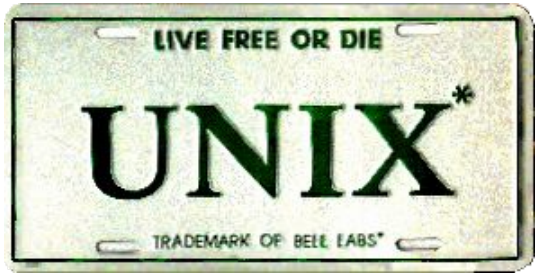


The UNIX CLI

Contents

Introduces the following utilities, listed in alphabetical order:

Starting with Unix

- **Logging In**

  – In order to use a UNIX system, you must first log in with a suitable username.

- A username is a unique name that distinguishes you from the other users of the system.

- Your username and initial password are assigned to you by the system administrator.

- UNIX first asks you for your username by prompting you with the line “login:” and then asks for your password.

Starting with Unix

- When you enter your password, the letters that you type are not displayed on your terminal for security reasons.
- UNIX is case sensitive, so make sure that the case of letters is matched exactly to those of your password.
- Depending on how your system is set up, you should then see either a $, a % or another prompt. That is your default shell prompting you to provide some course of action.
- Here's an example login :

```
UNIX®  System V  Release 4.0
login : umesh
Password :              --> What is typed here is secret and doesn't show.
Last login:  Sun  Feb  15  18:33:26  from mars.cse.iitb.ac.in
$ _
```

Shells

- The $ or % prompt that you see when you first log in is displayed by a special kind of program called a shell.
- A Shell is a program that acts as a middleman between you and the raw UNIX operating system.
- It lets you run programs, build pipelines of processes, save output to files, and run more that one program at the same time.
- A shell executes all of the commands that you enter.
- The four most popular shells are:

  – the Bourne shell (sh)
  – the Korn shell (ksh)
  – the C shell (csh)
  – the Bash Shell (bash)

Shells

- All of these shells share a similar set of core functionality, together with some specialized properties.
- The Korn shell is a superset of the Bourne shell, and thus users typically choose either the C shell or the Korn shell to work with. The Bash shell, or Bourne Again Shell, is an attempt to combine the best features from Korn and C shells. Bash is becoming the most popular shell, because it is freely available and comes as a standard shell on Linux systems.
- The shell languages are tailored to manipulating files and processes in the UNIX system, which makes them more convenient in many situations.

Running a utility

- To run a utility, simply enter its name at the prompt and press the Enter key.
- Pressing the Enter key tells UNIX that you've entered the command an that you wish it to be executed.
- One utility that every system has is called date, which displays the current date and time:

```
$ date                                  --> run the date utility.
Thu Aug 19 21:23:33 PDT 2004
$ _
```

Running another shell

- If you don’t like the shell, just type in the name of the one that you like and press Enter.

```
> echo $SHELL
/usr/bin/csh
> sh                    --> csh -> sh
$ bash                        --> sh -> bash
bash-2.03$ ksh                    --> bash -> ksh
$ ^D                          --> ^D == exit
bash-2.03$ exit ^D                --> back to bash
$ ^D                          --> back to sh
> _                       --> back to csh
```

date [yymmddhhmm[.ss]]

- Without any arguments, date displays the current date and times.

- If arguments are provided, date sets the date to the supplied setting, where:
  - yy is the last two digits of the year,
  - the first mm is the number of the month,
  - dd is the number of the day,
  - hh is the number of hours (using the 24-hour clock), and
  - the last mm is the number of minutes.

- The optional ss is the number of seconds. ***Only a super-user may set the date.***

clear

- This utility clears your screen.

man: online help

- All UNIX systems have a utility called man (short for manual page) that puts this information at your fingertips.

man [section] word
man -k keyword

- The manual pages are on-line copies of the original UNIX documentation, which is usually divided into eight sections. They contain information about utilities, system calls, file formats, and shells.

- The first usage of man displays the manual entry associated with word. If no section number is specified, the first entry that it finds is displayed.

- The second usage of man displays a list of all the manual entries that contain keyword.

Organization of the manual pages

- The typical division of topics in manual pages (sections) is as follows:

1. Commands and Application Programs.
2. System Calls
3. Library Functions
4. Special Files
5. File Formats
6. Games
7. Miscellaneous
8. System Administration Utilities

Using man

Here's an example of man in action:

```
$ man   -k   mode              ---> search for keyword "mode"
chmod (1V)    - change the permissions mode of a file
chmod, fchmod(2V)  - change mode of file
getty(8)                - set terminal mode
ieeeflags(3M)          - mode and status function
umask(2V)              - set file creation mode mask

$ man chmod                   ---> select the first manual entry.
CHMOD(1V)          USER    COMMANDS              CHMOD(1V)
NAME
      chmod - change the permissions mode of a file
SYNOPSIS
      chmod C -fR  V  mode filename …
                          --> the description of chmod goes here.
SEE  ALSO
       csh(1), ls(1V), sh(1), chmod(2V), chown(8)
```

Using man sections

```
$ man   -s   2   chmod              ---> select the manual entry from section 2.
  CHMOD(2V)  SYSTEM CALLS  CHMOD(2V)
  NAME
       chmod, fchmod  - change mode of file

  SYNOPSIS
       #include <sys/stat.h>
       int  chmod(path, mode)
       char *path;
       mode_t  mode;
  … the description of chmod() goes here.
  SEE ALSO
         chown(2V), open(2V), stat(2V), sticky(8)
 $ _
```

Using special characters

- Some characters are interpreted specially when typed at a UNIX terminal.
- These characters are sometimes called metacharacters and may be listed by using the stty utility with the -a (all)option.
- Here's an example of the use of the stty utility for listing metacharacters:

```
$ stty  -a                                  --> obtain terminal line settings
speed 38400 baud; rows 50; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

Meta-characters

- The carat(^) in front of each letter means that the Control key must be pressed at the same time as the letter.

- The default meaning of each option is as follows:

Meta-character	Meaning
erase	Backspace one character
kill	Erase all of the current line.
werase	Erase the last word.
rprnt	Reprint the line.
flush	Ignore any pending input and reprint the line.
lnext	Don't treat the next character specially.
susp	Suspend the process for a future awakening.
intr	Terminate (interrupt) the foreground job with no core dump.
quit	Terminate the foreground job and generate a core dump.
stop	Stop/restart terminal output.
eof	End of input.

Terminating a Process: Control-c

- There are often times when you run a program and then wish to stop it before it's finished.
- The standard way to execute this action in UNIX is to press the keyboard sequence Control-C.
- Most processes are immediately killed and your shell prompt is returned.
- Here's an example of the use of Control-C:

```
$ man  chmod
 CHMOD(1V)    USER    COMMANDS    CHMOD(1V)
  NAME
        chmod - change the permissions mode of a file
^C
$ _
```

Pausing Output to Terminal: Control-s/Control-q

- If the output of a process starts to rapidly scroll up the screen, you may pause it by pressing Control-S.

- To resume the output, you may either press Control-s again or press Control-q.

- This sequence of control characters is sometimes called XON/XOFF protocol.

- Here's an example of its use:

```
$ man chmod
 …
 ^s                    ---> suspend terminal output.
 ^q                    ---> resume terminal output.
 …
$ _
```

End of Input: Control-d

- You must tell the utility when the input from the keyboard is finished.
- To do so, press Control-D on a line of its own after the last line of input. Control-D signifies the end of input.

- For example, the mail utility allows you to send mail from the keyboard to a named user:

$ mail soumen@cse.iitb.ac.in --> send mail to my friend soumen.
Hi Soumen, --> input is entered from the keyboard.
I hope you get this piece of mail. How about building the decentralized internet one of these days?

- with best wishes from Umesh
^D --> tell the terminal that there's no more input.
$ _

Setting/Changing password

- After you first login to a UNIX system, it's a good idea to change your initial password. The password protects all your private information.
- You might be forced to change your password by the administrator.
- Remember however, that superuser can access everything.
- Passwords should generally be at least six letters long and should not be words from a dictionary or proper nouns. Some system administrators will put restrictions on the life span of passwords, so you have to exchange them periodically.
- The best is to use a mixed expression of letters, numbers and other characters, like "GWK#145W%".
- If you forget your password, the only thing to do is to contact your system administrator and ask for a new password.

Setting/Changing Password: passwd

- passwd allows you to change your password.
- You are prompted for your old password and then twice for the new one.
- The new password may be stored in an encrypted form in the password file "/etc/passwd" or in a "shadow" file (for more security), depending on your version of UNIX.

- An example of changing a password:

```
$ passwd
  Current password : penguin
  New password( ? For help ) : GWK145W          --> invisible
  New password( again ) : GWK145W          --> invisible
  Password changed for umesh
  $ _
```

• Note that you wouldn't normally be able to see the passwords, as UNIX turns off the keyboard echo when you enter them.

Logging out

- **To leave the UNIX system,** press the keyboard sequence **Control-D** at your shell prompt.

- This command tells your login shell that there is no more input for it to process, causing it to disconnect you from the UNIX system.

- Most systems then display a “login:” prompt and wait for another user to log in.

```
$ ^D
UNIX®  System V Release 4.0
login :                    --> wait for another user to log in.
```

- If you connect to a remote server through ssh connection, you will not see the new login prompt; instead, you will be disconnected.

User Home Directory

- Every UNIX process has a location in the directory hierarchy, termed its current working directory.
- When you log into a UNIX system, your shell starts off in a particular directory called your home directory.
- In general, every user has a different home directory, which often begins with the prefix “/home”.
- For example, my author’s home directory is called “/home/umesh”.
- The system administrator assigns these home-directory values.

Printing Working Directory: pwd

- To display your shell's current working directory, use the pwd utility, which works like this:

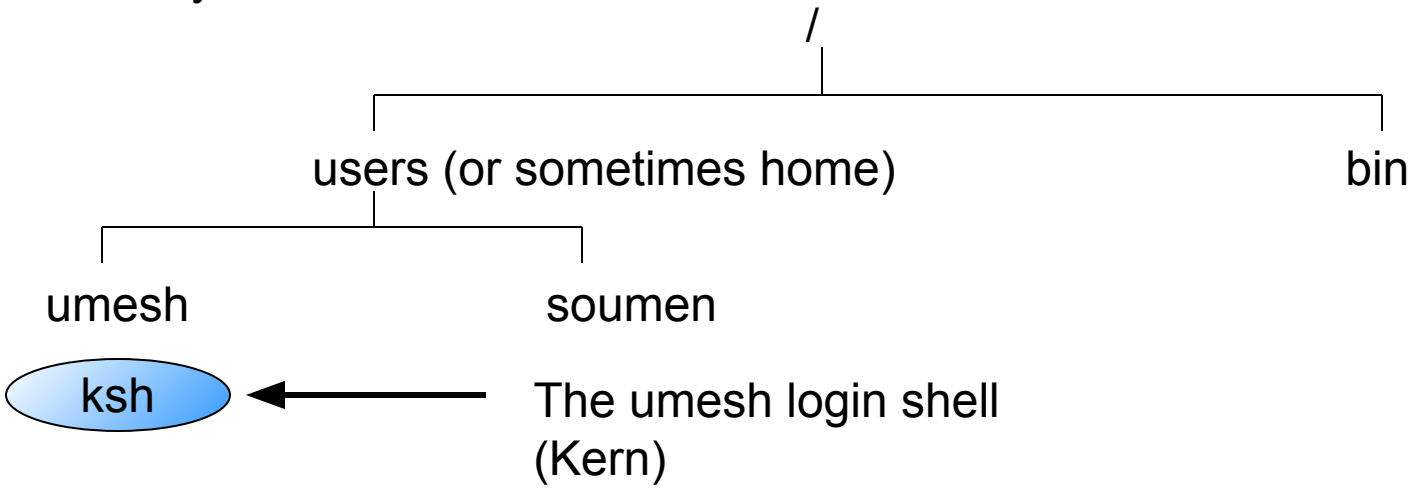
UNIX® System V Release 4.0

login : umesh
Password : ……

$pwd
/home/umesh
$ _

Directory Hierarchy, Home Directory and Login Shell

•Here's a diagram that indicates the location of our login Korn shell in the directory hierarchy:



Creating a file with cat

cat -n {fileName}*

- The cat utility takes its input from standard input or from a list of files and displays them to standard output.
- The -n option adds line numbers to the output. cat is short for “concatenate” which means “to connect in a series of links.”
- By default, the standard input of a process is from the keyboard and the standard output is to the screen.

```
$ cat  > heart      --> store keyboard input into a file called “heart”.
I hear her breathing,
I’m  surrounded by the sound.
Floating in this secret place,
I never shall be found.
^D                  --> tell cat that the end of input has been reached.
$ _
```

Listing Contents of a Directory: ls

- The **ls** utility, which lists information about a file or a directory.

ls -adgIsFR { fileName }* {directoryName}*

- ls lists all of the files in the current working directory in alphabetical order, excluding files whose names start with a period.
- The -a option causes such files to be included in the listing.
- The -d option causes the details of the directories themselves to be listed, rather than their contents.
- The -g option list a file's group.
- The -l option generates a long listing, including permission flags, the file's owner, and the last modification time.

Listing Contents of a Directory

- The -s option causes the number of disk blocks that the file occupies to be included in the listing. (A block is typically between 512 and 4K bytes.)

- The -F option causes a character to be placed after the file's name to indicate the type of the file:
  - * means an executable file, / means a directory file,
  - @ means a symbolic link, and = means a socket.

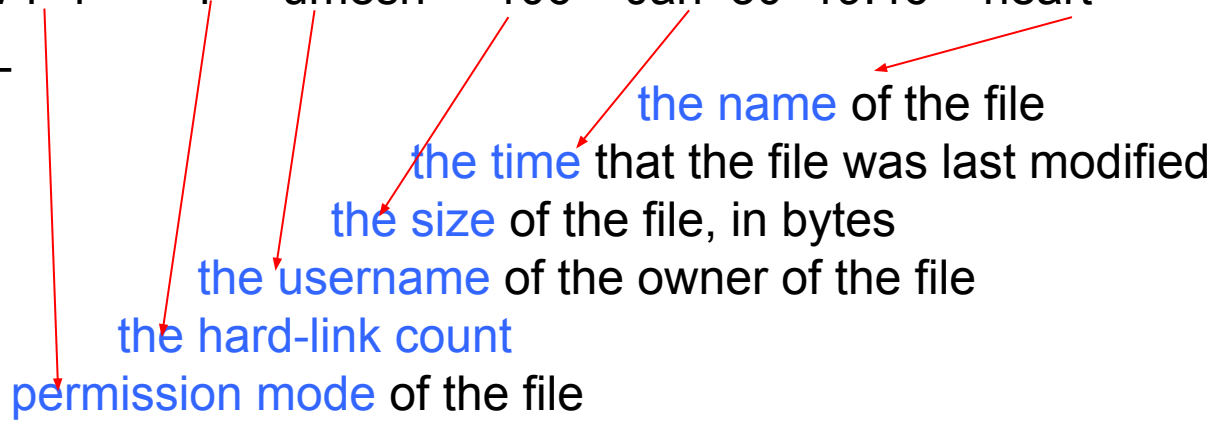
- The -R option recursively lists the contents of a directory and its subdirectories.

Directory Listing

•Here's an example of the use of ls :

```
$ ls                    --> list all files in current directory.
heart

$ ls  -l  heart                 --> long listing of "heart."
-rw-r--r--     1     umesh     106     Jan  30  19:46     heart
$ _
```



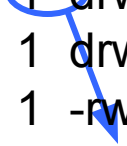
Directory Listing

Field #	Field value	Meaning
1	-rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file
2	1	the hard-link count
3	umesh	the username of the owner of the file
4	106	the size of the file, in bytes
5	Jan 30 19:46	the time that the file was last modified
6	heart	the name of the file

Listing Contents of a Directory

- You may obtain even more information by using additional options:

```
$ ls  -algFs                    --> extra-long  listing of current dir
total  3                        --> total number of blocks of storage.
1  drwxr-xr-x     3   umesh   cs   512    Jan   30  22:52 ./
1  drwxr-xr-x    12   root    cs   1024   Jan   30  19:45 ../
1  -rw-r--r--     1   umesh   cs   106    Jan   30  19:46 heart
$ _
```



- The -s option generates an extra first field, which tells you how many disk blocks the file occupies.

- On some UNIX systems, each disk block is 1024 bytes long, which implies that a 106-byte file actually takes up 1024 bytes of physical storage.

Displaying a File: cat

- cat with the name of the file that you wanted to display:

```
$ cat  heart                --> list the contents of the “heart” file.
I  hear her breathing.
I’m  surrounded by the sound.
Floating in this secret place,
I never shall be found.
$ _
```

- cat is good for listing the contents of small files, but it doesn’t pause between full screens of output.

Displaying a File: more

more -f [+lineNumber] { fileName }*

- The more utility allows you to scroll a list of files, one page at a time.
- By default, each file is displayed starting at line 1, although the +option may be used to specify the starting line number.
- The -f option tells more not to fold (or wrap) long lines.

- After each page is displayed, more displays the message "--more--" to indicate that it's waiting for a command.

- To list the next page, press the space bar.
- To list the next line, press the Enter key.
- To quit from more, press the "q" key.
- ^B will display the previous page
- H will display help page
- Try:

```
$ ls –la /usr/bin > myLongFile
$ more myLongFile
```

Displaying a File: page

page -f [+lineNumber] { fileName }*

- The page utility works just like more, except that it clears the screen before displaying each page.

- This feature sometimes makes the listing a little quicker.

- Try:

$ page myLongFile

Displaying a File: head and tail

head -n { fileName }*

- The head utility displays the first n lines of a file. If n is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

tail -n { fileName }*

- The tail utility displays the last n lines of a file. If n is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

- The first two lines and last two lines of my “heart” file.

```
$ head -2 heart                 -->  list the first two lines.
I hear her breathing,
I’m surrounded by the sound.
$ tail -2 heart                 -->  list the last two lines.
Floating in this secret place,
I never shall be found.
$ head -15 myLongFile           --> see what happens
```

Renaming/Moving a File: mv

mv -i oldFileName newFileName
mv -i {fileName}* directoryName
mv -i oldDirectoryName newDirectoryName

- The first form of mv renames oldFileName as newFileName.
- The second form allows you to move a collection of files to a directory.
- The third form allows you to move an entire directory.
- The -i option prompts you for confirmation if newFileName already exists so that you do not accidentally replace its contents. You should learn to use this option (or set a convenient shell alias that replaces "mv" with "mv –i"; we will come back to this later).

Renaming/Moving Files: mv

•Here's how to rename the file using the first form of the mv utility:

```
$ mv  heart   heart.ver1                --> rename to "heart.ver1".
$ ls
heart.ver1
$ _
```

•We will see other uses (moving files) shortly.

Making Directory: mkdir

mkdir -p newDirectoryName

- The mkdir utility creates a directory. The -p option creates any parent directories in the newDirectoryName pathname that do not already exist.

- If newDirectoryName already exists, an error message is displayed and the existing file is not altered in any way.

```
$ mkdir lyrics          --> creates a directory called “lyrics”.
$ ls –lF                --> check the directory listing in order
                        --> to confirm the existence of the
                        --> new directory.
-rw-r--r--    1   umesh   106   Jan  30  23:28    heart.ver1
drwxr-xr-x    2   umesh   512   Jan  30  19:49    lyrics/
$ _
```

Moving Files

- Once the “lyrics” directory is created, we can move the “heart.ver1” file into its new location. To do so, used mv and confirm the operation using ls:

```
$ mv heart.ver1  lyrics          --> move into “lyrics”.
$ ls                             --> list the current directory.
lyrics/                          --> “heart.ver1” has gone.
$ ls lyrics                      --> list the “lyrics” directory.
heart.ver1                       --> “heart.ver1” has moved.
$ _
```

Changing Directories: cd

- **cd [directoryName]**

- The following might be inconvenient; especially if we deal with large hierarchy:

```
$ vi   lyrics/heart.ver1          --> invoke the vi editor
```

- Instead, change directory:

```
$ cd  lyrics                  --> change directory
$ vi   heart.ver1                   --> invoke the vi editor
```

- The cd shell command changes a shell's current working directory to be directoryName.

- If the directoryName argument is omitted, the shell is moved to its owner's home directory.

Reorganizing Directories

```
$ pwd                          --> display where I am
/home/umesh
$ cd lyrics                    --> move into the “lyrics” directory
$ pwd
/home/umesh/lyrics
$ cd ..                        --> move up one level
$ pwd                          --> display new position
/home/umesh
$ cd lyrics                    --> move into the “lyrics” directory
$ pwd
/home/umesh/lyrics
$ ls ~/                        --> “~/” refers to home directory
/home/umesh
$ _
```

Copying Files: cp

- To copy the file, I used the cp utility, which works as follows:

cp -i oldFileName newFileName
cp -ir { fileName }* directoryName

- The first form of cp copies the contents of oldFileName to newFileName.

- If the label newFileName already exists, its contents are replaced by the contents of oldFileName.

- The -i option prompts you for confirmation if newFileName already exists so that you do not accidentally overwrite its contents. Like with mv, it is a good idea to use this option or create an alias.

Copying Files: cp

- The -r option causes any source files that are directories to be recursively copied, thus copying the entire directory structure.

- cp actually does two things
  - It makes a physical copy of the original file's contents.
  - It creates a new label in the directory hierarchy that points to the copied file.

```
$ cp heart.ver1  heart.ver2 --> copy to "heart.ver2".
$ ls -l heart.ver1  heart.ver2      --> confirm the existence of both files.
-rw-r--r--   1   umesh   106   Jan  30  23:28   heart.ver1
-rw-r--r--   1   umesh   106   Jan  31    00:12   heart.ver2
$ cp -i heart.ver1  heart.ver2     --> what happens?
```

Reorganizing Directories (again)

```
$ cd                    --> move back to my home directory
$ mkdir  lyrics.final           --> make the final lyrics directory
$ mv lyrics  lyrics.draft       --> rename the old lyrics dir
$ _
```

Deleting a Directory: rmdir

rmdir { directoryName }+

- The rmdir utility removes all of the directories in the list of directory names provided in the command. A directory must be empty before it can be removed.

- To recursively remove a directory and all of its contents, use the rm utility with the -r option (see next slide).

- Here, we try to remove the “lyrics.draft” directory while it still contains the draft versions, so we receive the following error message:

```
$ rmdir  lyrics.draft
rmdir : lyrics.draft : Directory not empty.
$ _
```

Deleting Directories: rm -r

- The rm utility allows you to remove a file's label from the hierarchy.
- Here's a description of rm:

rm -fir {fileName}*

- The rm utility removes a file's label from the directory hierarchy.
- If the filename doesn't exist, an error message is displayed.
- The -i option prompts the user for confirmation before deleting a filename. It is a very good idea to use this option or create a shell alias that translates from "rm" to "rm –i". If you don't, you will loose some files one day – you have been warned!
- If fileName is a directory, the -r option causes all of its contents, including subdirectories, to be recursively deleted.
- The -f option inhibits all error messages and prompts. It overrides the –i option (also one coming from an alias). This is dangerous!

Removing Directories

- To remove every file in the “lyrics.draft” directory, we move into the “lyrics.draft” directory and use rm:

```
$ lcp lyrics.draft/heart.ver2 lyrics.final/heart.final
$ cd  lyrics.draft          --> move to “lyrics.draft” directory
$ rm heart.ver1   heart.ver2
$ ls                        --> nothing remains
$ _
```

- To erase the draft directory:

```
$ cd                        --> move to my home directory
$ rmdir  lyrics.draft               --> this time it works
$ _
```

- To erase a collection of files:

```
$ cd  lyrics.draft          --> move into “lyrics.draft” directory
$ rm *                              --> erase all files in the current directory
```

Removing Directories with Files

- The -r option of rm can be used to delete the “lyrics.draft” directory and all of its contents with just one command:

```
$ cd                          --> move to my home directory.
$ rm  -r   lyrics.draft             --> recursively delete directory.
$ _
```

Printing Files: lp, lpstat and cancel

lp [-d destination] [-n copies] {fileName}*

- lp prints the named file(s) to the printer specified by the -d option. You get the name of the printer from the system administrator.
- If no files are specified, standard input is printed instead.
- By default, one copy of each file is printed, although this default may be overridden by using the -n option to specify the number of copies.

lpstat [destination]

- lpstat displays the status of all print jobs sent to any printer with the lp command.
- If a printer destination is specified, lpstat reports queue information for that printer only.
- lpstat displays information about the user, the name and size of the job, and a print-request ID.

Canceling a Print Job: cancel

cancel {request-ID}+

- cancel removes all of the specified jobs from the printer queue.
- If you're a super-user, then you may cancel any queued job, even if it was ordered by someone else.

```
$ lp -d lwcs heart.final                    --> order a printout.
request  id is lwcs-37( 1 file )
$ lpstat lwcs                               --> look at the printer status.
printer queue for lwcs
lwcs-36        ables  priority  0  Mar  18  17:02  on lwcs          inventory.txt  457 bytes
lwcs-37        umesh          priority  0  Mar  18  17:04  on lwcs  heart.final
$ cancel lwcs-37                            --> look at the printer status.
$ _
```

Print Files (BSD, Linux): lpr, lpq and lprm

lpr –m [-Pprinter] [-#copies] {fileName}*

- lpr prints the named file(s) to the printer specified by the -P
- option. If no printer is specified, then the value of $PRINTER
- environment variable is used.

- If no files are specified, standard input is printed instead.

- By default, one copy of each file is printed, although this default may be overridden by using the -# option to specify the number of copies.

- To receive mail when the printing job is done, -m option is used.

Querying Print Jobs: lpq

lpq -l [-Pprinter] {job#}* {userId}*

- lpq displays the status of all print jobs sent to the printer referenced by the –P option (or $PRINTER environment variable there is no -P option). The scope can be limited to just specified jobs and/or users.

- lpq displays information about the user, the name and size of the job, and a print-request ID.

- The –l option can be used to generate extra information.

Removing Print Jobs: lprm

lprm [-Pprinter] [-] {jobs}* {userId}*

- lprm removes all of the specified jobs from the queue of the printer specified by the –P option (or by $PRINTER environmnet variable if no printer is specified). The scope of the command can be controlled by specifying jobs to remove.
- The – option removes all jobs requested by the user issuing the command.
- If you're a super-user, then you may cancel any queued job, even if it was ordered by someone else by specifying the user Id.

```
$ lpr -Plwcs heart.final                --> order a printout
request  id is lwcs-37( 1 file )
$ lpq -Plwcs umesh                      --> look at the printer status
lwcs is ready and printing
Rank    Owner   Job     File(s)                 Total Size
1st     umesh    25     heart.final              106 bytes
$ lprm -Plwcs 25 umesh                  --> remove the job
```

Counting Lines, Words and Characters in Files: wc

wc -lwc {fileName}*

- The wc utility counts the number of lines, words, and/or characters in a list of files.
- If no files are specified, standard input is used instead.

- The -l option requests a line count,
- the -w option requests a word count,
- and the -c option requests a character count.

- If no options are specified, then all three counts are displayed.
-
- A word is defined by a sequence of characters surrounded by tabs, spaces, or new lines.

Counting Lines, Words and Characters in Files: wc

- **For example, to count lines, words and characters in the “heart.final” file, we used:**

```
$ cd  ~/lyrics.final
$ wc  heart.final                --> obtain a count of the number of lines,
                                          --> words, and characters.
      9      43     213    heart.final
$ _
```

File Attributes

•We u**sed** ls to obtain a long listing of “heart.final” and got the following output:

```
$ ls  -lgsF  heart.final
1  -rw-r--r--    1    umesh    cs    213    Jan    31  00:12  heart.final
$ _
```

File Attributes

Field #	Field value	Meaning
1	1	the number of blocks of physical storage occupied by the file
2	-rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file
3	1	the hard-link count
4	umesh	the username of the owner of the file
5	cs	the group name of the file
6	213	the size of the file, in bytes
7	Jan 31 00:12	the time that the file was last modified
8	heart.final	the name of the file

File Attributes

- **File Storage**
  - The number of blocks of physical storage taken up by the file is shown in field 1 and is useful if you want to know how much actual disk space a file is using.

- **Filenames**
  - The name of the file is shown in field 8. A UNIX filename may be up to 255 characters in length.

  - The only filenames that you definitely can't choose are "." and "..", as these names are predefined filenames that correspond to your current working directory and its parent directory, respectively. One can use " " (blank, space) in file names, but it is difficult to deal with files like that. Using special characters will definitely confuse your shell at some point.

- **Time of Last File Modification**
  - the time that the file was last modified and is used by several utilities.

File Attributes

- **File Owner**
  - Field 3 tells you the owner of the file. Every UNIX process has an owner, which is typically the same as the username of the person who started it.
  - the string of text known as the username is typically how we refer to a user, UNIX represents this identity internally as an integer known as the user ID.
  - The username is easier for humans to understand than a numeric ID.
- **File Group**
  - Field 5 shows the file's group. Every UNIX user is a member of a group. This membership is initially assigned by the system administrator and is used as part of the UNIX security mechanism.

File Attributes

- **File Types**
  - Field 2 describes the file's type and permission settings.

- In ls –lgsF example:

- 1 -rw-r--r-- 1 umesh cs 213 Jan 31 00:12 heart.final

- The first character of field 2 indicates the type of file, which is encoded as follows :

character	File Type
-	regular file
d	directory file
b	buffered special file(such as a disk drive)
c	unbuffered special file(such as a terminal)
l	symbolic link
p	pipe
s	socket

Determining Type of a File: file

file { fileName }+

- The file utility attempts to describe the contents of the fileName argument(s), including the language in which any of the text is written.

- file is not reliable; it may get confused.

- When file is used on a symbolic-link file, file reports on the file that the link is pointing to, rather than on, the link itself.

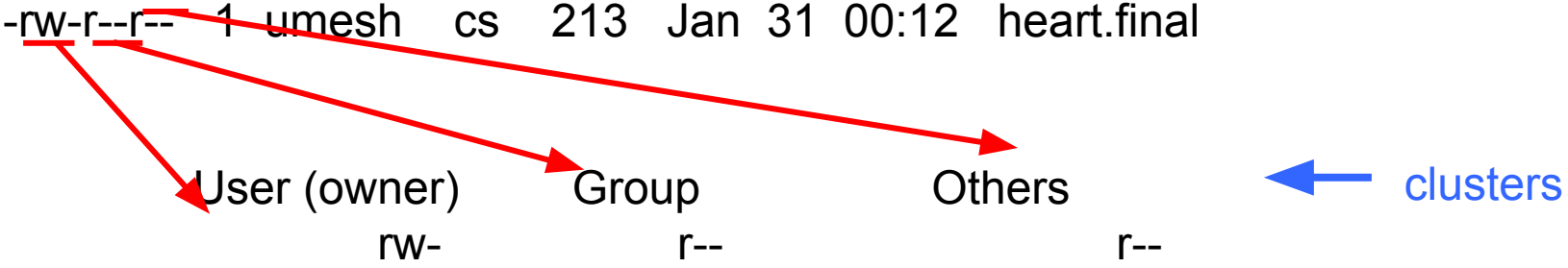
- For example,

```
$ file heart.final                    --> determine the file type.
heart.final: ascii text
$ _
```

File Permissions (Security)

•File permissions are the basis for file security. They are given in three clusters. In the example, the permission settings are “rw-r--r--”:

• 1 -rw-r--r-- 1 umesh cs 213 Jan 31 00:12 heart.final



Each cluster of three letters has the same format:

Read permission	Write permission	Execute permission
r	w	x

File Permission

•The meaning of the read, write, and execute permissions depends on the type of file:

	Regular file	Directory file	Special file
Read	read the contents	read the directory (list the names of files that it contains)	read from the file using the read() system call.
Write	change the contents	Add or remove files to/from the directory	write to the file using the write() system calls.
Execute	execute the file if the file is a program	access files in the directory	No meaning.

File Security

- When a process executes, it has four values related to file permissions:
  1. A real user ID
  2. An effective user ID
  3. A real group ID
  4. An effective group ID

- When you log in, your login shell process has its real and effective user IDs set to your own user ID and its real and effective group IDs set to your group ID.

1. When a process runs, the file permissions apply as follows:

2. If the process' effective user ID is the same as the owner of the file, the **User** permission apply.

- If the process' effective user ID is different from the owner of the file, but its effective group ID matches the file's group ID, then the Group permissions apply.

File Security

- If neither the process's effective user ID nor the process' effective group ID matches the owner of the file and the file's group ID, respectively, the Others permission apply.

- When an executable with "**set user ID**" permission is executed, the process' effective user ID becomes that of the executables.

- Similarly, when an executable with "**set group ID**" permission is executed, the process' effective group ID becomes that of the executable.

- "Set user ID" and "set group ID" permissions are indicated by an "s" instead of an "x" in the user and group clusters, respectively.

- They may be set using the **chmod** utility.

- The only way to create a new group is to ask the system administrator to add it.
- After a new group is added, any user who wants to be a part of that group must also ask the system administrator.

File Security

- A few other notes relating to file permissions:

- When a process creates a file, the default permissions given to that file are modified by a special value called the umask.

- The umask value is usually set to a sensible default, so we will wait until later to discuss it further.

- It's perfectly possible, although unusual, for the owner of a file to have fewer permissions than the group or anyone else.

- Hard-Link Count
  - Field 3 shows the file's hard-link count, which indicates how many labels in the hierarchy are pointing to the same physical file.
  - Hard links are rather advanced and are discussed in conjunction with the ln utility.

Listing File Group: groups

•groups

•The groups utility allows you to list all of the groups that you're a member of, and it works like this:

groups [userId]

•When invoked with no arguments, the group utility displays a list of all of the groups that you are a member of.

•If the name of a user is specified, a list of the groups to which that user belongs are displayed.

•Here's what we saw when we executed the groups utility:

```
$ groups                    --> list my groups
      cs          music
$ _
```

Changing File Group: chgrp

•Changing a File's group : chgrp

chgrp -R groupname { fileName }*

•The chgrp utility allows a user to change the group of files that he/she owns.

•A super-user can change the group of any file.

•All of the files that follow the groupname argument are affected.

•**The -R option** recursively changes the group of the files in a directory.

Changing File Group: example

```
$ ls -lg heart.final
-rw-r--r--   1   umesh   cs   213  Jan  31  00:12  heart.final
$ chgrp  music heart.final                --> change the group.
$ ls -lg heart.final                      --> confirm it changed.
-rw-r--r--    1   umesh   music  213  Jan  31  00:12  heart.final
$ _
```

- You may also use the chgrp utility to change the group of a directory.

Change File Permissions: chmod

chmod -R change{, change}* {fileName }+

- The chmod utility changes the modes (permissions) of the specified files according to the change parameters, which may take the following forms:

clusterSelection+newPermissions (add permissons)
clusterSelection-newPermissions (subtract permissions)
clusterSelection=newPermissions (assign permissions absolutely)

- where clusterSelection is any combination of:
  - u (user/owner)
  - g (group)
  - o (others)
  - a (all)

  and newPermissions is any combination of
  - r (read)
  - w (write)
  - x (execute)
  - s (set user ID/set group ID)

Changing File Permissions

- The **-R** option recursively changes the modes of the files in directories.

- Note that changing a directory's permission settings doesn't change the settings of the files that it contains.

- To remove read permission from others, we used **chmod** as follows:

```
$ ls -lg heart.final          --> to view the settings before the change.
-rw-r-----   1    umesh      music  213  Jan  31  00:12  heart.final
$ chmod g-r heart.final
$ ls -lg heart.final
-rw-------   1    umesh      music  213  Jan  31  00:12  heart.final
$ _
```

Changing File Permissions: examples

Requirement	Change parameters
Add group write permission	g+w
Remove user read and write permission	u-rw
Add execute permission for user, group, and others.	a+x
Give the group read permission only.	g=r
Add writer permission for user, and remove group read permission.	u+w,g-r

Changing File Permission: examples

•Here's an example of how to set these permissions:

```
$ cd                    --> change to home directory.
$ ls  -ld   .           --> list attributes of home directory.
drwxr-xr-x  45   umesh   4096  Apr   29   14:35
$ chmod   o-rx              --> update permissions.
$ ls  -ld   .           --> confirm.
drwxr---    45   umesh   4096  Apr  29  14:35
$ _
```

Changing File Permissions Using Octal Numbers

- The chmod utility allows you to specify the new permission setting of a file as an octal number.
- Each octal digit represents a permission triplet.

For example, if you wanted a file to have the permission settings of

rwxr-x--

then the octal permission setting would be 750, calculated as follows:

	User	Group	Others
setting	rwx	r-x	--
binary	111	101	000
octal	7	5	0

Changing File Permissions Using Octal Numbers

- The octal permission setting would be supplied to chmod as follows:

```
$ chmod  750   .                --> update permissions.
$ ls  -ld    .                  --> confirm.
drwxr-x---      45   umesh     4096    Apr  29 14:35
$ _
```

Changing File Owner: chown

chown -R newUserId {fileName}+

- The chown utility allows a super-user to change the ownership of files (some Unix versions allow the owner of the file to reassign ownership to another user). All of the files that follow the newUserId argument are affected.

- The -R option recursively changes the owner of the files in directories.

- Example: change the ownership of “heart.final” to “soumen” and then back to “umesh” again:

```
$ ls -lg heart.final              --> to view the owner before the change.
-rw-r-----  1 umesh  music 213 Jan 31 00:12 heart.final
$ chown soumen heart.final              --> change the owner to “soumen”.
$ ls -lg heart.final        --> to view the ownership after the change.
-rw-r-----  1 soumen    music 213 Jan 31 00:12 heart.final
$ chown umesh heart.final  --> change the owner back to “umesh”.
$ _
```

Change User Groups: newgrp

- When a process creates a file, the group ID of the file is set to the process' effective group ID, which means that when you create a file from a shell, the group ID of the file is set to the effective group ID of your shell.

- The system administrator is the one who chooses which one of your groups is used as your login shell's effective group ID. The only way to permanently alter your login shell's effective group ID is to ask the system administrator to change it.

newgrp [-][groupname]

- The newgrp utility, when invoked with a group name as an argument, creates a new shell with an effective group ID corresponding to the group name. The old shell sleeps until you exit the newly created shell.

- You must be a member of the group that you specify.

- If you use a dash(-) instead of a group name as the argument, a shell is created with the same settings as those of the shell that was created when you logged into the system.

Changing Groups: example

```
$ date > test1                  --> create from a “cs” group shell.
$ newgrp  music                 --> create a “music” group shell.
$ date > test2                  --> create from a “music” group shell.
^D
$ ls  -lg    test1   test2      --> look at each file’s attributes.
-rw-r--r--        1    umesh    cs       29  Jan  31   22:57    test1
-rw-r--r--        1    umesh    music    29  Jan  31   22:57    test2
$ _
```

Terminal Type

- Several UNIX utilities, including the two standard editors vi and emacs, need to know what kind of terminal you're using so that they can control the screen correctly.

- The type of your terminal is stored by your shell in something called an environment variable.

- Before vi or emacs can work correctly, your shell's TERM environment variable must be set to your terminal type.

- Default terminal type is "unknown".

- Usually, you do not need to worry about the terminal settings, because in modern Unix systems terminal is set automatically from a terminfo database that maintains compiled versions of files describing terminal capabilities for many terminals. Use infocmp to decompile data for your terminal.

- On BSD systems, you may need to explore tset and stty commands.

Editing Files with vi

Editing Files with vi

- The two most popular UNIX text editors are called vi and emacs.
- It's handy to be reasonably proficient in vi, as it is found on nearly every version of UNIX.
- The vi editor was originally developed for BSD UNIX by Bill Joy of the University of California at Berkeley.
- This is a standard utility for System V and most other versions of UNIX.
- "vi" stands for "visual editor".
- To edit an existing file, supply the name of the file as a command-line parameter.

```
$ vi poem.txt
```

- vi then enters command mode and awaits instructions.

Editing a File: vi

```
Line 1
Line 2
Line 3
Line 4
Line 5
I always remember standing in the rains,
On a cold and damp september,
Brown Autumn leaves were falling softly to the ground,
Like the dreams of a life as they slide away.
~
~
~
~
"poem.txt" [noeol] 4L, 176C                                   1,1           All
```

Text Entry Mode in vi

- To enter text-entry mode from command mode, press one of the keys in the table below.
- Each key enters you into text-entry mode in a slightly different way:

Key	Action
i	Text is inserted in front of the cursor.
I	Text is inserted at the beginning of the current line.
a	Text is added after the cursor.
A	Text is added to the end of the current line.
o	Text is added after the current line.
O	Text is inserted before the current line.
R	Text is replaced (overwritten).
:	Enter an extended command.

vi: Command Mode

- To edit text, you must enter command mode.
- To transfer from text-entry mode to command mode, press the Esc key.
- If you accidentally press the Esc key when in command mode, nothing bad happens.
- A lot of editing features require parameters and are accessed by pressing the colon (:) key, followed by the command sequence, followed by the Enter key.
- When the colon key is pressed, the remainder of the command sequence is displayed at the bottom of the screen.
- the Enter key is indicated as <Enter>.
  - The “<“ and “>” characters act as delimiters and should not be entered.

vi: Command Mode

•For example, to delete lines one through three, you'd enter the following command sequence:

:1,3d<Enter>

•vi allows you to use the "$" to denote the line number of the last line in the file and the "." to denote the line number of the line currently containing the cursor.

For example, the sequence

:.,.+2d<Enter>

would delete the current line and the two lines that follow it.

vi: Line Ranges

•Here are some other examples of commands for line ranges:

Range	Selects
1,$	all of the lines in the file
1,.	all of the lines from the start of the file to the current line, inclusive
.,$	all of the lines from the current line to the end of the file, inclusive
.-2	the single line that's two lines before the current line

vi: Comman Editing Features

•The most common vi editing features can be grouped into the following categories:

•cursor movement
•deleting text
•replacing text
•pasting text
•searching text
•search/replacing text
•saving/loading files
•miscellaneous (including how to quit vi)

vi: Cursor Movements

•Here is a table of the common cursor-movement commands:

Movement	Key sequence
Up one line	Arrow Up or the “k” key
Down one line	Arrow Down or the “j” key
Right one character	Arrow Right or the “l” key (will not wrap around)
Left one character	Arrow Left or the “h” key (will not wrap around)
To start of line	^
To end of line	$
Back one word	the “b” key
Forward one word	the “w” key
Down a half screen	Control-d
Forward one screen	Control-f
Up a half screen	Control-u
Back one screen	Control-b
To line nn	:nn<Enter> (nn G also works)

vi: Deleting Text

Here is a table of the common text-deletion commands:

Item to delete	Key sequence
Character	Position the cursor over the character and the press “x” key.
Word	Position the cursor at the start of word and then type the two character “dw” .
Line	Position the cursor anywhere on the line and then type the two characters “dd” (Typing a number ahead of “dd” will cause vi to delete the specified number of Lines beginning with the current line.)
Current position to end of current line	Press the “D” key.
Block of lines	:<range>d<Enter>

vi: Replacing Text

Following is a table of the common text-replacement commands:

Item to replace	Key sequence
Character	Position the cursor over the character, press “r” key, and then type the replacement character.
Word	Position the cursor at start of word, type the two characters “cw”, type the replacement text and press the Esc key.
Line	Position the cursor anywhere on the line, type the two characters “cc”, type the replacement text, and press the Esc key.

vi: Pasting Text

Here is a table of the most common pasting operations:

Action	Key sequence
Copy(yank) lines into paste buffer.	:<range>y<Enter>
Insert (put) paste buffer after current line.	p or :pu<Enter> (contents of paste buffer unchanged)
Insert paste buffer after line nn	:nnpu<Enter> (contents of paste buffer unchanged)

- For example, to copy the first two lines of the poem into the paste buffer and then paste them after the third line, I entered the following two commands:

```
:1,2y
:3pu
```

vi: Searching

•Here is a table of the most common search operations:

Action	Key Sequence
Search forward from current position for string sss.	/sss/ <Enter>
Search backward from current position for string sss.	?sss? <Enter>
Repeat last search.	n
Repeat last search in the opposite direction	N

•For example, I searched for the substring “ark” in line one of the poem by entering the following commands:

:1<Enter>
/ark/<Enter>

vi: Searching/Replacing

•You may perform global search-and-replace operations by using the following commands:

Action	Key Sequence
Replace the first occurrence of sss on each line with ttt.	:<range>s/sss/ttt/<Enter>
Replace every occurrence of sss on each line with ttt (global replace).	:<range>s/sss/ttt/g<Enter>

vi: Searching/Replacing

- For exampe, to replace every occurrence of the substring “re” with “XXX”, we enter the command displayed below:

I XXXmember walking in the rain,
On a cold and dark September,
Brown Autumn leaves weXXX falling softly to the ground,
Just like the dXXXams of a life as they slip away.
~
:1,$s/re/XXX/g

vi: Saving/Loading Files

•Here is a table of the most common save/load file commands:

Action	Key Sequence
Save file as <name>.	:w<name><Enter>
Save file with current name.	:w<Enter>
Forced save into a file that exists	:w!<Enter>
Save only certain lines to another file. :<range>w<name><Enter>	
Read in contents of another file at current position	:r<name><Enter>
Edit file <name> instead of current file.	:e<name><Enter>
Edit next file on initial command line.	:n<Enter>

vi: Saving Files

- For example, we saved the poem in a file called “rain.doc” by entering the command displayed below:

```
I  remember walking in the rain,
On a cold and dark September,
Brown Autumn leaves were falling softly to the ground,
Just like the dreams of a life as they slip away.
~
:w rain.doc
```

vi: Miscellaneous Commands

•Here's a list of the most common miscellaneous commands, including the commands for quitting vi:

Action	Key Sequence
Redraw screen.	Control-L
Execute command in a subshell and then return to vi.	:!<command><Enter>
Execute command in a subshell and read its output into the edit buffer at the current position.	:r!<command><Enter>
Repeat the last command.	:.<Enter>
Repeat the next command nn times.	nn<command>
Quit vi if work is saved.	:q<Enter>
Quit vi and discard unsaved work.	:q!<Enter>

Customizing vi

• The list of options varies between implementation. To get the list issue the following vi command:

:set all

•to see the list. You can set a specific option using :set as well; to unset an option, use :unset.

:set autoindent —-> set indentation based on the previous line
:unset autoindent

•Other common options:
:set ignorecase --> ignore case during searches
:set number --> display line numbers
:set history=20 --> set the command history buffer to 20
:set showmatch --> moves the cursor briefly to a starting parenthesis or a brace when a match is typed in

Preserving vi customization

- To avoid typing in :set or :unset each time you run vi, create a .exrc file in your home directory. You can control vi options as shown in this example:

- <sample .exrc file>

```
set autoindent
set ignorecase
set showmode
set history=20
set nonumber
```

Quit vi

- To finally quit vi after saving the final version of the poem, we type “:q” as illustrated below:

I remember walking in the rain,
On a cold and dark September,
Brown Autumn leaves were falling softly to the ground,
Just like the dreams of a life as they slip away.
~
:q

- To quit without saving the file and without a prompt, we use “:q!”.

DIY: Script for an Editing Session with vi

•open vi

$ vi

•issue the following command

:r! ls –la /usr/bin --> this is to generate relatively large text

•jump to the first line
•scroll down one page
•delete 5th character from line 60
•add new line after line 60 and input text “previous changed”
•save the text to file “tmp”
•add “WARNING: “ at the beginning of the newly added line
•move word by word to “changed”
•replace “changed” with “modified” in the newly added line
•find “ksh” and delete 5 lines after the line containing it

DIY: Script for an Editing Sessions with vi

- go to the last line
- delete characters 5 to 20 of the last line
- find “csh”
- replace all “sh” with “SH”
- copy all lines from between 50 and 60 to the end of the file
- save the file as “tmp-1”
- save all and quit (Enter: “ZZ”)
- open vi with “tmp” and “tmp-1”
- change all “cSH” in “tmp” to “csh”
- go to the next file (“tmp-1”) and change all instances of “bin” to “BIN”
- save files
- change mode of “tmp” so it is not writable:

```
^Z                      --> stop the vi process
$ chmod a-w tmp                 --> change the mode
$ fg                    --> restore the vi process
```

- edit “tmp” with vi
- change all “sys” to “SYS” and save the text to “tmp”
- exit vi

Editing Files with emacs

Editing Files with Emacs

- Emacs (Editor MACroS) had its start in Artificial Intelligence community, but evolved into one of the most popular – and powerful – editors in Unix. Richard Stallman and Guy Steele wrote the first version and originated the Free Software Foundation movement.

- Emacs is not as “standard” as vi, but many Unix systems will have it.

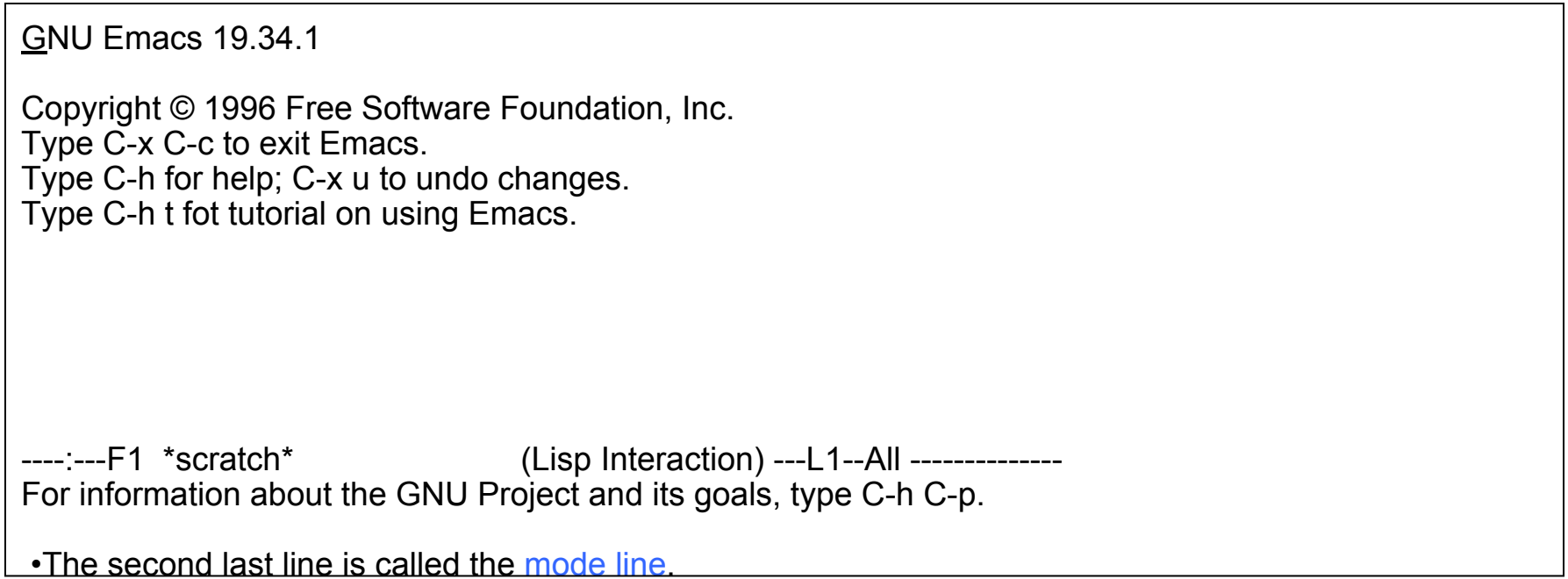
- There are also GUI versions of Emacs (e.g., Xemacs).

- To start Emacs type the following:

- $ emacs

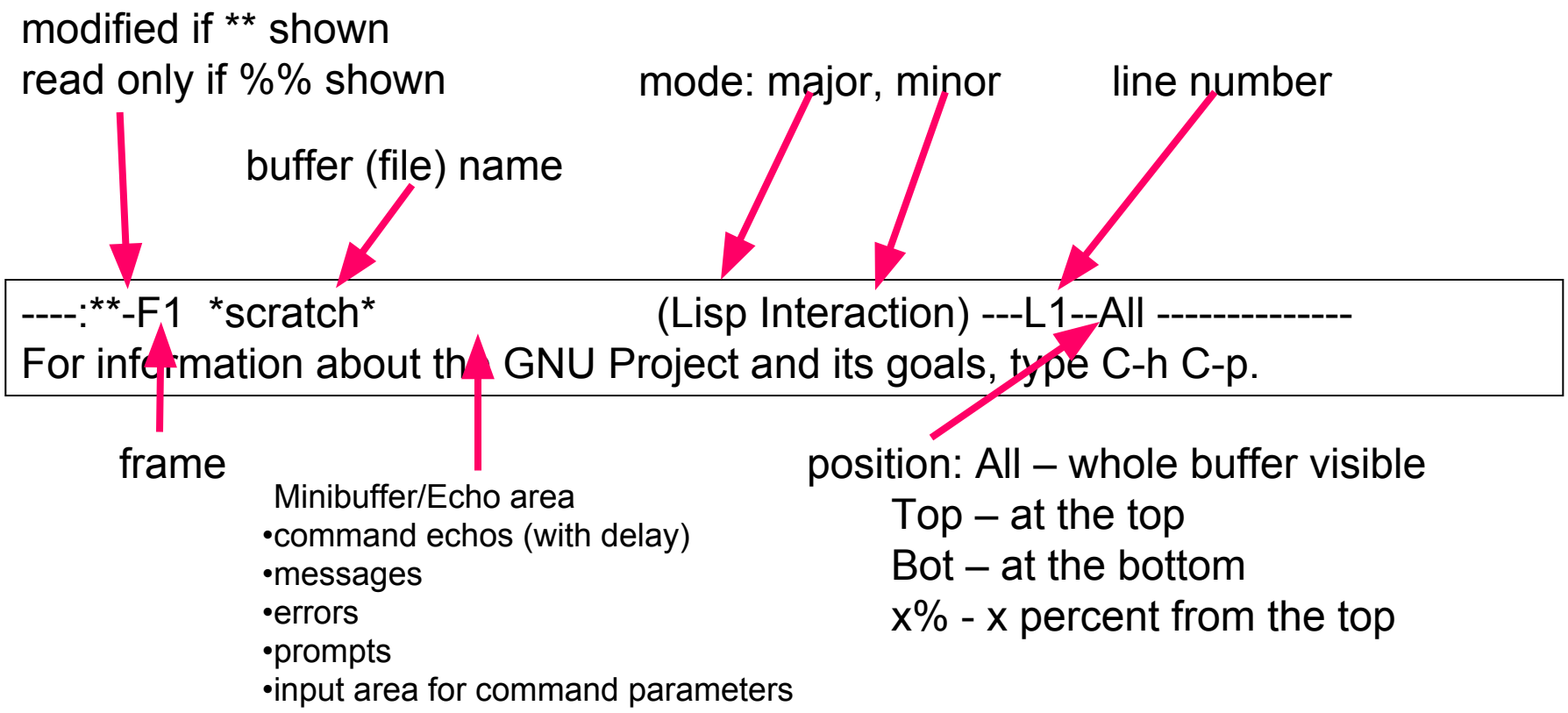
Emacs: Starting Page

You will get a screen similar to the following:



- The second last line is called the mode line.
- The last line is the message line.

Emacs: Mode Line



Emacs: Entering Commands

- Emacs has plenty of commands, so we can touch only a small subset here. In addition, Emacs is extensible through macros written in Emacs Lisp. New commands and modes can be added.

- Good tutorial available by typing:

<Control-h> t (press Control and x together and then t)

We will use a shortcut: C-h t

- In addition to control sequences, Emacs uses also meta sequences. They use either a Meta Key together with another key or Escape Key followed by another key; for example:

<Meta-v> and ESC v refer to a meta sequence M-v (move page back in this case).

On PC-based Unix systems, Alt Key is usually mapped as Meta Key.

Emacs: Getting out of Trouble

Command	Command sequence
Run tutorial	C-h t
Show help options	C-h C-h
Apropos	C-h a
Function description	C-h f <function name>
Describe key	C-h k <key sequence>
Access the Info system	C-h i
Cancel last command	C-g
Redraw screen	C-l (small letter L)
Exit Emacs	C-x C-c

Emacs: Windows and Buffers

- Emacs uses multiple buffers for both editing files and management. If you open a file, it gets its own buffer, but there are buffers for help, tutorial, messages, minibuffer (the bottom line), scrapbook, etc.

Command	Key sequence
Open file in a buffer	C-x C-f <file name>
Save buffer (file)	C-x C-s
List buffers	C-x C-b
Switch to another buffer	C-x b
Split screen vertically	C-x 2
Split screen horizontally	C-x 3
Delete all but one window	C-x 1
Switch to another window	C-x o

Emacs: Moving Around a Buffer

Command	Key sequence
Up one line	C-p (also Arrow Up)
Down one line	C-n (also Arrow Down)
Right one character	C-f (also Arrow Right)
Left one character	C-b (also Arrow Left)
Start of the line	C-a
End of the line	C-e
Forward one word	M-f
Back one word	M-b
Down one screen	C-v
Up one screen	M-v
Start of the buffer	M-<
End of the buffer	M->

Emacs: Deleting, Pasting and Undoing

Command	Key sequence
Delete character at the cursor	C-d or Delete Key
Delete character before cursor	Backspace Key
Delete word after the cursor	M-d
Delete word before the cursor	M-Backspace Key
Delete the rest of the line	C-k
Delete whole sentence	M-k
Insert (yank) last kill buffer	C-y
Retrieve previous kill (scroll the list)	M-y
Append next kill to the kill buffer	M-C-w (Meta and Control and w)
Undo	C-x u

Emacs: Searching and Replacing

- Emacs provides incremental search; i.e., the matches are found as you type the string to find.

Command	Key sequence
Search forward	C-s
Search backward	C-r
Repeat the last search forward	C-s (subsequent)
Repeat the last search backward	C-r (subsequent)
Cancel search	C-g
Interactive replace	M-x query-replace
Global replace	M-x replace-string

- M-x is a prefix for Emacs functions. In query-replace and replace-string Emacs will interactively ask for the string to replace and its new value.
- You can also use query-replace-regexp and replace-regexp to replace text using regular expressions.
- query-* functions are interactive; replace-* are global.
- Use C-h a or C-h f to get help on available functions.
- Use Tab Key in mini-buffer to view available options interactively. It's useful to look for Emacs functions as well as scanning files, directories, help options, etc.

DIY: Script for an Editing Session with emacs

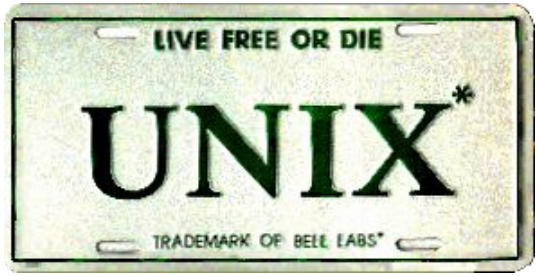
- create a large file and edit it with emacs:

$ ls –la /usr/bin > tmp-3
$ emacs tmp-3

- jump to the end of the buffer
- delete the last line
- jump to the beginning of the buffer
- scroll down one page
- delete 5th character from line 60
- go to the end of the line 60
- add new line after line 60 and input text “previous changed”
- save the text to file “tmp-4”
- jump to the beginning of the new line and add “WARNING: “ at its beginning
- move word by word to “changed”
- replace “changed” with “modified” in the newly added line
- find “ksh” and kill 5 lines after the line containing it

DIY: Script for an Editing Session with emacs

- go to the last line and yank the killed buffer
- go to line 50 and delete all characters after character 10
- find “csh”
- replace all “sh” with “SH”
- save the file as “tmp-3”
- open “tmp-4” in a new buffer
- select the tmp-3 buffer
- change all “cSH” in “tmp” to “csh”
- split window into two windows
- go to the other window and load buffer tmp-4 in there
- make the window with the buffer tmp-4 the only window
- split the window vertically, go to another window and load tmp-3
- change all instances of “bin” to “BIN” in tmp-3
- save both files
- find out what is the key combination for suspending emacs session
- change mode of “tmp-3” so it is not writable (use ^Z as with vi)
- change all “sys” to “SYS” and save the text to “tmp-3”
- get help on “replace”
- find out what’s the function assigned to ^K
- quit emacs



UNIX Utilities for Power Users

Contents

Introduces utilities for power users, listed in alphabetical order:

Introduction

We introduce about thirty useful utilities.

section	Utilities
Filtering files	egrep, fgrep, grep, uniq
Sorting files	sort
Comparing files	cmp, diff
Archiving files	tar, cpio, dump
Searching for files	find
Scheduling commands	at, cron, crontab
Programmable text processing	awk, perl
Hard and soft links	ln
Switching users	su
Checking for mail	biff
Transforming files	compress, crypt, gunzip, gzip, sed, tr, ul, uncompress
Looking at raw file contents	od
Mounting file systems	mount, umount
Identifying shells	whoami
Document preparation	nroff, spell, style, troff
Timing execution of commands	time

Filtering Files

- **egrep**, **fgrep**, and **grep**, which filter out, all lines that do not contain a specified pattern.

- **uniq**, which filters out duplicate adjacent lines

- Here's an example of this use of **grep**:

```
$ cat grepfile                    ---> list the file to be filtered
Well you know it's your bedtime,
So turn off the light,
Say all your prayers and then,
Oh you sleepy young heads dream of wonderful things,
Beautiful mermaids will swim through the sea,
And you will be swimming there too.

$ grep  the  grepfile             ---> search for the word "the"
So turn off the light,
Say all your prayers and then,
Beautiful mermaids will swim through the sea,
And you will be swimming there too.
```

Searching for Regex: grep

- grep comes from the ed (Unix text editor) search command “global regular expression print” or g/re/p
- This was such a useful command that it was written as a standalone utility
- There are two other variants, egrep and fgrep that comprise the grep family
- grep is the answer to the moments where you know you want a the file that contains a specific phrase but you can’t remember it’s name

- Family differences

- grep - uses regular expressions for pattern matching
- fgrep - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file
- egrep - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family
- agrep – approximate grep; not standard

Searching for Regex: grep

```
$ grep -wn  the grepfile-->-n: line number, -w: whole words only
2:So turn off the light,
5:Beautiful mermaids will swim through the sea,
$ grep -wnv  the grepfile       --> reverse the filter.
1:Well you know it's your bedtime,
3:Say all your prayers and then,
4:Oh you sleepy young heads dream of wonderful things,
6:And you will be swimming there too.
$ grep  -w  x  *.c          ---> search all files ending in ".c".
a.c:test ( int x )
fact2.c: long factorial(x)
fact2.c: int x;
fact2.c:   if (( x==1 ) || ( x==0 ))
fact2.c:      result = x * factorial(x-1);
$ grep -wl x  *.c           ---> list names of files that contain matches.
a.c
fact2.c
$ _
```

Removing Duplicate Lines: uniq

- The uniq utility displays a file with all of its identical adjacent lines replaced by a single occurrence of the repeated line.

- Here's an example of the use of the uniq utility:

```
$ cat  animals          ---> look at the test file.
cat  snake
monkey snake
dolphin elephant
dolphin elephant
goat elephant
pig pig
pig pig
monkey pig
```

Removing Duplicate Lines: uniq

```
$ uniq  animals          ---> filter out duplicate adjacent lines.
cat  snake
monkey snake
dolphin elephant
goat elephant
pig pig
monkey pig
$ uniq  -c  animals      ---> display a count with the lines.
    1   cat snake
    1   monkey elephant
    2   dolphin elephant
    1   goat elephant
    2   pig pig
    1   monkey pig
```

Sorting Files : sort

- The **sort** utility sorts a file in ascending or descending order based on one or more soft fields.

```
$ cat sortfile                    --> list the file to be sorted.
jan  Start  chapter  3 10th
Jan  Start  chapter  1 30th
Jan Start chapter  5 23rd
Jan End chapter 3 23rd
Mar Start chapter 7 27
may End chapter 7 17th
Apr End Chapter 5 1
Feb End chapter 1 14
$ sort  sortfile                  --> sort it.
Apr End Chapter 5 1
Feb End chapter 1 14
Jan  Start  chapter  1 30th
Jan End chapter 3 23rd
Jan Start chapter  5 23rd
Mar Start chapter 7 27
jan  Start  chapter  3 10th
may End chapter 7 17th
```

Sorting Files : sort

$ sort -r softfile --> sort it in reverse order.

may End chapter 7 17th
jan Start chapter 3 10th
Mar Start chapter 7 27
Jan Start chapter 5 23rd
Jan End chapter 3 23rd
Jan Start chapter 1 30th
Feb End chapter 1 14
Apr End Chapter 5 1

Comparing Files: cmp, diff

•There are two utilities that allow you to compare the contents of two files:

•cmp, which finds the first byte that differs between two files
•diff, which displays all of the differences and similarities between two files

•Testing for sameness: cmp

•The cmp utility determines whether two files are the same.

```
$ cat lady1                          --> look at the first test file.
Lady of the night,
I hold you close to me,
And all those loving words you say are right.
$ cat lady2                          --> look at the second test file.
Lady of the night,
I hold you close to me,
And everything you say to me is right.
$ cmp lady1  lady2                        --> files differ.
lady1 lady2 differ: char 48, line 3
$ _
```

File Differences: diff

- The **diff** utility compares two files and displays a list of editing changes that would convert the first file into the second file.

```
$ diff lady1 lady2              --> compare lady1 and lady2.
3c3
< And all those loving words you say are right.
…
> And everything you say to me is right.
$ _
```

Archives: cpio, tar, dump

- There are several occasions on which you'll want to save some files to a secondary storage medium such as a disk or tape:
  - for daily, weekly, or monthly backups
  - for transport between non-networked UNIX sites
  - for posterity

- cpio, which allows you to save directory structures onto a single backup volume.
  - It's handy for saving small quantities of data, but the single-volume restriction makes it useless for large volume.

- tar, which allows you to save directory structures onto a single backup volume.
  - It's specially designed to save files onto tape, so it always archives files at the end of the storage medium.

- dump, which allows you to save a file system onto multiple backup volumes.
  - It's specially designed for doing total and incremental backups, but it's tricky to restore individual files.

Copying Files : cpio

- The cpio utility allows you to create and access special cpio-format files.
- Unfortunately, the cpio utility is unable to write special-format files to multiple volumes, so the entire backup file must be able to reside on a single storage medium.
- If the backup is too large for a single storage medium: use the **dump** utility instead.

```
$ ls  *.c   | cpio -ov  > backup              --> save in “backup”.
main1.c
main2.c
palindrome.c
reverse.c
3 blocks
$ ls  -l  backup                          --> examine “backup”.
-rw-r--r--   1  umesh  1536  Jan   9  18:34  backup
$ rm *.c                                  --> remove the original files.
$ cpio  -it  < backup                           --> restore the files.
main1.c
main2.c
palindrome.c
reverse.c
3 blocks
```

Tape Archiving: tar

- The tar utility was designed specifically for maintaining an archive of files on a magnetic tape.
- Nevertheless, tar is also used commonly when archiving into an archive file rather than a tape. However, some suggest that you should use the cpio utility instead.

- When you add a file to an archive file using tar, the file is always placed at the end of the archive file, since you cannot modify the middle of a file that is stored on tape.

tar {ctxfvuz} {archiveName} {fileOrDirectoryName}*

  - c: creates a tar-format file
  - f: use the next file name as the output file; if neither f nor – as the archive name is used, the output goes to /dev/rmt/0m
  - v: encourages verbose output
  - t: generates a table of contents
  - x: extract
  - z: compress with gzip (more later) – in some implementations!

Tape Archiving: tar

```
$ tar  -cvf  tarfile  .                    --> archive the current directory.
a  ./main1.c   1  blocks
a  ./main2.c   1  blocks
…etc.;  edited out for space considerations.
a  ./main2  48 blocks.
a  ./tmp/b.c  1 blocks.
A  ./tmp/a.c  1 blocks.
$ ls -l tarfile                  --> look at the archive file “tarfile”.
-rw-r--r--  1  umesh   65536   Jan  10  12: 44  tarfile
$ _
$ tar  -tvf  tarfile                      --> look at the table of contents.
rwxr-xr-x    496/62       0  Jan  10 12:44  1998  ./
rw-r--r--    496/62     172  Jan  10 12:41  1998 ./main1.c
rw-r--r--   496/62     198   Jan    9  18:36  1998 ./main2.c
…
rw-r--r--   496/62        9  Jan  10  12:42  1998 ./tmp/a.c
$ _
$ tar  -vxf  tarfile  ./tmp               --> extract archived “tmp” files.
x  ./tmp/b.c,  9 bytes,  1 tape blocks
x  ./tmp/a.c, 9 bytes, 1 tape blocks
$ ls tmp                                  --> confirm restoration.
a.c    b.c
$ _
```

Incremental Backups: dump and restore

- Here's a system administrator's typical backup strategy:
  - Perform a weekly total-file system backups.
  - Perform a daily incremental backup, storing only those files that were changed since the last incremental backup.

- This kind of backup strategy is supported nicely by the dump and restore utilities.

- Example:

- Performs a level-zero dump of the filesystem on /dev/da0 to the tape drive /dev/rmt0 with verification:

- $ dump 0 fv /dev/rmt0 /dev/da0
- --> a level-zero dump will always dump all files,
- --> v option to verify each volume of media after it is written.

Incremental Backups: restore

- A level 0, full backup, guarantees the entire file system is copied. A level number above 0, incremental backup, tells dump to copy all files new or modified since the last dump of the same or lower level.

- The restore utility allows you to restore files from a dump backup.

- We can use restore to extract a couple of previously saved files from the dump device "/dev/rmt0":

- -x option : to restore only the specified filename(s) from dump file.

- $ restore -x f /dev/rmt0 wine.c hacking.c

Finding Files: find

•find startingDir searchOptions commandToPerform

•Here are some examples of find in action:

```
$ find  /code  -name  '*.c'  -print          --> print C source files
                                        --> in the current directory or
                                        --> any of its subdirectories.
./proj/fall.17/play.c
./proj/fall.17/rerefee.c
./proj/fall.17/player.c
./rock/guess.c
./rock/play.c
./rock/player.c
./rock/referee.c
$ find /code -mtime -14 –ls          --> list modified files during the last 14 days
…
$ find . -name '*.txt' - print       --> find all text files in the current directory
…
```

Scheduling Commands: cron, crontab, at

- There are two utilities that allow you to schedule commands to be executed at a later point in time:

1. **crontab**, which allows you to create a scheduling table that describes a series of jobs to be executed on a periodic basis

2. **at**, which allows you to schedule jobs to be executed on a one-time basis

- Periodic Execution: **cron** and **crontab**

- The **crontab** utility allows you to schedule a series of jobs to be executed on a periodic basis.

Scheduling Commands: crontab

- To use **crontab**, you must prepare an input file that contains lines of the format:

minute hour day month weekday command

- where the values of each field are as follows:

Field	Valid Value
minute	0-59
hour	0-23
day	1-31
month	1-12
weekdays	1-7 (1=Mon, 2=Tue, 3=Wed, 4=Thu, 5=Fri, 6=Sat, 7=Sun)
command	any UNIX command

Crontab file

- Whenever the current time matches a line's description, the associated command is executed by the shell specified in the $SHELL environment variable.

- If any of the first five fields contain an "*" instead of a number, the field always matches.

- Any characters following a "%" are copied into a temporary file and used as the command's standard input.

- a sample crontab file that we created in our home directory and called "crontab.cron":

```
$ crontab -e                    --> create a default crontab file.
$ crontab -l                    --> list the crontab file.
 0    8    *    *    1   echo Happy Monday Morning
 *    *    *    *    *   echo One Minute Passed > /dev/tty1
 30   14   1    *    1   mail  users % Jan Meeting At 3pm
```

Periodic Scheduling: cron

- a single process called "cron" that is responsible for executing the commands in registered crontab files in a timely fashion.

- It is started when the UNIX system is booted and does not stop until the UNIX system is shut down.

- To register a custom crontab file, use the crontab utility with the name of the crontab file as the single argument:

```
$ vi crontab.cron                    --> create the crontab file.
$ crontab   crontab.cron                    --> register the crontab file.
$ _
```

- To unregister a crontab file, use the "-r" option:

```
$ crontab   -r                       --> unregister my crontab file.
$ _
```

One-Time Execution : at

•The **at** utility allows you to schedule one-time commands and/or scripts.

```
$ cat at.csh                --> look at the script to be scheduled.
#! /bin/csh
echo  at  done > /dev/tty1          --> echo output to terminal.
$ at  now  +2  minutes  at.csh    --> schedule script to
                                --> execute in two minutes.
Job   2519  at  Sat  Jan  10  17:30:00   1998
$ at  -l                --> look at the at schedule atq on some systems).
     2519  a     Sat  Jan  10  17:30:00  1998
$ _
$ at 17:35  at.csh          --> schedule the script again.
Job  2520  at  Sat  Jan  10  17:35:00  1998
$ at  -r  2520          --> deschedule (atrm on some systems)
$ _
```

Hard Links: ln

- The **ln** utility allows you to create both hard links and symbolic (soft) links between files.

- In the following example, we add a new label “hold” to the file referenced by the existing label “hold.3”.

```
$ ls  -l                    --> look at the current contents of the directory.
total 3
-rw-r--r--   1   umesh   123  Jan  12  17:32  hold.1
-rw-r--r--   1   umesh    89  Jan  12  17:34  hold.2
-rw-r--r--   1   umesh    91  Jan  12  17:34  hold.3
$ ln hold.3  hold           --> create a new hard link.
$ ls  -l                    --> look at the new contents of the directory.
total  4
-rw-r--r--  2  umesh    91  Jan  12  17:34  hold
-rw-r--r--  1  umesh   124  Jan  12  17:32  hold.1
-rw-r--r--  1  umesh    89  Jan  12  17:34  hold.2
-rw-r--r--  2  umesh    91  Jan  12  17:34  hold.3
```

Hard Links: ln

```
$  rm  hold          --> remove one of the links.
$  ls  -l           --> look at the updated contents of the directory.
total  3
-rw-r--r--   1   umesh   123  Jan  12  17:32   hold.1
-rw-r--r--   1   umesh    89  Jan  12  17:34   hold.2
-rw-r--r--   1   umesh    91  Jan  12  17:34   hold.3
$ _
```

- Note that the hard-link count field was incremented from one to two when the hard link was added and then went back to one again when the hard link was deleted:

Soft Links: ln

To create a soft link, use ln -s

```
$ ls -l
total 48
-rw-r--r--          1 user  users   84 26 Sep 17:08 tmp
-rw-r--r--          1 user  users   24 26 Sep 19:41 tmp1
$ ln tmp tmp2
$ ls -l
total 56
-rw-r--r--          2 user  users 84 26 Sep 17:08 tmp
-rw-r--r--          1 user  users 24 26 Sep 19:41 tmp1
-rw-r--r--          2 user  users 84 26 Sep 17:08 tmp2
$ ln -s tmp tmp3
$ ls -l
total 64
-rw-r--r--          2 user  users 84 26 Sep 17:08 tmp
-rw-r--r--          1 user  users 24 26 Sep 19:41 tmp1
-rw-r--r--          2 user  users 84 26 Sep 17:08 tmp2
lrwxr-xr-x    1 user  users 3   27 Sep 09:08 tmp3 -> tmp
$ _
```

Substituting a User: su

- A lot of people think that the name of the **su** utility stands for super-user, but it doesn't. Instead, it stands for substitute user.
- This utility allows you to create a subshell owned by another user.

- Here's an example of the use of su:

```
$ whoami          ---> find out my current user ID.
umesh
$ su          ---> substitute user.
Password:         ---> enter super-user password here.
$ whoami          --> confirm my current user ID has changed.
root
$ … perform super-user tasks here.
$ ^D              --> terminate the child shell.
```

- You can also use sudo for "one shot" command execution

Transforming Files: compress, uncompress

- There are several utilities that perform a transformation on the contents of a file, including the following utilities:

- **compress** and **uncompress**, which convert a file into a space-efficient intermediate format and then back again.

- This utility is useful for saving disk space.

- compress is useful to reducing the amount of disk space that you take up and packing more files into an archive file.

Transforming Files: compress, uncompress

•Here's an example of its use:

```
$ ls -l palindrome.c reverse.c          --> examine the original files.
-rw-r--r--   1   umesh     224   Jan  10  13:05  palindrome.c
-rw-r--r--   1   umesh     266   Jan  10  13:05  reverse.c
$ compress  -v  palindrome.c  reverse.c  --> compress them.
palindrome.c : Compression : 20.08%  -- replaced with palindrome.c.Z
reverse.c : Compression : 22.93%   -- replaced with reverse.c.Z
$ ls  -l  palindrome.c.Z  reverse.c.Z
-rw-r--r--   1  umesh     179  Jan  10  13:05  palindrome.c.Z
-rw-r--r--   1  umesh     205  Jan  10  13:05  reverse.c.Z
$ uncompress  -v  *.Z                   --> restore the original files.
palindrome.c.Z:    -- replaced with palindrome.c
reverse.c.Z:    -- replaced with reverse.c
$ ls  -l  palindrome.c  reverse.c              --> confirm the restoration.
-rw-r--r--   1  umesh   224  Jan  10  13:05  palindrome.c
-rw-r--r--   1  umesh   266  Jan  10  13:05  reverse.c
$ _
```

Compressing Files: gzip, gunzip

- compress and uncompress use a patented algorithm, so they may be removed from the UNIX system to avoid paying royalties

- instead, freelly distributed GNU utilities gzip and gunzip are commonly used

gzip –cv {fileName}+
gunzip –cv {fileName}+

- gzip replaces a file with a compressed version and adds a .gz suffix to its name. The –c option will redirect the output to the standard output, so the original file is not overwritten. The –v option displays compression statistics.

- gunzip upacks files compressed with gzip or with compress. The options have similar meanings as in gzip.

Compressing Files: gzip, gunzip

•Here's an example of its use:

```
$ ls -l palindrome.c reverse.c          --> examine the original files.
-rw-r--r--   1  umesh    224  Jan  10  13:05  palindrome.c
-rw-r--r--   1  umesh    266  Jan  10  13:05  reverse.c
$ gzip  -v  palindrome.c  reverse.c  --> compress them.
palindrome.c:              20.08%  -- replaced with palindrome.c.gz
reverse.c:          22.93%   -- replaced with reverse.c.gz
$ ls  -l  palindrome.c.Z  reverse.c.Z
-rw-r--r--   1  umesh    179  Jan  10  13:05  palindrome.c.gz
-rw-r--r--   1  umesh    205  Jan  10  13:05  reverse.c.gz
$ gunzip  -v  *.gz                  --> restore the original files.
palindrome.c.gz:          20.08% -- replaced with palindrome.c
reverse.c.gz:             22.93% -- replaced with reverse.c
$ ls  -l  palindrome.c  reverse.c              --> confirm the restoration.
-rw-r--r--   1  umesh   224  Jan  10  13:05  palindrome.c
-rw-r--r--   1  umesh   266  Jan  10  13:05  reverse.c
$ _
```

Compressed Archives

- Very often, tar and gzip are used together to create files that are compressed archives. Such compressed packages can be used for content distribution. For example, in some Unix systems, to tar all .cc and .h files into a tar file named foo.tgz use:

$ tar cvzf foo.tgz *.cc *.h

- This creates (c) a compressed (z) tar file named foo.tgz (f) and shows the files being stored into the tar file (v).

- The .tgz suffix is a convention for gzipped tar files, it's useful to use the convention since you'll know to use z to restore/extract.

- It's often more useful to tar a directory (which tars all files and subdirectories recursively unless you specify otherwise). The nice part about tarring a directory is that it is untarred as a directory rather than as individual files.

$ tar cvzf foo.tgz cps100

- will tar the directory cps100 (and its files/subdirectories) into a tar file named foo.tgz.

Common Practice: mixing tar and compress or gzip

- To see a tar file's table of contents use:

$ tar tzf foo.tgz

- To extract the contents of a tar file use:

$ tar xvzf foo.tgz

- This untars/extracts (x) into the directory from which the command is invoked, and prints the files being extracted (v).

- If you want to untar into a specified directory, change into that directory and then use tar. For example, to untar into a directory named newdir:

```
$ mkdir newdir
$ cd newdir
$ tar xvzf ../foo.tgz
```

- You can extract only one (or several) files if you know the name of the file. For example, to

Transforming Files: tr

- tr copies the standard input to the standard output with substitution or deletion of selected characters
- input characters from string1 are replaced with the corresponding characters in string2 (the last character in string2 is used for padding if necessary)

tr –dsc string1 string2

- with the –c option, string1 is substituted with its complement; i.e., all characters that are not in the original string1

- the –d option results in deletion from the input string of all characters specified in string1

- the –s option condenses all repeated characters in the input string to a single character

Transforming Files with tr: Examples

```
$ echo "a   lot     of      space" | tr alo xyz          --> transfer a to x, l to y
                                                        --> and o to z
x   yzt     zf      spxce


$ echo "a   lot     of      space" | tr [:lower:] [:upper:]      --> transfer all lower case charcters to upper.
A  LOT   OF      SPACE

$ echo "a   lot     of      space" | tr " " .       --> transfer all spaces into dots
a...lot.....of......space
$ echo "a   lot     of      space" | tr " " . | tr -s "."         --> transfer all spaces into dots and
                                                                        --> compact the dots

a.lot.of.space
$ _
```

Transforming Files: ul

- Some Unix utilities format characters, so they look good on terminals. For example, man is one of them. The special characters may not print correctly, so there is where the ul utility is useful.

ul –tterminal {filename}*

- ul transfers all underlined characters into characters that are emphasized properly for the specified terminal

- For example, try:

```
$ man who
Q
$ man who | ul
$ man who | ul -tdumb
```

Inspecting Raw File Content: od

- The od utility allows to inspect the content of a file in a variety of ways.

od [–bcdosx] fileName [offset[.][b]]

Use:

-b Interpret bytes in octal (hexadecimal).
-c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
-d Interpret 16-bit words in decimal.
-o Interpret 16-bit words in octal.
-s Interpret 16-bit words in signed decimal.
-x Interpret 16-bit words in hexadecimal.

- offset indicates the starting point in the file; if preceded with “x”, it’s a hex; if followed with “.” then it’s a decimal

- using b indicates the number of blocks rather than octal numbers

- xd is a similar utility to dump in hexadecimal by default

Inspecting Raw File Content: od

```
$ echo "1234567890abcdefghijklmnopqrstuvwxyz" | od -b
0000000 061 062 063 064 065 066 067 070 071 060 141 142 143 144 145 146
0000020 147 150 151 152 153 154 155 156 157 160 161 162 163 164 165 166
0000040 167 170 171 172 012
0000045
$ echo "1234567890abcdefghijklmnopqrstuvwxyz" | od -c
0000000   1   2   3   4   5   6   7   8   9   0   a   b   c   d   e   f
0000020   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v
0000040   w   x   y   z  \n
0000045
$ echo "1234567890abcdefghijklmnopqrstuvwxyz" | od -d
0000000 12594 13108 13622 14136 14640 24930 25444 25958
0000020 26472 26986 27500 28014 28528 29042 29556 30070
0000040 30584 31098 02560
0000045
$ echo "1234567890abcdefghijklmnopqrstuvwxyz" | od -o
0000000 030462 031464 032466 033470 034460 060542 061544 062546
0000020 063550 064552 065554 066556 067560 070562 071564 072566
0000040 073570 074572 005000
0000045
$ echo "1234567890abcdefghijklmnopqrstuvwxyz" | od -x
0000000 3132 3334 3536 3738 3930 6162 6364 6566
```

Mounting File System: mount, umount

- A super-user may extend the file system by using the **mount** utility and reverse the effect of the **mount** utility by using the **umount** utility.

Utility : **mount** -o*options* [deviceName directory]
umount deviceName

- **mount** is a utility that allows you to “splice” a device’s file system into the root hierarchy.

- When used without any arguments, **mount** displays a list of the currently mounted devices.

- The **umount** utility unmounts a previously mounted file system.

Mounting File System: mount

```
$ /sbin/mount                          --> list the currently mounted devices.
/dev/dsk1  on  / (rw)
$ ls   /usr                            --> "/usr" is currently empty.
$ /sbin/mount -r /dev/dsk2  /usr       --> mount the "/dev/dsk2" device
                                       --> as a read-only /usr volume

$ /sbin/mount
/dev/dsk1   on  /  (rw)
/dev/dsk2   on  /usr  (rw)
$ ls   /usr                    --> list the contents of the mounted device.
bin/        etc/        include/    lost+found/    src/     ucb/
demo/       games/      lib/        pub/           sys/     ucblib/
dict/       hosts/      local/      spool/         tmp/
$ _
```

Un-Mounting a File System: umount

- To unmount a device, use the **umount** utility.

- I unmounted the "/dev/dsk2" device and then listed the "/usr" directory.

- The files were no longer accessible.

```
$ /sbin/umount   /dev/dsk2        --> unmount the device.
$ /sbin/mount                     --> list the currently mounted devices.
/dev/dsk1  on  /   (rw)
$ ls  /usr                        --> note that "/usr" is empty again.
$ _
```

Identifying Shells: whoami

- You can use the **whoami** utility to display the owner of a shell:

  Utility: **whoami**

  Displays the owner of a shell.

- For example, when user umesh executes whoami at his terminal, he sees this:

```
$ whoami
umesh
$ _
```

Timing Execution: time

- The time utility is useful if you need to measure the time it takes to execute a command, another utility or a program.

- For example:

```
$ time find /opt/local/bin -name bash -print
/opt/local/bin/bash

real     0m0.304s          ---> total time elapsed
user     0m0.010s          ---> time consumed by system overhead
sys      0m0.080s          ---> time used to execute “find” to the
                                ---> standard error stream
$ _
```