub-directories.

find -iname "MyCProgram.c"
./mycprogram.c
./backup/mycprogram.c

```
./backup/MyCProgram.c
./MyCProgram.c
```

3. Limit Search To Specific Directory Level Using mindepth and maxdepth

Find the passwd file under all sub-directories starting from root directory.

```
# find / -name passwd
./usr/share/doc/nss_ldap-253/pam.d/passwd
./usr/bin/passwd
./etc/pam.d/passwd
./etc/passwd
```

Find the passwd file under root and one level down. (i.e root — level 1, and one sub-directory — level 2)

```
# find -maxdepth 2 -name passwd
./etc/passwd
```

Find the passwd file under root and two levels down. (i.e root — level 1, and two sub-directories — level 2 and 3)

```
# find / -maxdepth 3 -name passwd
./usr/bin/passwd
./etc/pam.d/passwd
./etc/passwd
```

Find the password file between sub-directory level 2 and 4.

```
 # find -mindepth 3 -maxdepth 5 -name passwd
./usr/bin/passwd
./etc/pam.d/passwd
```

4. Executing Commands on the Files Found by the Find Command.

In the example below, the find command calculates the md5sum of all the files with the name MyCProgram.c (ignoring case). {} is replaced by the current file name.

```
# find -iname "MyCProgram.c" -exec md5sum {} \;
d41d8cd98f00b204e9800998ecf8427e  ./mycprogram.c
d41d8cd98f00b204e9800998ecf8427e  ./backup/mycprogram.c
d41d8cd98f00b204e9800998ecf8427e  ./backup/MyCProgram.c
d41d8cd98f00b204e9800998ecf8427e  ./MyCProgram.c
```

5. Inverting the match.

Shows the files or directories whose name are not MyCProgram.c .Since the maxdepth is 1, this will look only under current directory.

```
# find -maxdepth 1 -not -iname "MyCProgram.c"
.
./MybashProgram.sh
./create_sample_files.sh
```

./backup
./Program.c

6. Finding Files by its inode Number.

Every file has an unique inode number, using that we can identify that file. Create two files with similar name. i.e one file with a space at the end.

```
# touch "test-file-name"

# touch "test-file-name "
[Note: There is a space at the end]

# ls -1 test*
test-file-name
test-file-name
```

From the ls output, you cannot identify which file has the space at the end. Using option -i, you can view the inode number of the file, which will be different for these two files.

```
# ls -i1 test*
16187429 test-file-name
16187430 test-file-name
```

You can specify inode number on a find command as shown below. In this example, find command renames a file using the inode number.

```
# find -inum 16187430 -exec mv {} new-test-file-name \;

# ls -i1 *test*
16187430 new-test-file-name
16187429 test-file-name
```

You can use this technique when you want to do some operation with the files which are named poorly as shown in the example below. For example, the file with name — file?.txt has a special character in it. If you try to execute “rm file?.txt”, all the following three files will get removed. So, follow the steps below to delete only the “file?.txt” file.

```
# ls
file1.txt  file2.txt  file?.txt
```

Find the inode numbers of each file.

```
# ls -i1
804178 file1.txt
804179 file2.txt
804180 file?.txt
```

Use the inode number to remove the file that had special character in it as shown below.

```
# find -inum 804180 -exec rm {} \;

# ls
file1.txt  file2.txt
[Note: The file with name "file?.txt" is now removed]
```

7. Find file based on the File-Permissions

Following operations are possible.

Find files that match exact permission
Check whether the given permission matches, irrespective of other permission bits
Search by giving octal / symbolic representation

For this example, let us assume that the directory contains the following files. Please note that the file-permissions on these files are different.

```
# ls -l
total 0
-rwxrwxrwx 1 root root 0 2009-02-19 20:31 all_for_all
-rw-r--r-- 1 root root 0 2009-02-19 20:30 everybody_read
---------- 1 root root 0 2009-02-19 20:31 no_for_all
-rw------- 1 root root 0 2009-02-19 20:29 ordinary_file
-rw-r----- 1 root root 0 2009-02-19 20:27 others_can_also_read
----r----- 1 root root 0 2009-02-19 20:27 others_can_only_read
```

Find files which has read permission to group. Use the following command to find all files that are readable by the world in your home directory, irrespective of other permissions for that file.

```
# find . -perm -g=r -type f -exec ls -l {} \;
-rw-r--r-- 1 root root 0 2009-02-19 20:30 ./everybody_read
-rwxrwxrwx 1 root root 0 2009-02-19 20:31 ./all_for_all
----r----- 1 root root 0 2009-02-19 20:27 ./others_can_only_read
-rw-r----- 1 root root 0 2009-02-19 20:27 ./others_can_also_read
```

Find files which has read permission only to group.

```
# find . -perm g=r -type f -exec ls -l {} \;
----r----- 1 root root 0 2009-02-19 20:27 ./others_can_only_read
```

Find files which has read permission only to group [search by octal]

```
# find . -perm 040 -type f -exec ls -l {} \;
----r----- 1 root root 0 2009-02-19 20:27 ./others_can_only_read
```

8. Find all empty files (zero byte file) in your home directory and it's subdirectory

Most files of the following command output will be lock-files and place holders created by other applications.

```
# find ~ -empty
```

List all the empty files only in your home directory.

```
# find . -maxdepth 1 -empty
```

List only the non-hidden empty files only in the current directory.

```
# find . -maxdepth 1 -empty -not -name ".*"
```

9. Finding the Top 5 Big Files

The following command will display the top 5 largest file in the current directory and it's subdirectory. This may take a while to execute depending on the total number of files the command has to process.

```
# find . -type f -exec ls -s {} \; | sort -n -r | head -5
```

10. Finding the Top 5 Small Files

Technique is same as finding the bigger files, but the only difference the sort is ascending order.

```
# find . -type f -exec ls -s {} \; | sort -n  | head -5
```

In the above command, most probably you will get to see only the ZERO byte files (empty files). So, you can use the following command to list the smaller files other than the ZERO byte files.

```
# find . -not -empty -type f -exec ls -s {} \; | sort -n  | head -5
```

11. Find Files Based on file-type using option -type

Find only the socket files.

```
# find . -type s
```

Find all directories

```
# find . -type d
```

Find only the normal files

```
# find . -type f
```

Find all the hidden files

```
# find . -type f -name ".*"
```

Find all the hidden directories

```
# find -type d -name ".*"
```

12. Find files by comparing with the modification time of other file.

Show files which are modified after the specified file. The following find command displays all the files that are created/modified after ordinary_file.

```
# ls -lrt
total 0
-rw-r----- 1 root root 0 2009-02-19 20:27 others_can_also_read
----r----- 1 root root 0 2009-02-19 20:27 others_can_only_read
-rw------- 1 root root 0 2009-02-19 20:29 ordinary_file
-rw-r--r-- 1 root root 0 2009-02-19 20:30 everybody_read
-rwxrwxrwx 1 root root 0 2009-02-19 20:31 all_for_all
---------- 1 root root 0 2009-02-19 20:31 no_for_all

# find -newer ordinary_file
.
./everybody_read
./all_for_all
./no_for_all
```

13. Find Files by Size

Using the -size option you can find files by size.

Find files bigger than the given size

```
# find ~ -size +100M
```

Find files smaller than the given size

```
# find ~ -size -100M
```

Find files that matches the exact given size

```
# find ~ -size 100M
```

Note: – means less than the give size, + means more than the given size, and no symbol means

exact given size.

14. Create Alias for Frequent Find Operations

If you find some thing as pretty useful, then you can make it as an alias. And execute it whenever you want.

Remove the files named a.out frequently.

```
# alias rmao="find . -iname a.out -exec rm {} \;"
# rmao
```

Remove the core files generated by c program.

```
# alias rmc="find . -iname core -exec rm {} \;"
# rmc
```

15. Remove big archive files using find command

The following command removes *.zip files that are over 100M.

```
# find / -type f -name *.zip -size +100M -exec rm -i {} \;"
```

Remove all *.tar file that are over 100M using the alias rm100m (Remove 100M). Use the similar concepts and create alias like rm1g, rm2g, rm5g to remove file size greater than 1G, 2G and 5G respectively.

```
# alias rm100m="find / -type f -name *.tar -size +100M -exec rm -i {} \;"
# alias rm1g="find / -type f -name *.tar -size +1G -exec rm -i {} \;"
# alias rm2g="find / -type f -name *.tar -size +2G -exec rm -i {} \;"
# alias rm5g="find / -type f -name *.tar -size +5G -exec rm -i {} \;"

# rm100m
# rm1g
# rm2g
# rm5g
```

Find Files Based on Access / Modification / Change Time

You can find files based on following three file time attribute.

1. **Access time** of the file. Access time gets updated when the **file accessed**.
2. **Modification time** of the file. Modification time gets updated when the **file content modified**.
3. **Change time** of the file. Change time gets updated when the **inode data changes**.

In the following examples, the difference between the **min option** and the **time option** is the argument.

min argument treats its argument as **minutes**. For example, min 60 = 60 minutes (1 hour).

time argument treats its argument as **24 hours**. For example, time 2 = 2*24 hours (2 days). While doing the 24 hours calculation, the fractional parts are ignored so 25 hours is taken as 24 hours, and 47 hours is also taken as 24 hours, only 48 hours is taken as 48 hours. To get more clarity refer the -atime section of the **find command** man page.

-

Example 1: Find files whose content got updated within last 1 hour

To find the files based up on the content modification time, the option -mmin, and -mtime is used. Following is the definition of mmin and mtime from man page.

-mmin n File's data was last modified **n minutes** ago.
-mtime n File's data was last modified **n*24 hours** ago.

Following example will find files in the current directory and sub-directories, whose content got updated within last 1 hour (60 minutes)

```
# find . -mmin -60
```

In the same way, following example finds all the files (under root file system /) that got updated within the last 24 hours (1 day).

```
# find / -mtime -1
```

Example 2: Find files which got accessed before 1 hour

To find the files based up on the file access time, the option -amin, and -atime is used. Following is the definition of amin and atime from find man page.

-amin n File was last accessed **n minutes** ago
-atime n File was last accessed **n*24 hours** ago

Following example will find files in the current directory and sub-directories, which got accessed within last 1 hour (60 minutes)

```
# find -amin -60
```

In the same way, following example finds all the files (under root file system /) that got accessed within the last 24 hours (1 day).

```
# find / -atime -1
```

Example 3: Find files which got changed exactly before 1 hour

To find the files based up on the file inode change time, the option -cmin, and -ctime is used. Following is the definition of cmin and ctime from find man page.

-cmin n File's status was last changed **n minutes** ago.
-ctime n File's status was last changed **n*24 hours** ago.

Following example will find files in the current directory and sub-directories, which changed within last 1 hour (60 minutes)

```
# find . -cmin -60
```

In the same way, following example finds all the files (under root file system /) that got changed within the last 24 hours (1 day).

```
# find / -ctime -1
```

Example 4: Restricting the find output only to files.

(Display only files as find command results)

The above find command's will also show the directories because directories gets accessed when the file inside it gets accessed. But if you want only the files to be displayed then give -type f in the find command as

The following **find command** displays files that are accessed in the last 30 minutes.

```
# find /etc/sysconfig -amin -30
.
./console
./network-scripts
./i18n
./rhn
./rhn/clientCaps.d
./networking
./networking/profiles
./networking/profiles/default
./networking/profiles/default/resolv.conf
./networking/profiles/default/hosts
./networking/devices
./apm-scripts
```

[Note: The above output contains both files and directories]

```
# find /etc/sysconfig -amin -30 -type f
./i18n
./networking/profiles/default/resolv.conf
./networking/profiles/default/hosts
```

[Note: The above output contains only files]

Example 5: Restricting the search only to unhidden files.

(Do not display hidden files in find output)

When we don't want the hidden files to be listed in the find output, we can use the following regex.
The below find displays the files which are modified in the last 15 minutes. And it lists only the unhidden files. i.e hidden files that starts with a . (period) are not displayed in the find output.

```
# find . -mmin -15 \( ! -regex ".*/\..*" \)
```

Finding Files Comparatively Using Find Command

Human mind can remember things better by reference such as, i want to find files which i edited after editing the file “test”. You can find files by referring to the other files modification as like the following.

Example 6: Find files which are modified after modification of a particular FILE

Syntax: find -newer FILE

Following example displays all the files which are modified after the /etc/passwd files was modified. This is helpful, if you want to track all the activities you’ve done after adding a new user.

```
# find -newer /etc/passwd
```

Example 7: Find files which are accessed after modification of a specific FILE

Syntax: find -anewer FILE

Following example displays all the files which are accessed after modifying /etc/hosts. If you remember adding an entry to the /etc/hosts and would like to see all the files that you’ve accessed since then, use the following command.

```
# find -anewer /etc/hosts
```

Example 8: Find files whose status got changed after the modification of a specific FILE.

Syntax: find -cnewer FILE

Following example displays all the files whose status got changed after modifying the /etc/fstab. If you remember adding a mount point in the /etc/fstab and would like to know all the files who status got changed since then, use the following command.

```
find -cnewer /etc/fstab
```

Perform Any Operation on Files Found From Find Command

We have looked at many different ways of finding files using **find command** in this article and also in our previous article. If you are not familiar in finding files in different ways, i strongly recommend you to read the part 1.

This section explains about how to do different operation on the files from the find command. i.e how to manipulate the files returned by the find command output.

We can specify any operation on the files found from find command.

```
find <CONDITION to Find files> -exec <OPERATION> \;
```

The OPERATION can be anything such as:

rm command to remove the files found by find command.
mv command to rename the files found.
ls -l command to get details of the find command output files.
md5sum on find command output files
wc command to count the total number of words on find command output files.
Execute any **Unix shell command** on find command output files.
or Execute your own **custom shell script** / command on find command output files.

Example 9: ls -l in find command output.

Long list the files which are edited within the last 1 hour.

```
# find -mmin -60
./cron
./secure

# find -mmin -60 -exec ls -l {} \;
-rw-------  1 root root 1028 Jun 21 15:01 ./cron
-rw-------  1 root root 831752 Jun 21 15:42 ./secure
```

Example 10: Searching Only in the Current Filesystem

System administrators would want to search in the root file system, but not in the other mounted partitions. When you have multiple partitions mounted, and if you want to search in /. You can do the following.

Following command will search for *.log files starting from /. i.e If you have multiple partitions mounted under / (root), the following command will search all those mounted partitions.

```
# find / -name "*.log"
```

This will search for the file only in the current file system. Following is the xdev definition from find man page:

-xdev Don't descend directories on other filesystems.

Following command will search for *.log files starting from / (root) and only in the current file system. i.e If you have multiple partitions mounted under / (root), the following command will NOT search all those mounted partitions.

```
# find / -xdev -name "*.log"
```

Example 11: Using more than one { } in same command

Manual says only one instance of the {} is possible. But you can use more than one {} in the same command as shown below.

```
# find -name "*.txt" cp {} {}.bkup \;
```

Using this {} in the same command is possible but using it in different command it is not possible, say you want to rename the files as following, which will not give the expected result.

```
find -name "*.txt" -exec mv {} `basename {} .htm`.html \;
```

Example 12: Using { } in more than one instance.

You can simulate it by writing a shell script as shown below.

```
# mv "$1" "`basename "$1" .htm`.html"
```

These double quotes are to handle spaces in file name. And then call that shell script from the **find command** as shown below.

```
find -name "*.html" -exec ./mv.sh '{}' \;
```

So for any reason if you want the same file name to be used more than once then writing the simple shell script and passing the file names as argument is the simplest way to do it.

Example 13: Redirecting errors to /dev/null

Redirecting the errors is not a good practice. An experienced user understands the importance of getting the error printed on terminal and fix it.
Particularly in find command redirecting the errors is not a good practice. But if you don't want to see the errors and would like to redirect it to null do it as shown below.

```
find -name "*.txt" 2>>/dev/null
```

Sometimes this may be helpful. For example, if you are trying to find all the *.conf file under / (root) from your account, you may get lot of "Permission denied" error message as shown below.

```
$ find / -name "*.conf"
/sbin/generate-modprobe.conf
find: /tmp/orbit-root: Permission denied
find: /tmp/ssh-gccBMp5019: Permission denied
find: /tmp/keyring-5iqiGo: Permission denied
find: /var/log/httpd: Permission denied
find: /var/log/ppp: Permission denied
/boot/grub/grub.conf
find: /var/log/audit: Permission denied
find: /var/log/squid: Permission denied
find: /var/log/samba: Permission denied
find: /var/cache/alchemist/printconf.rpm/wm: Permission denied
[Note: There are two valid *.conf files burned in the "Permission denied" messages]
```

So, if you want to just view the real output of the **find command** and not the "Permission denied" error message you can redirect the error message to /dev/null as shown below.

```
$ find / -name "*.conf" 2>>/dev/null
/sbin/generate-modprobe.conf
/boot/grub/grub.conf
[Note: All the "Permission denied" messages are not displayed]
```

Example 14: Substitute space with underscore in the file name.

Audio files you download from internet mostly come with the spaces in it. But having space in the file name is not so good for Linux kind of systems. You can use the find and rename command combination as shown below to rename the files, by substituting the space with underscore.

The following replaces space in all the *.mp3 files with _

```
$ find . -type f -iname "*.mp3" -exec rename "s/ /_/g" {} \;
```

Example 15: Executing two find commands at the same time

As shown in the examples of the find command in its manual page, the following is the syntax which can be used to execute two commands in single traversal.

The following find command example, traverse the filesystem just once, listing setuid files and directories into /root/suid.txt and large files into /root/big.txt.

```
# find /    \( -perm -4000 -fprintf /root/suid.txt '%#m %u %p\n' \) , \
 \( -size +100M -fprintf /root/big.txt '%-10s %p\n' \)
```

ssh command

Login to remote host

```
ssh -l jsmith remotehost.example.com
```

Debug ssh client

```
ssh -v -l jsmith remotehost.example.com
```

Display ssh client version

```
$ ssh -V
OpenSSH_3.9p1, OpenSSL 0.9.7a Feb 19 2003
```

sed command

Sed regular expressions

The sed regular expressions are essentially the same as the grep regular expressions. They are summarized below.

^	matches the beginning of the line
$	matches the end of the line
.	Matches any single character
(character)*	match arbitrarily many occurences of (character)
(character)?	Match 0 or 1 instance of (character)

[abcdef]	Match any character enclosed in [] (in this instance, a b c d e or f) ranges of characters such as [a-z] are permitted. The behaviour of this deserves more description. See the page on grep for more details about the syntax of lists.
[^abcdef]	Match any character *NOT* enclosed in [] (in this instance, any character other than a b c d e or f)
(character)\{m,n\}	Match m-n repetitions of (character)
(character)\{m,\}	Match m or more repetitions of (character)
(character)\{,n\}	Match n or less (possibly 0) repetitions of (character)
(character)\{n\}	Match exactly n repetitions of (character)
\(expression\)	Group operator.
\n	Backreference - matches nth group
expression1| expression2	Matches expression1 or expression 2. Works with GNU sed, but this feature might not work with other forms of sed.

Special Characters

The special character in *sed* are the same as those in *grep*, with one key difference: the forward slash / is a special character in *sed*. The reason for this will become very clear when studying *sed* commands.

How it Works: A Brief Introduction

Sed works as follows: it reads from the standard input, one line at a time. for each line, it executes a series of editing commands, then the line is written to STDOUT. An example which shows how it works : we use the s sommand. s means "substitute" or search and replace. The format is

```
s/regular-expression/replacement text/{flags}
```

We won't discuss all the flags yet. The one we use below is g which means "replace all matches"

```
>cat file
I have three dogs and two cats
>sed -e 's/dog/cat/g' -e 's/cat/elephant/g' file
I have three elephants and two elephants
>
```

OK. So what happened ? Firstly, *sed* read in the line of the file and executed

```
s/dog/cat/g
```

which produced the following text:

```
I have three cats and two cats
```

and then the second command was performed on the *edited line* and the result was

I have three elephants and two elephants

We actually have a name for the "current text": it is called the *pattern space*. So a precise definition of what *sed* does is as follows :

> *sed reads the standard input into the pattern space, performs a sequence of editing commands on the pattern space, then writes the pattern space to STDOUT.*

Getting Started: Substitute and delete Commands

Firstly, the way you usually use *sed* is as follows:

```
>sed -e 'command1' -e 'command2' -e 'command3' file
>{shell command}|sed -e 'command1' -e 'command2'
>sed -f sedscript.sed file
>{shell command}|sed -f sedscript.sed
```

so *sed* can read from a file or STDIN, and the commands can be specified in a file or on the command line. Note the following :

> that if the commands are read from a file, trailing whitespace can be fatal, in particular, it will cause scripts to fail for no apparent reason. I recommend editing sed scripts with an editor such as vim which can show end of line characters so that you can "see" trailing white space at the end of line.

The Substitute Command

The format for the substitute command is as follows:

> [*address1*[,*address2*]]s*/pattern/replacement/[flags]*

The flags can be any of the following:

n	replace nth instance of *pattern* with *replacement*
g	replace all instances of *pattern* with *replacement*
p	write pattern space to STDOUT if a successful substitution takes place
w *file*	Write the pattern space to *file* if a successful substitution takes place

If no flags are specified the first match on the line is replaced. note that we will almost always use the s command with either the g flag or no flag at all.

If one address is given, then the substitution is applied to lines containing that address. An address can be either a regular expression enclosed by forward slashes /regex/ , or a line number . The $ symbol can be used in place of a line number to denote the last line.

If two addresses are given separated by a comma, then the substitution is applied to all lines between the two lines that match the pattern.

This requires some clarification in the case where both addresses are patterns, as there is some

ambiguity here. more precisely, the substitution is applied to all lines from the first match of *address1* to the first match of *address2* and all lines from the first match of *address1 following* the first match of *address2* to the next match of *address1* Don't worry if this seems very confusing (it is), the examples will clarify this.

The Delete Command

The delete command is very simple in its syntax: it goes like this

[*address1* , *address2*]]d

And it deletes the content of the pattern space. All following commands are skipped (after all, there's very little you can do with an empty pattern space), and a new line is read into the pattern space.

Example 1

>cat file

http://www.foo.com/mypage.html

>sed -e 's@http://www.foo.com@http://www.bar.net@' file

http://www.bar.net/mypage.html

Note that we used a different delimiter, @ for the substitution command. Sed permits several delimiters for the s command including @%,;: these alternative delimiters are good for substitutions which include strings such as filenames, as it makes your sed code much more readable.

Example 2

>cat file

the black cat was chased by the brown dog

>sed -e 's/black/white/g' file

the white cat was chased by the brown dog

That was pretty straight forward. Now we move on to something more interesting.

Example 3

>cat file

the black cat was chased by the brown dog.
the black cat was not chased by the brown dog

```
>sed -e '/not/s/black/white/g' file

the black cat was chased by the brown dog.
the white cat was not chased by the brown dog.
```

In this instance, the substitution is only applied to lines matching the regular expression not. Hence it is not applied to the first line.

Example 4

```
>cat file

line 1 (one)
line 2 (two)
line 3 (three)
```

Example 4a

```
>sed -e '1,2d' file

line 3 (three)
```

Example 4b

```
>sed -e '3d' file
line 1 (one)
line 2 (two)
```

Example 4c

```
>sed -e '1,2s/line/LINE/' file

LINE 1 (one)
LINE 2 (two)
line 3 (three)
```

Example 4d

```
>sed -e '/^line.*one/s/line/LINE/' -e '/line/d' file

LINE 1 (one)
```

3a : This was pretty simple: we just deleted lines 1 to 2.
3b : This was also pretty simple. We deleted line 3.
3c : In this example, we performed a substitution on lines 1-2.
3d : now this is more interesting, and deserves some explanation. Firstly, it is clear that line 2

and 3 get deleted. But let's look closely at what happens to line 1.
First, line 1 is read into the pattern space. It matches the regular expression ^line.*one So the substitution is carried out, and the resulting pattern space looks like this:

LINE 1 (one)

So now the second command is executed, but since the pattern space does not match the regular expression line, the delete command is not executed.

Example 5

>cat file

hello
this text is wiped out
Wiped out
hello (also wiped out)
WiPed out TOO!
goodbye
(1) This text is not deleted
(2) neither is this ... (goodbye)
(3) neither is this
hello
but this is
and so is this
and unless we find another g**dbye
every line to the end of the file gets deleted

>sed -e '/hello/,/goodbye/d' file

(1) This text is not deleted
(2) neither is this ... (goodbye)
(3) neither is this

This illustrates how the addressing works when two pattern addresses are specified. sed finds the first match of the expression "hello", deleting every line read into the pattern space until it gets to the first line after the expression "goodbye". It doesn't apply the delete command to any more addresses until it comes across the expression "hello" again. Since the expression "goodbye" is not on any subsequent line, the delete command is applied to all remaining lines.

awk command

Remove duplicate lines using awk

```
$ awk '!($0 in array) { array[$0]; print }' temp
```

Print all lines from /etc/passwd that has the same uid and gid

```
$awk -F ':' '$3==$4' passwd.txt
```

Print only specific field from a file.

```
$ awk '{print $2,$5;}' employee.txt
```

diff command

To compare the contents of two files:

```
diff email addresses
2a3,4
> Jean JRS@pollux.ucs.co
> Jim  jim@frolix8
```

This displays a line by line difference between the file email and addresses.

To make these files match you need to add (a) lines 3 and 4 (3,4) of the file addresses (>) after line 2 in the file email.

Here are the contents of files email and addresses used in this example. Line numbers are shown at the beginning of each line to help you follow this example.

```
email                   addresses
1 John erpl08@ed          1 John erpl08@ed
2 Joe  CZT@cern.ch        2 Joe  CZT@cern.ch
3 Kim  ks@x.co            3 Jean JRS@pollux.ucs.co
4 Keith keith@festival    4 Jim  jim@frolix8
                          5 Kim  ks@x.co
                          6 Keith keith@festival
```

sort command

To sort the file

To sort the fruits file with the **LC_ALL**, **LC_COLLATE**, or **LANG** environment variable set to En_US, enter:

```
LANG=En_US sort fruits
```

This command sequence displays the contents of the fruits file sorted in ascending lexicographic order. The characters in each column are compared one by one, including spaces, digits, and special characters. For instance, if the fruits file contains the text:

```
banana
orange
Persimmon
apple
%%banana
apple
ORANGE
```

the **sort** command displays:

```
%%banana
ORANGE
Persimmon
apple
apple
banana
orange
```

In the ASCII collating sequence, the % (percent sign) precedes uppercase letters, which precede lowercase letters. If your current locale specifies a character set other than ASCII, your results may be different.

To sort in dictionary order

sort -d fruits
This command sequence sorts and displays the contents of the fruits file, comparing only letters, digits, and spaces. If the fruits file is the same as in example 1, then the **sort** command displays:

```
ORANGE
Persimmon
apple
apple
%%banana
banana
orange
```

The **-d** flag ignores the % (percent sign) character because it is not a letter, digit, or space, placing %%banana with banana.

To group lines that contain uppercase and special characters with similar lowercase lines

sort -d -f fruits
The **-d** flag ignores special characters and the **-f** flag ignores differences in case. With the **LC_ALL**, **LC_COLLATE**, or **LANG** environment variable set to C, the output for the fruits file becomes:

```
apple
apple
%%banana
banana
ORANGE
orange
Persimmon
```

To sort, removing duplicate lines

sort -d -f -u fruits
The **-u** flag tells the **sort** command to remove duplicate lines, making each line of the file unique. This command sequence displays:

```
apple
%%banana
```

ORANGE
Persimmon

Not only is the duplicate apple removed, but banana and ORANGE as well. These are removed because the **-d** flag ignores the %% special characters and the **-f** flag ignores differences in case.

1. To sort as in example 4, removing duplicate instances unless capitalized or punctuated differently, enter:
   sort -u +0 -d -f +0 fruits
   Entering the +0 -d -f does the same type of sort that is done with -d -f in example 3. Then the +0 performs another comparison to distinguish lines that are not identical. This prevents the **-u** flag from removing them.

   Given the fruits file shown in example 1, the added +0 distinguishes %%banana from banana and ORANGE from orange. However, the two instances of apple are identical, so one of them is deleted.

   apple
   %%banana
   banana
   ORANGE
   orange
   Persimmon

To specify the character that separates fields

sort -t: +1 vegetables
This command sequence sorts the vegetables file, comparing the text that follows the first colon on each line. The +1 tells the **sort** command to ignore the first field and to compare from the start of the second field to the end of the line. The -t: flag tells the **sort** command that colons separate fields. If vegetables contains:

yams:104
turnips:8
potatoes:15
carrots:104
green beans:32
radishes:5
lettuce:15

Then, with the **LC_ALL**, **LC_COLLATE**, or **LANG** environment variable set to C, the **sort** command displays:

carrots:104
yams:104
lettuce:15
potatoes:15
green beans:32
radishes:5
turnips:8

Note that the numbers are not in numeric order. This happened when a lexicographic

sort compares each character from left to right. In other words, 3 comes before 5, so 32 comes before 5.

To sort numbers

sort -t: +1 -n vegetables
This command sequence sorts the vegetables file numerically on the second field. If the vegetables file is the same as in example 6, then the **sort** command displays:

```
radishes:5
turnips:8
lettuce:15
potatoes:15
green beans:32
carrots:104
yams:104
```

To sort more than one field

sort -t: +1 -2 -n +0 -1 -r vegetables
OR
sort -t: -k2,2 n -k1,1 r vegetables
This command sequence performs a numeric sort on the second field (+1 -2 -n). Within that ordering, it sorts the first field in reverse alphabetic order (+0 -1 -r). With the **LC_ALL**, **LC_COLLATE**, or **LANG** environment variable set to C, the output looks like this:

```
radishes:5
turnips:8
potatoes:15
lettuce:15
green beans:32
yams:104
carrots:104
```

The command sorts the lines in numeric order. When two lines have the same number, they appear in reverse alphabetic order.

To replace the original file with the sorted text

sort -o vegetables vegetables
This command sequence stores the sorted output into the vegetables file (-o vegetables).

Sort a file in descending order

```
$ sort -r names.txt
```

export command

To view oracle related environment variables.

```
$ export | grep ORACLE
declare -x ORACLE_BASE="/u01/app/oracle"
declare -x ORACLE_HOME="/u01/app/oracle/product/10.2.0"
declare -x ORACLE_SID="med"
declare -x ORACLE_TERM="xterm"
```

To export an environment variable:

```
$ export ORACLE_HOME=/u01/app/oracle/product/10.2.0
```

gzip command

To create a *.gz compressed file:

```
$ gzip test.txt
```

To uncompress a *.gz file:

```
$ gzip -d test.txt.gz
```

Display compression ratio of the compressed file using gzip -l

```
$ gzip -l *.gz
         compressed        uncompressed  ratio uncompressed_name
              23709               97975  75.8% asp-patch-rpms.txt
```

bzip2 command

To create a *.bz2 compressed file:

```
$ bzip2 test.txt
```

To uncompress a *.bz2 file:

```
bzip2 -d test.txt.bz2
```

unzip command

To extract a *.zip compressed file:

```
$ unzip test.zip
```

View the contents of *.zip file (Without unzipping it):

```
$ unzip -l jasper.zip
Archive:  jasper.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
    40995  11-30-98 23:50   META-INF/MANIFEST.MF
    32169  08-25-98 21:07   classes_
```

```
15964  08-25-98 21:07   classes_names
10542  08-25-98 21:07   classes_ncomp
```

ps command

ps command is used to display information about the processes that are running in the system.

While there are lot of arguments that could be passed to a ps command, following are some of the common ones.

To view current running processes.

```
$ ps -ef | more
```

To view current running processes in a tree structure. H option stands for process hierarchy.

```
$ ps -efH | more
```

top command

top command displays the top processes in the system (by default sorted by cpu usage). To sort top output by any column, Press O (upper-case O) , which will display all the possible columns that you can sort by as shown below.

```
Current Sort Field:  P  for window 1:Def
Select sort field via field letter, type any other key to return

  a: PID        = Process Id            v: nDRT       = Dirty Pages count
  d: UID        = User Id               y: WCHAN      = Sleeping in Function
  e: USER       = User Name             z: Flags      = Task Flags
```

To displays only the processes that belong to a particular user use -u option. The following will show only the top processes that belongs to oracle user.

```
$ top -u oracle
```

df command

Displays the file system disk space usage. By default df -k displays output in bytes.

```
$ df -k
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/sda1            29530400   3233104  24797232  12% /
/dev/sda2           120367992  50171596  64082060  44% /home
```

df -h displays output in human readable form. i.e size will be displayed in GB's.

```
ramesh@ramesh-laptop:~$ df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/sda1            29G  3.1G   24G  12% /
/dev/sda2           115G   48G   62G  44% /home
```

Use -T option to display what type of file system.

```
ramesh@ramesh-laptop:~$ df -T
```

```
Filesystem    Type   1K-blocks      Used Available Use% Mounted on
/dev/sda1     ext4    29530400   3233120  24797216  12% /
/dev/sda2     ext4   120367992  50171596  64082060  44% /home
```

mount command

To mount a file system, you should first create a directory and mount it as shown below.

```
# mkdir /u01

# mount /dev/sdb1 /u01
```

You can also add this to the fstab for automatic mounting. i.e Anytime system is restarted, the filesystem will be mounted.

```
/dev/sdb1 /u01 ext2 defaults 0 2
```

chmod command

chmod command is used to change the permissions for a file or directory.

Give full access to user and group (i.e read, write and execute) on a specific file.

```
$ chmod ug+rwx file.txt
```

Revoke all access for the group (i.e read, write and execute) on a specific file.

```
$ chmod g-rwx file.txt
```

Apply the file permissions recursively to all the files in the sub-directories.

```
$ chmod -R ug+rwx file.txt
```

Following are the symbolic representation of three different roles:

u is for user,
g is for group,
and o is for others.

Following are the symbolic representation of three different permissions:

r is for read permission,
w is for write permission,
x is for execute permission.

Following are few examples on how to use the symbolic representation on chmod.

1. Add single permission to a file/directory

Changing permission to a single set. + symbol means adding permission. For example, do the following to give execute permission for the user irrespective of anything else:

```
$ chmod u+x filename
```

2. Add multiple permission to a file/directory

Use comma to separate the multiple permission sets as shown below.

```
$ chmod u+r,g+x filename
```

3. Remove permission from a file/directory

Following example removes read and write permission for the user.

```
$ chmod u-rx filename
```

4. Change permission for all roles on a file/directory

Following example assigns execute privilege to user, group and others (basically anybody can execute this file).

```
$ chmod a+x filename
```

5. Make permission for a file same as another file (using reference)

If you want to change a file permission same as another file, use the reference option as shown below. In this example, file2′s permission will be set exactly same as file1′s permission.

```
$ chmod --reference=file1 file2
```

6. Apply the permission to all the files under a directory recursively

Use option -R to change the permission recursively as shown below.

```
$ chmod -R 755 directory-name/
```

7. Change execute permission only on the directories (files are not affected)

On a particular directory if you have multiple sub-directories and files, the following command will assign execute permission only to all the sub-directories in the current directory (not the files in the current directory).

```
$ chmod u+X *
```

chown command

chown command is used to change the owner and group of a file. \

To change owner to oracle and group to db on a file. i.e Change both owner and group at the same time.

```
$ chown oracle:dba dbora.sh
```

Use -R to change the ownership recursively.

```
$ chown -R oracle:dba /home/oracle
```

passwd command

Change your password from command line using passwd. This will prompt for the old password followed by the new password.

```
$ passwd
```

Super user can use passwd command to reset others password. This will not prompt for current password of the user.

```
# passwd USERNAME
```

Remove password for a specific user. Root user can disable password for a specific user. Once the password is disabled, the user can login without entering the password.

```
# passwd -d USERNAME
```

su command examples

Switch to a different user account using su command. Super user can switch to any other user without entering their password.

```
$ su - USERNAME
```

Execute a single command from a different account name. In the following example, john can execute the ls command as raj username. Once the command is executed, it will come back to john's account.

```
[john@dev-server]$ su - raj -c 'ls'

[john@dev-server]$
```

Login to a specified user account, and execute the specified shell instead of the default shell.

```
$ su -s 'SHELLNAME' USERNAME
```

date command examples

Set the system date:

```
# date -s "01/31/2010 23:59:53"
```

Once you've changed the system date, you should syncronize the hardware clock with the system date as shown below.

```
# hwclock –systohc

 hwclock --systohc –utc
```