Chapter – 2 Getting Started with Unix

Brief history of UNIX

Unix is a multitasking, multi-user computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna.

The Unix operating system was first developed in assembly language, but by 1973 had been almost entirely recoded in C, greatly facilitating its further development and porting to other hardware.

Today's Unix system evolution is split into various branches, developed over time by AT&T as well as various commercial vendors, universities (such as University of California, Berkeley's BSD), and non-profit organizations.

Unix is a free open source UNIX OS for PCs that was originally developed in 1991 by Linus Torvalds, a Finnish undergraduate student.

The open source nature of Unix means that the source code for the Unix kernel is freely available so that anyone can add features and correct deficiencies. This approach has been very successful and what started as one person's project has now turned into a collaboration of hundreds of volunteer developers from around the globe.

A distribution comprises a prepackaged kernel, system utilities, GUI interfaces and application programs.

Architecture of Unix operating system

The unix system is mainly composed of three different parts: the kernel, the file system and the shell.

The kernel is the heart of the operating system – a collection of programs mostly written in C that directly communicate with hardware.

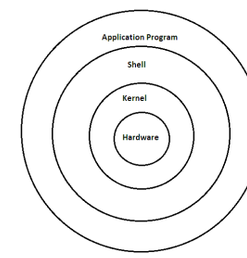
There is only one kernel for any system. It's part of the unix system that is loaded into the memory when the system is booted.

It manages the system resource, allocate time between users and processes, decide process priorities and perform other task.
The shell is actually the interface between the user and kernel. The shell is also called command interpreter. It listens to terminal and translates your request into actions on the part of the kernel.

No command can be executed unless it obtains the clearance of the shell.

The shell when analyzing the command and its associated arguments, often modifies and simplifies its input before it forward to kernel.

The Application programs are utility and collections of programs that user use for their different purpose on top of all layer. The file systems are the organization structure of data.



One can imagine the unix system as a series of circles, most inner circle represents the computer hardware n top of that kernel resides; it means it's the kernel who deals with system resources. the third circle from inner represents the programming shell and the last one representing application program. Shell interprets the application program command and communicates to the kernel.

Unix Features

Unix has number of features. Here listed below are some important features of unix operating system.

- **Multi-user**
  - unix is a multi user operating system means more than one user can shared resources and work with unix operating system.
  - The resources are actually shared between all users like RAM, CPU or the hard disk.
  - The compute breaks up a unit of time into several segments and each user is allocated a segment s so at any point of time the machine will be doing job of a single user.
  - Unix handles multiple users by use of time sharing technique
- **Multitasking System**
  - In Microsoft windows is considered to be a multitasking system where you can work with word, excel, media player concurrently without quitting any application.
  - In unix a single user can also run multiple tasks concurrently. You can switch jobs between background and foreground, suspend or even kill them. This is done by running one job normally and the others in the background.
- **Programming Facility**
  - Unix was designed for a programmer.
  - Unix shell programming language has all the necessary things like control structure, loops and variables as a programming language.
  - Though the language is somewhat difficult to learn. The shell can be combined easily with command and other programs.
- **System calls & libraries**

- Unix is written in C. though there are a couple of hundred specialized functions, they all do system calls.
- These calls are built into kernel, and all library functions are written using them.
- A user who wishes to write to a file, uses write() system call without going into the details of write() operation.

- **Software Available**
  - Thousands of application packages are available for the UNIX/LINUX system. In addition to commercial packages, many programs are written and made available in the public domain.
  - Most open source, freeware packages used in the scientific world are developed and maintained on Unix platform.
- **Network Capabilities**
  - Today UNIX workstation comes with TCP/IP and Ethernet connection.
  - It is simple to logon to a remote machine, send mail to user on another machine or transfer files and get network environment.
- **Windowing System**
  - Unix had to come up with its own GUI. The X window system provides an environment that allows you to have multiple windows where you can run applications on each window individually.
  - It replaces the command line with a screenful of objects in the form of menus, icons, buttons, and dialog boxes.
  - X window supports networked applications so that an application may run on one machine and have display on another.
  - Many database frond end software use X window system to run forms and application.

Unix Shell $ 'sh'

A Unix shell is a command-line interpreter or shell that provides a traditional user interface for the Unix operating system and for Unix-like systems.
Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands.

Types of Shell

A number of different shells have been developed for Unix - like operating systems. They share many similarities, but there are also some differences with regard to commands, syntax and functions that are important mainly for advanced users.

The following are the different types of shell in UNIX

Bourne Shell(sh)

- This is the original shell of UNIX which comes with every UNIX system and may be that is the reason that it is so popular.
- It was written by Stephen Bourne at Bell Labs in 1974. It is a simple shell with a small size and few features. It is good for good features for I/O control but not well suited for interactive users.
- Its default prompt is $.

C shell(csh)

- It has a syntax that resembles that of the highly popular C programming language (also developed at Bell Labs), and thus it is sometimes preferred by programmers.
- It was created in 1978 by Bill Joy (who also wrote the vi text editor and later co-founded Sun Microsystems) at the University of California at Berkeley (UCB).Its I/O is more awkward than Bourne shell but it is nicer for interactive use.
- It provides additional features such as job control and history.
- Its default prompt is %.

Kornshell(ksh)

- It is a superset of sh developed by David Korn at Bell Labs in 1983. It contains many features of the C shell as well, including a command histories.
- It also features built-in arithmetic evaluation and advanced scripting capabilities similar to those found in powerful programming languages such as awk, sed and perl.

Bourne-again shell(bash)

- It is the default shell on Unix operating system. It also runs on nearly every other Unix-like operating system as well, and versions are also available for other operating systems including the Microsoft Windows systems.
- It is a superset of sh (i.e., commands that work in sh also work in bash, but the reverse is not always true), and it has many more commands than sh, making it a powerful tool for advanced users.

Unix file system navigation

Unix divide the file system in mainly three category.

1) Ordinary file – contains data only
2) Directory file – contains the other files and directory
3) Device files – represents all hardware and device files.

1) Ordinary file:

This is the traditional definition of a file. It consists of a data on some permanent media. You can put anything inside this kind of file. This includes all data, programs, objects, and executable code, all unix commands etc.
The most common type of ordinary file is text file. This is just regular file containing printable characters.

2) Directory file

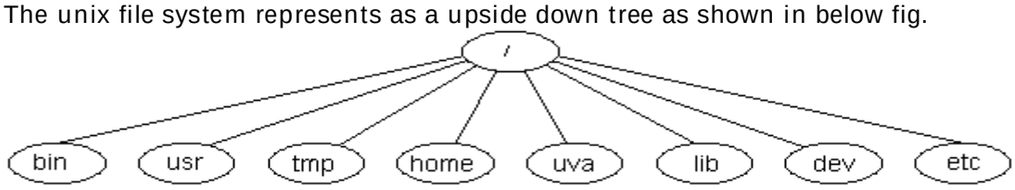
A directory contains no external data, but keep some files of sub-directory that itcontains. The unix file system is organized with a number of such directories and sub-directories. You can also create them when you need it.

3) Device file

The definition of physical device stores in device files. This definition includes printers, taps, floppy drives, cd-rom, hard disk and terminals.
When you issues a command to print a file , you are really directing the file's output to the file associated with printers.

The unix file system

The unix file system represents as a upside down tree as shown in below fig.

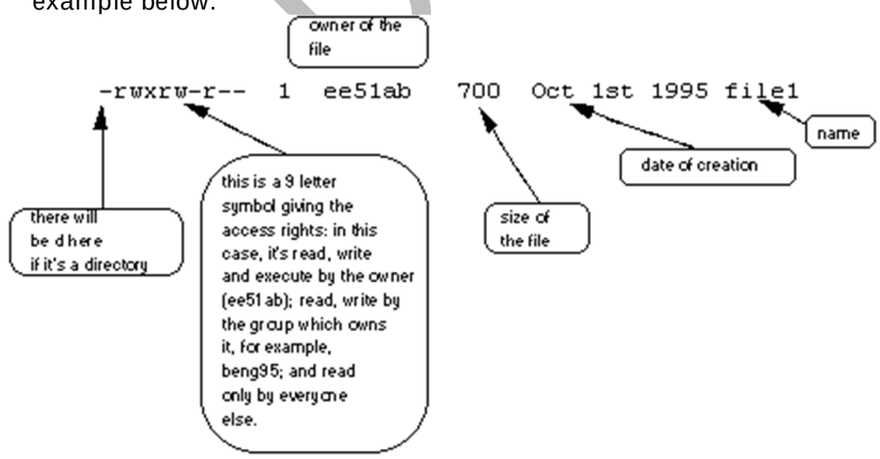


The file system begins with a directory called root. The root directory represents as slash(/). Branching from the root are several other directories called bin, lib, usr, etc, tmp and dev.
The root directory also contains a file called unix which is unix kernel itself.

Directory	Contains
bin	Binary executable file
lib	Library Functions
dev	Device related files
etc	Binary executable files required from system administration
tmp	Temporary file created by unix users
usr	Home directory for all unix users

Permissions /Rights

In your **unixstuff** directory, type
% ls -l (l for long listing!)
You will see that you now get lots of details about the contents of your directory, similar to the example below.



Each file (and directory) has associated access rights, which may be found by typing **`ls -l`**. Also, **`ls -lg`** gives additional information as to which group owns the file (beng95 in the following example):

```
-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1
```

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the left hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string.
The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

- The left group of 3 gives the file permissions for the user that owns the file (or directory) (ee51ab in the above example);
- the middle group gives the permissions for the group of people to whom the file (or directory) belongs (eebeng95 in the above example);

- the rightmost group gives the permissions for all others.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access rights on files.

- r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file
- w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file
- x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate

Access rights on directories.

- r allows users to list files in the directory;
- w means that users may delete files from the directory or move files into it;
- x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.
  So, in order to read a file, you must have execute permission on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.

Some examples

-rwxrwxrwx	a file that everyone can read, write and execute (and delete).
-rw-------	a file that only the owner can read and write - no-one else can read or write and no-one has execution rights (e.g. your mailbox file).

Changing Access Rights

chmod (changing a file mode)

Only the owner of a file can use chmod to change the permissions of a file. The options of chmod are as follows

Symbol	Meaning
u	User
g	Group
o	Other
a	All
r	Read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

For example, to remove read write and execute permissions on the file **biglist** for the group and others, type

```
% chmod go-rwxbiglist
```

This will leave the other permissions unaffected.
To give read and write permissions on the file **biglist** to all,

```
% chmoda+rwbiglist
```

Unix Shell Commands

Telnet :

Every unix vendor offers the Telnet and rlogin utilities in its TCP/IP package to connect to a remote unix system. telnet belongs to DARPA command set, while rlogin is a member of the Berkeley set of r-utilities. You must have an account on the remore machine to use telnet with the machine's IP address as argument.

```
$ telnet 192.168.2.1
        Login
```

You now have to enter your login name at this prompt,and then the password to gain access to the remote machine. After you have finished, you can press <Ctrl-d>, or type exit to logout and return to your local shell.
When telnet is used without the address, the system display the telnet> prompt, from where you can use its internal commands.

```
telnet>! ls -l
```

Login Commands :

passwd

The passwd command is used to change password. passwd when invoked by user, asks for the old password, after which it demands the new password twice.
When you enter a password, the string is encrypted by the system, and the encryption is stored in the file /etc/shadow file. The password entered by user will not display on screen.

Syntax

passwd

e.g.

```
$ passwd
Old password:
New password:
Retype new password:
```

Logout

The logout command is used to complete the user session.

Syntax

Logout

e.g.

$ logout

who

This command display who is on the system. Basically user list of the system.
who displays various pieces of information about logged in users to standard output. who lists the user's name, terminal line, login time, and remote host name, if applicable, for each logged in user it reports.

Syntax

who [options] [am i]

Options

-l # writes the idle time for each displayed user. The idle time is the time since last activity on that line.
-m # writes only information about the current terminal.
-q # writes only the login names and the number of users logged on.

e.g.

```
$ who
Raval              tty10    Dec 10 23:15
Raval              tty11    Dec 11 00:20
$ who -i
Raval            tty10    Dec 10 23:15 01:24
Raval    tty11    Dec 11 00:20    .
$ who -m
Raval              tty11    Dec 11 00:20
$ who -q
RavalRaval
# users=2
$ who am i              # equivalent to who -m
Raval              tty11    Dec 11 00:20
```

Clear

clear command clears the screen and puts cursor at beginning of first line.

Syntax :

Clear

e.g.

clear

FILE/DIRECTORY RELATED COMMANDS :

ls command

ls is to unix as bir is to DOS. It gives the directory listing or list the content of current directory or specified directory.
There are lot of options with ls command.
Lets begin with the plain and simple ls command without any option.

$ ls

```
Administrator  Default        User           Public         empemplistneww      typescript
All Users      Guest          adminlist      emp1           hrlistsaleslist
Default        Vdr     desktop.ini     emp3  I        nst     test
```

It display all the content of current directory which contain file, directory as well.

Options	Description
-x	Display multi-columnar output
-F	Marks executables with * and directories with /
-r	Sorts files in reverse order

-a	Shows all files including ., .. and those beginning with a dot(show hidden files)
-R	Recursive listing of all files in sub-directories
-l	Long listing showing seven attributes of a file
-d	Forces listing of a directory
-t	Sorts files by modification time
-u	Sorts files by access time
-i	Shows inode number of a file

ls –x
when you have several files, its better to display the filenames in multiple columns. Linux does it by default but still you can produce the same output using –x option.

```
$ ls -x
Administrator  All Users      Default              Default User   Guest
Vdr    Public         adminlist      desktop.ini    emp
emp1           emp3           emplisthrlist  inst
neww   saleslist       test                  typescript
```

ls-F
-F options is used to identify files and directories and executable file from the list of files.

```
$ls -F
Administrator/        Guest/        desktop.ini*   emplistsaleslist
All Users/            Vdr/          emp    hrlist  test/
Default/              Public/       emp1           inst    typescript
Default User/         adminlist     emp3           neww
```

The ***** indicates that the file contains executable code. While the **/** referes to a directory.
ls –r
you can reverse the oreder of the presentation with –r (reverse) option. The file will then be listed with names sorted in reverse ASCII sequence.

```
$ls -r
typescript      neww  emplistemp    Public  Default User  Administrator
test    inst          emp3          desktop.ini   Vdr  Default
saleslist       hrlist          emp1          adminlist       Guest   All Users
```

ls-a
ls doesn't normally show all files in a directory. There are certain hidden files(files beginning with a dot) in every directory, especially in the login(or home) directory that normally don't show up in the listing.
The **–a** option (all) lists all the hidden files along with the other files . The directory .represents the current directory and .. signifies the parent directory.

```
$ls -a
.       All Users      Guest          adminlist      emp1           hrlist  saleslist
..              Default        Vdr    desktop.ini    emp3           inst    test
Administrator          Default User   Public                 emp    emplistneww    typescript
```

ls –R
The –R(recursive) option, an enhancement from Berkeley, list all files and sub-directories in a directory tree. Similar to DIR/S command of DOS, this traversal of the directory tree is done recursively till there are no sub-directories left.

```
$ ls –R
dir1  file1  file2

./dir1:
emp
```

in above example there are two files in current directory named file1 and file2 and there is one directory called dir1 the –R options show recursive list of that directory also.
ls –l
ls –l options provide long listing of a file with information like, size of the file, permission on the file, it's modifaction time and ownership details.

```
$ ls -l
total 3
drwxr-xr-x     2 Vdr  None        512 Dec  8 16:47 dir1
-rw-r--r--     1 Vdr  None          3 Dec  8 16:46 file1
-rw-r--r--     1 Vdr  None          3 Dec  8 16:46 file2
```

here total 3 indicates total 3 block are occupied by these files on disk.

First column indicates permissions on the file. It is devided in 10 character.
drwxr-xr-x

1st represents type of file
Which can be anything from below.

File type	Meaning
-	Ordinary file
d	Directory file
c	Character special file
b	Block special file
l	Symbolic link

2nd, 3rd, 4th –indicate r(read) w(write) e(execute) permission of user or owner
5th, 6th, 7th - indicate r(read) w(write) e(execute) permission of group.
8th,9th,10th - indicate r(read) w(write) e(execute) permission of other.

Second column indicates number of links for that file.
Third column indicates file owner name.
fourth column indicates group owner.
Fifth colum indication medication time for a file.
Last column indicates file or directory name.

The directories .and ..

Enter in unixstuff directory.

```
$ cdunixstaff
$ ls -a
```

As you can see, in the unixstuff directory (and in all other directories), there are two special directories called (.) and (..)

The current directory (.)

In UNIX, (.) means the current directory, so typing
$ cd .
NOTE: there is a space between cd and the dot
means stay where you are (the unixstuff directory).
This may not seem very useful at first, but using (.) as the name of the current directory will save a lot of typing.

The parent directory (..)

(..) means the parent of the current directory, so typing
$ cd ..
will take you one directory up the hierarchy (back to your home directory).
Note: typing cd with no argument always returns you to your home directory. This is very useful if you are lost in the file system.

~ (your home directory)

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory. So typing
$ ls ~/unixstuff
will list the contents of your unixstuff directory, no matter where you currently are in the file system.

$ ls ~
List the files and folders from your home directory

Creating files

In unix you can create file using cat command as follows:

Syntax:
cat><finename>

e.g.
you can create a file name test using following command.

$cat>test

Now after implement this command unix will allow you to enter some data in file test.
You can use <ctrl d> to save a content of a file and come out of a file.
You can also use touch command to create a file.
e.g.
$touch sample

The above command will create a file called sample.

You can create multiple files quickly using following command
$touch sample1 sample2 sample3
Above command will create 3 files name sample1, sample2 ,sample3.

Display content of a file

To display content of any file or terminal you can use cat command as follows:

syntax
cat <filename>

e.g.
$cat test

The above command will display the content of test file on the terminal.
You can get content of multiple files on the terminal using cat command as follows:
e.g.
$cat sample1 sample2 sample3

The above command will display the content of sample1, sample2 and sample3 on terminal.

Removing a file

In unix to remove a file rm command is used.

syntax
rm<filename>

rm command removes the given file or files supplied to it.

e.g.
$rm test
The above command will remove the file named test from current directory.

Remove file with confirmation

rm supports –I option which support user confirmation before delete a file.

e.g.
$rm –I test

the above command will ask for user confirmation before removing file test.
You can remove more than one file using following command

e.g.
$rm test1 test2 test3

The above command will remove file test1, test2 and test3.

Copy file

In unix you can use cp command to copy a file.
syntax
cp<soucefile><destination file>
e.g.
$cpletter.aletter.b

The above command will copy the contents of letter.a into a file letter.b. ifletter.bdoesnot exist it will be created. If it exist unix will overwrite the content of the file.

Move file

Renaming of file in DOS interpreted in unix as moving them. The mv command also has the power to rename directories.
Syntax
mv <old filename><newfilename>

e.g.
suppose we want to rename the file test to sample we should say
$mv test sample

Moving is different than copying in that the source file erase from its original location and moved at specified destination.
e.g.
$mv file1 file2 newdir

on execution of the above command file1 and file2 are no longer present at the original location but are moved to directory newdir.

man

This is help command, and will explains you about online manual pages you can also use man in conjunction with any command to learn more about that command for example.
e.g.
manls

pwd – checking your current directory

at any time you want to know your current working directory you can use pwd(present working directory) command.

$pwd
/usr/vdr

What you see above is a pathname which is simply a sequence of directory names separated by slashes.
This path name shows your location with reference to the top root. Pwd here tells you that you are working in the directory vdr. Which has the parent directory user, which in turn is directly under root.

The home directory

When you log into the system UNIX auto matically place you in a directry called home directory. Its created by the system when a user account is created.
The shell variable **home** knows your home directory.
$echo $home
/usr/henry

The home directory is decided by the system administrator at the time of opening a user account.

cd: changing directories

you can move around in the file system by using cd (change directory) command. It changes the current directory to the directory specify as the argument for instance,

progs

$pwd
/usr/vdr

$cd progs

$pwd
/usr/vdr/progs

The command cd progs here means change your sub directory to progs under the current directory.
You can use absolute path name also with cd command.

$pwd
/usr/vdr/progs

$cd /bin #absolute path name given

$pwd
/bin

mkdir: making directories

The mkdir(make directory) is used to create directories. The command is followed by the name of directories to be created.
A directory patch is created under current directory like this:

$mkdir patch
You can create a number of directories with one mkdir command.

$mkdir patch dbs doc #create three directories

Sometimes the system refuse to create a directory.

$mkdir test
mkdir: can't make directory test

This happens because

1) The directory test already exist
2) There may be an ordinary file exist in same directory with name test
3) User donot have permission for that

rmdir:removing directories

The rmdir(remove directory) command removes directories.

$rmdirpis
Above command will remove pis directory.
You can remove more than one directory as follows:

$rmdir pics test
Here both pics and test directory will be removed.
A subdirectory can't be remove with rmdir unless its empty.

Ln :

When a file is copied, both the original and copy occupy separate pace in the disk. Unix allows a file to have more than one name, and yet maintain a single copy in the disk. Changes in one file will also be reflected in the other. The file is said to have more than one links. A file can have as many names as you want to give it, but the only thing that is common to all of them is that they all have the same inode number/
In unix you can easily know from the fourth column of the ls –l listing that how many links are available.
This command is used to make a hard link to a file.

$ln filename1 filename2

Symbolic links or soft links are also used in unix.

```
$  ln –s note note1
$ ls –l
```

You can identify symbolic link by the character l seen in the permission field. The two files have different sizes. The notation → suggest that note1 just contain the pathnae for the file note.

chmod

there is an important default security feature provided by UNIX in the sense that it write-protects a file from all expect the owner of the file. By default all user have read permission.
This can be understands by creating 1 file and see permission of that as follows.
$ cat>xstart

```
$ ls -l xstart
-rw-r--r--   1 Vdr  None          0 Dec 11 04:08 xstart
```

As you can see the owner have read, write permission. Group, other have only read permission on xstart file. By default for user also, a file does not have executable permission. So how one does executes such a file? To do that, the permission of the file need to be changed. This is done with chmode (change mode) command.

The chmod command is use to set three permission for all three categories of users(owner, group and others) of a file. It can be used only b the owner of a file.

Syntax

chmod category operation permission filename(S)

Category	Operation	Permission
u – user	+ assign permission	r – read permission
g – group	- remove permission	w – write permission
o – other	= assign permission	x – execute permission
a- all		

e.g.
now let's assign permission (+) execute (x) permission to user (u).

```
$ chmodu+xxstart

$ ls -l xstart
-rwxr--r--   1 Vdr  None          0 Dec 11 04:08 xstart
```

To enable all of category to execute a file you can assign permission as follows.

```
$ chmodugo+xxstart          # or you can use $chmoda+xxstart

$ ls -l xstart
-rwxr-xr-x   1 Vdr  None          0 Dec 11 04:08 xstart

$ chmod a-x xstart          # taking back permission (-)  execute (x)from all users(a).

 $ ls -l xstart
-rw-r--r--   1 Vdr  None          0 Dec 11 04:08 xstart
```

Absolute assignment of permission
Absolute assignment in chmod is done using = operator. Unlike + & - permissions it assign only those permission which assign and remove other permission on a file.
e.g.

```
$ chmod a=r xstart     # assign r (read) permission to a (all) users & take back other permission.

$ ls -l xstart
-r--r--r--   1 Vdr  None          0 Dec 11 04:08 xstart
```

Assign permission using octal notation
There is short-hand notation available for changing file permissions. Chmod also takes numeric argument to describe both category, as well as the permission. The notation takes the form of an octal representation of the permission, and assign like below.

Read permission	**4**
Write permission	2
Execute permission	1

When there is more than one permission associated with a particular category, the respective numbers are added. E.g. if a file has read & write permissions for the owner , the octal representation of the owner's permission will be 4+2=6.
When these things repeated for all category we will have 3 digit octal number with each octal bit represents 1 category.
e.g.
if we wish to assign read & write (4+2=6) permission to all category we can assign then as follows.

```
$ chmod 666 xstart

$ ls -l xstart
-rw-rw-rw-   1 Vdr  None          0 Dec 11 04:08 xstart
```

Umask

When you create files and directories, the permissions assigned to them depend on the system's default settings. The UNIX system has the
Following default permissions for all files and directories

- rw-rw-rw- (octal 666) for regular files.
- rxwrwxrwx (octal 777) for directories

The current value of mask can be display by using umask without any argument.

```
$ umask
022
```

This is an octal number which has to be subtracted from the system default to obtain the actual default. This become 644(666-022) for ordinary file and 755(777-022) for directories
$umask 000
The system default permission become 666 for files and 777 for directories.

chown

chown (Change Ownership) gives away the ownership of a file to any user. This command can be run only by the owner of the file. It is used with username to whom ownership is to be given away, followed by a list of file names.

Synatx

chown<username><filenames>

To give away ownership of file note to user Sharma. The chown command can be used as follows.
e.g.
$chownsharma note

chgrp

chgrp (change group) command changes the group ownership of a file. It shares a similar syntax like chown. In following example user Sharma changes the group ownership of dept file to dba group.
e.g.
$chgrpdbadept
Both chown&chgrp use –R option to perform the action in recursive manner.

find

find is a good command for system administration. It finds the directory for files matching criteria, and then takes some action on selected files.

Syntax

Find <path_list><selection criteria><action>

Path_list

Path_list is a path where you want to search or find the file. You can simply proved the path here. If you omits this field it will find in current path.

Selection Criteria

Select Criteria	Significance
-name fname	select file fname
-user uname	select file if owned by uname
-type d	select file if it's directory
-size +x[c]	selects file if size greater than x blocks.

action

-exec cmd executes UNIX command cmd followed by {} \;
-ok cmd same as –exec expect run with permission.
-print prints selected file on standard output.

e.g.

```
$ find -name "employee"  -print                #finding “employee” file and printing
./employee

$ find -type d "test" –print             # finding “test” directory & printing
test

$ find -name "employee" -exec cat \; #finding file “employee” and display using –exec cat command.
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
```

318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000

Pg : The command pg writes the contents of a file onto the screen a page at a time. Type

$ pg science.txt

Press the [space-bar] if you want to see another page, and type [q] if you want to quit reading. As you can see, less is used in preference to catfor long files.

More:

The command more writes the contents of a file onto the screen a page at a time. Type

$ more science.txt

Press the [space-bar] if you want to see another page, and type [q] if you want to quit reading. As you can see, less is used in preference to catfor long files.

Less:

The command less writes the contents of a file onto the screen a page at a time. Type

$ less science.txt

Press the [space-bar] if you want to see another page, and type [q] if you want to quit reading. As you can see, less is used in preference to catfor long files.

More and less command can work with multiple file also.

e.g.

$ more science.txt it.txt computer.txt

$ less science.txt it.txt computer.txt

It will display the file in sequence. In between you can move to previous or next file

:n next file

:p previous file

Simple searching using less/more

Using less, you can search though a text file for a keyword (pattern). For example, to search through science.txt for the word 'science', type

$ less science.txt

$ more science.txt

then, still in less, type a forward slash [/] followed by the word to search

/science

As you can see, less finds and highlights the keyword. Type [n] to search for the next occurrence of the word.

head

The head command as the name suggest display the top of the file. When it is used without any option it display first ten records of a file.

Syntax

head [options] <file name>

e.g.

$ cat employee #content on employee file on which head command is applied.

111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000

$ head employee #without any options head command display first ten records of file.

111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700

You can specify number of lines to display from head of the files in head command. Use – symbol, followed by a number argument(no of lines).

$ head -3 employee

111|amit|sales|9000

112|anit|sales|8000
113|raj|hr|7000

If you want to get specified number of character from beginning of a file you can do this using head command with options -c <number>.
e.g.
$ head -c 20 employee #first 20 character from employee file
111|amit|sales|9000

tail

The tail command displays the end of file. It provides an additional method of addressing lines, and like head it display the last ten lines when used without argument.
When used with argument it can display last 3 record of file as follows.
e.g.
$ tail -3 employee
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000
There is one restriction tail can't display segment more than 20480 characters. The +count option allows you to address line from beginning of file instead of end. Where +count represents the line number from where selection should begin.
e.g.
$tail +11 employee #display record from 11th record.

wc

wc is a word counting program supported by unix. It counts lines, words and characters depending upon option used. It takes one or more file name as its argument & displays four columnar output(default).

syntax

wc [options] [filenames]

options

-l # display no of lines.
-w # display no of words.
-c # display no of characters.

e.g. #applying wc command on following file
$ cat file1
hello
learninglinux
wc command

$ wc file1
3 5 32 file1

In above output 1st column represents no of lines, 2nd column represents no of words, 3rd column represents no of character and 4th column display file name.

$ wc -l file1 # display no of lines of file1
3 file1

$ wc -w file1 # display no of words of file1
5 file1

$ wc -c file1 # display no of characters of file1
32 file1

touch

you sometimes require to set modification time & access time to predefined values. The touch command changes these times, and used in following manner.
touch [options] <expression><filenames>

options

-m # change modification time
-a # change access time

expression

expression consist of an eight digit number, using the format MMDDhhmm (month, day, hour &miute).

When touch command use without any option it change both modification time and access time of a file.
e.g.
$ ls -l employee # modification time
-rw-r--r-- 1 Vdr None 247 Dec 11 04:45 employee
$ ls -le employee # access time
-rw-r--r-- 1 Vdr None 247 2012-04-11 04:45:04 employee

$ touch -m 09090909 employee # changing modification time of employee.
$ ls -l employee

-rw-r--r-- 1 Vdr None 247 Sep 9 2012 employee

Split

Large files are sometimes impossible to edit with an editor, and need to be split up. The split command breaks up its input into several equi-line segments. All these segments are created as separate file in current directory.

syntax

split<filename>

Split by default breaks up a file in 1000 line pieces. It creates a group of splits files xaa, xab, xac ..xzz. you can specify no of lines to split in split command (default 1000)by specifying a size with –l option.

split - l <size><filename>

You can select your own split file grouping by specifying it as the last argument of the command line.

split - <size><filename><split file group name>

e.g.

```
$ cat emp                    #content of emp file to be split
111|amit|sales|7000
112|raj|admin|9000
113|rahul|sales|8000
114|sachin|admin|9999

$ split -l 2 emp                    # each split file will have 2 line
$ catxaa                     #content of split file xaa(default name)
111|amit|sales|7000
112|raj|admin|9000

$ catxab
113|rahul|sales|8000
114|sachin|admin|9999
```

nl

There is a separate command in the unix system elaborate schemes for numbering lines using the nl command. nl number all logical lines i.e. those containing something apart from the new line character.

$ nl department

```
1  01|accounts|6213
2  01|accounts|6213
3  02|admin|5423
4  03|marketing|6521
5  03|marketing|6521
6  03|marketing|6521
```

Operators in Redirection and Piping :

| (**piping**) --- Lets you execute any number of commands in a sequence.

The second command will be executed once the first is done, and so forth, using the previous command's output as input. You can achieve the same effect by putting the output in a file and giving the filename as an argument to the second command, but that would be much more complicated, and you'd have to remember to remove all the junkfiles afterwards. Some examples that show the usefulness of this:

ls | more --- will show you one screenful at a time, which is useful with any command that will produce a lot of output,

Redirecting the Output

We use the > symbol to redirect the output of a command. For example, to create a file called **list1** containing a list of fruit, type

$ cat> list1

Then type in the names of some fruit. Press [**Return**] after each one.

pear
banana
apple
^D {this means press [Ctrl] and [d] to stop}

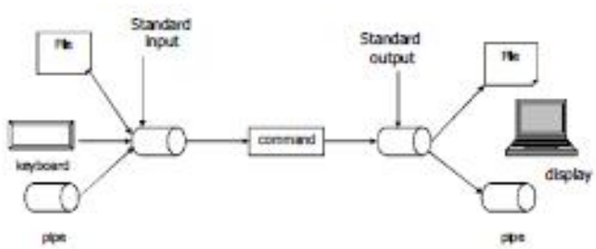
What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called **list1**

To read the contents of the file, type

$cat list1

REDIRECTION :

Standard I/O



Redirection

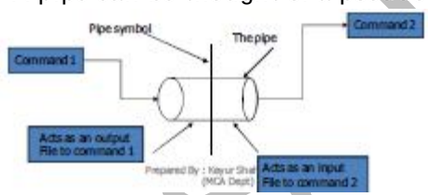
- ♣ The mechanism of changing the input source and/or output destination is called redirection.
- ♣ In unix, the redirection is accomplish by using the following operators.
  1. < input redirection
  2. > output redirection
  3. >> output redirection with appending
  4. | connecting the output of one command as input to another command.

  The file descriptor 0 and 1 is implicitly prefixed to the redirection operator < and >.
- ♣ The file descriptor 2 represents the standard error.
- ♣ The output of a program can be redirected using either > and >>.
- ♣ When > is used, destination files are overwritten.
- ♣ When >> is used, destination files are appended to existing file.

  Example:
  – $wc< sample / / input has been redirected from a file
  – $wc sample >newsample / / output has been redirected to file newsample.
  – $wc< sample >newsample / / input & output both are redirected
  or
  – $wc>newsample< sample / / input & output both are redirected
  or
  – $>newsample< sample wc
  – < operator has higher precedence than > operator
- ♣ Errors or messages can be stored exclusively in a file by redirecting them with the use of file descriptor 2 explictly.

  – $cat sample 2>errorfile
  – $cat sample 2>>errorfile

  Pipe and Pipeline – Connecting Commands
- ♣ A pipe is a general mechanism by using which, the output of one program is connected or redirected as the input to another program directly (without using the temp files in between).
- ♣ A pipe can be thought of a pseudo file that connects two programs in execution (processes).



- ♣ When an interconnection between two programs using pipe is established, the following action takes place.

– The program to the left of the pipe (command1) uses the pipe as though the pipe is an output file and
– The program to the right of the pipe (command2) uses the pipe as though it is an input file.

Example:

– $who | wc-l
– $who | grepmurthy | wc-l
OR
– $who | >grepmurthy | >wc – l

Mixing inputs from standard input and file

- ♣ A hyphen (-) is used as an argument to indicate the standard input. Such facility is useful when the used likes to mix input from the keyboard with the contents of file.

  Example:
  / /here the input from the keyboard is connected with the contents of
  letter.body
  – $cat –letter.body> letter
  – $cat /dev/stdinletter.body> letter

3.2.1 Appending to a file

The form >> appends standard output to a file. So to add more items to the file **list1**, type
$cat>> list1
Then type in the names of more fruit

```
peach
grape
orange
```

^D (Control D to stop)

To read the contents of the file, type

$cat list1

You should now have two files. One contains six fruit, the other contains four fruit.
We will now use the cat command to join (concatenate) **list1** and **list2** into a new file called **biglist**. Type

$cat list1 list2 >biglist

What this is doing is reading the contents of **list1** and **list2** in turn, then outputing the text to the file **biglist**
To read the contents of the new file, type

$catbiglist

Redirecting the Input

We use the < symbol to redirect the input of a command.
The command sort alphabetically or numerically sorts a list. Type

$sort

Then type in the names of some animals. Press [Return] after each one.

dog
cat
bird
ape
^D (control d to stop)

The output will be

ape
bird
cat
dog

Using < you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, type

$sort<biglist

and the sorted list will be output to the screen.
To output the sorted list to a file, type,

$ sort <biglist>slist

Use cat to read the contents of the file **slist**

To see who is on the system with you, type

$who

Pipes

One method to get a sorted list of names is to type,

$who> names.txt
$ sort < names.txt

This is a bit slow and you have to remember to remove the temporary file called names when you have finished. What you really want to do is connect the output of the who command directly to the input of the sort command. This is exactly what pipes do. The symbol for a pipe is the vertical bar |
For example, typing

$who | sort

will give the same result as above, but quicker and cleaner.
To find out how many users are logged on, type

$who | wc –l

Finding Patterns in Files

grep

grep is one of the most popular & useful UNIX filters. It scans a file for the occurance of a pattern, and can display the selected pattern, the line number in which they were found.

Synatx

grep [options] <pattern><filename>

Options

-c # counting occurrences of pattern.
-n # display line numbers containing the pattern.
-v # select all but the lines containing the pattern.
-I # ignore case which matching
-e # pass multiple pattern using –e options.

Regular Expression in pattern

*****	**matches zero or more occurrences of previous character**
.	matches a single character
[pqr]	matches a single character p, q, r
[^pqr]	matches a single character which is not p, q, r
^pat	matches a pat at beginning of a file

pat$	matches pattern pat at end of line.

e.g.

```
$ cat employee                    # file on which grep command is applied.
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000

$ grep "hr" employee          # grep records where pattern is “hr” from employee file.
113|raj|hr|7000
214|rahul|hr|6500
421|zaheer|hr|8700
423|irfan|hr|9000

$ grep -c "hr" employee           # counting records where “hr” pattern found.
4

$ grep -n "hr" employee           # displaying line number of matching pattern “hr”
3:113|raj|hr|7000
4:214|rahul|hr|6500
10:421|zaheer|hr|8700
11:423|irfan|hr|9000
$ grep -v "hr" employee           # displaying all records but not “hr” pattern records.
111|amit|sales|9000
112|anit|sales|8000
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
424|yusuf|sales|7000
$ grep -i "HR" employee           # ignoring case while pattern matching in grep command.
113|raj|hr|7000
214|rahul|hr|6500
421|zaheer|hr|8700
423|irfan|hr|9000
$ grep '^1..' employee        # grep records which are beginning with 1 in employee file.
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
```

egrep

The egrep (extending grep) command authored by Alfred aho, extends grep’s pattern-matching capacity. It offers all the options of grep, but its most useful feature is the facility to specify more than one pattern for search. Each pattern is separated by pipe (|).

Syntax

egrep [options] <pattern><filenames>

options

-f # specify file name in which pattern are stored.

Regular expression set used by egrep are as follows.

ch+	**matches one or more occurance of character Ch**
ch?	Matches zero or one occurance of character Ch
exp1|exp2	matches expression exp1 or expression exp2
(x1|x2)x3	matches expression x1x3 or x2x3.

e.g.

```
$ cat employee                    # file on which egrep command is applied.
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
```

```
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000
```

```
$ egrep 'hr|admin' employee # applying pattern “hr” & “admin” on employee file.
113|raj|hr|7000
214|rahul|hr|6500
216|dhoni|admin|9900
317|sachin|admin|9999
421|zaheer|hr|8700
423|irfan|hr|9000
```

```
$ egrep 'a(m|n)it' employee    # matching pattern technique in egrep.
111|amit|sales|9000
112|anit|sales|8000
```

Now we are taking example to read pattern from file and then search for that pattern on employee file using egrep command.

e.g.

```
$ cat>pat                # patterns “hr” & “admin” are stored in pat file.
hr|admin
```

```
$ egrep -f pat employee                # specifying file name in pattern.
113|raj|hr|7000
214|rahul|hr|6500
216|dhoni|admin|9900
317|sachin|admin|9999
421|zaheer|hr|8700
423|irfan|hr|9000
```

fgrep

fgrep, like grep and egrep accepts multiple pattern, both from command line and a file but unlike grep, egrep it doesn’t accept regular expression.

Alternative patterns in grep are specified by one pattern from another pattern by the newline.

e.g.

```
$ cat>pat                #pattern in file each pattern must be separated by newline.
hr
sales
```

```
$ fgrep -f pat employee                # reading pattern from file.
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000
```

histroy

history command list out 16 command most recently use.

```
$ history
1438    join a b
1439    cat a
1440    cat b
1441    clear
1442    sed '3q' employee
1443    sed -n '1,2p' employee
1444    sed -n '$p' employee
1445    sed -n '3,6!p' employee
1446    sed -n '3,10!p' employee
1447    sed '$a\
        531|vinay|hr|6000
' employee>newemp
1448    cat newemp
1449    sed '/sales/hr/d' employee >emplist
1450    sed '/sales/d' employee >emplist
1451    cat emplist
1452    clear
1453    history
```

We can find out last 5 commands by using -5 as an argument as follows.

```
$ history -5
1450    sed '/sales/d' employee >emplist
1451    cat emplist
1452    clear
1453    history
1455    history -5
```

Every command has an event number associated with it. By default, korn store all previous command in the file $HOME/.sh_history.

r

The r command is used with or without an argument to repeat previous commands. When it's used without argument it repeats last command.
When it is used with argument (event number) it repeat execute that event.

```
$ date
Thu Dec 12 04:10:04 IST 2012
$ r
date
Thu Dec 12 04:10:05 IST 2012
```

$_variable

It stores the last argument of the last command. We often run several commands with same file name, and instead of specifying the file name every time we can use $_ in its place.

```
$ cat emplist2
8000
9000
7600
9999
$ wc $_                 #use of $_ instead of filename
     4       4      20 emplist2
```

Working with columns and fields

cut

while head & tail are used to slice a file horizontally, you can slice a file vertically with the cut command. Cut identifies both column and fields and it's a useful filter for programmer.

Syntax

Cut [options] <filename>

Options

```
-c      # specifies columns from the file.
-f      # specifies fields from the file.
-d      # specifies delimiter to separate the fields.
```

Cut can be used to extract specific columns from the file.
e.g.

```
$ cat>emplist           # file on which cut command implies.
111|amit|8000
112|anit|9000
113|raj |7600
114|ram |9999

$ cut -c 5-8 emplist    #cutting characters from 5th to 8th column (employee name).
amit
anit
raj
ram
```

Here in above example the name starts from column 5th and ends at 8th columns so we are cutting them using –c option.

File often contain fixed size length record so it's better to cut fields rather than columns. Two options needed to be used here –d (delimiter) for the filed separation, and –f (field) for specifying field list.

e.g.

```
$ cut -d "|" -f 2,3 emplist     # cutting 2nd& 3rd field. Delimiter is | sign to separate field in file.
amit|8000
anit|9000
raj |7600
ram |9999
```

paste

what you cut with cut command can be pasted back with paste command. It's a special type of concatenation in that it paste files vertically rather than horizontally.

e.g.
$ cat>emplist # file on which cut command implies. File contain 3 fields.

111|amit|8000
112|anit|9000
113|raj |7600
114|ram |9999

$ cut -d "|" -f 1-2 emplist>emplist1 # cutting 1st& 2nd field from emplist& save it to emplist1

$ cut -d "|" -f 3 emplist>emplist2 # cutting 3rd field from emplist& save it to emplist2

$ paste emplist1 emplist2 #pasting content of emplist1 & emplist2 vertically using paste command.
111|amit 8000
112|anit 9000
113|raj 7600
114|ram 9999

$ paste -d \| emplist1 emplist2 #pasting with delimiter.
111|amit|8000
112|anit|9000
113|raj |7600
114|ram |9999

Join

Join command joins lines on a common field. Joins prints to the standard output line for each pair of input lines, one each from file1 and file2 that have common in that.

Syntax

Join [options] <file1><file2>

$cat 1.txt # content of file1
1 abc
2 lmn
3 pqr

$cat 2.txt #content of file2
1 abc
3 lmn
9 opq

$join 1.txt 2.txt #joining two files
1 abcabc
3 pqrlmn

since the two files are trying to match on the 1st column, and both have a 1 and a 3 in that column. To tell join to use the second column to join with, we type:
$join -1 2 -2 2 1.txt 2.txt #where "-1 2" stands for the 2nd filed of the 1st file, and "-2 2" stands for the 2nd field of the 2nd file
abc 1 1
lmn 2 3
which are just those fields that match in both files. This is refered to as an inner join. Inner joins look for rows that match rows in the other table. The problem with inner joins is that only rows that match between tables are returned.
$join -a1 -1 2 -2 2 1.txt 2.txt #(where "-a1" says include all the records from the first file.
abc 1 1
lmn 2 3
pqr 3

this is missing a number for pqr for the second file (since there is no pqr in that file) and is missing "9 opq" from the second file. This is an example of a left outer join, called such because it includes all of the rows from the left (or first) file specified.

Tools for sorting :

Sort

Using sort command as name suggest you can sort any file. Sorting starts with the first character of each line, and proceed to next character only when two lines are identical. By defaults sort reorders file in ASCII sequence.

Syntax

Sort [options] <filename>

Options

-t \ < delimiter > + <no> # t used to specify field <delimiter> used to specify delimiter used to separate field. + indicates after this field is sorted.
-r # sort file in reverse ASCII order.

-o # sort file and send output to another or same file.
-c # used to check whether file is sorted or not
-n # sort numeric file.

e.g.
$ sort -t "|" +1 emplist # sort emplist file on after 1 field and use delimiter |
111|amit|8000
113|anit|7600
112|ram|9000
114|ram|9999

$ sort -r -t "|" +1 emplist # sort empfile on reverse order
114|ram|9999
112|ram|9000
113|anit|7600
111|amit|8000

Uniq

Uniq removes duplicate records from file. The command is most useful when placed in pipelines, and can be used as SQL type query tool. Uniq simply fetches one copy of each record and writes it to the standard output.
Consider sorted file below which include duplicate lines.

Syntax

Uniq<filename>

e.g.

$ cat department # file with duplicate entry of records.
01|accounts|6213
01|accounts|6213
02|admin|5423
03|marketing|6521
03|marketing|6521
03|marketing|6521

$ uniq department
01|accounts|6213
02|admin|5423
03|marketing|6521

Compering Files :

cmp

cmp (compare) file shows that whether two files are identical (same) or not. If you want to know whether content of two file are same or not you can know this by cmp command. It compares two files and display results.

Syntax

cmp [options] <file1><file2>

two files are compared byte by byte and the location of first mismatched is echoed on the screen.cmp when invoked without options does not bother about subsequent mismatch(just display first mismatch only).if two files are identical cmp will not display anything.

e.g.
$ cat>file1
abcd

$ cat>file2
abef

$ cmp file1 file2
file1 file2 differ: char 3, line 1 # file differs at 3rd character in line 1

$ cmp -l file1 file2
3 143 145 #third character differ in ASCII value 143 - 145
4 144 146

Changing Information in Files :

tr

thetr (translate) command is one UNIX filter that manipulates individual character in a file. More specifically it translates character using one or two compact expressions.

Syntax

tr [options] expression1 expression2 standard input

the first difference that should strike you is that this command takes its input only from the standard input. It does not take filename as argument. By default it translate each character in expression1 to its mapped counterpart in expression2 .

e.g.
$ cat employee # file on which translate command to apply

```
111|amit|sales|9000
112|anit|sales|8000
113|raj|hr|7000
214|rahul|hr|6500
215|sharma|marketing|9800
216|dhoni|admin|9900
317|sachin|admin|9999
318|virat|marketing|6900
319|raina|sales|7600
421|zaheer|hr|8700
423|irfan|hr|9000
424|yusuf|sales|7000

$ tr '|/' '#/' < employee          # replacing | with # sign
111#amit#sales#9000
112#anit#sales#8000
113#raj#hr#7000
214#rahul#hr#6500
215#sharma#marketing#9800
216#dhoni#admin#9900
317#sachin#admin#9999
318#virat#marketing#6900
319#raina#sales#7600
421#zaheer#hr#8700
423#irfan#hr#9000
424#yusuf#sales#7000
```

Since tr doesn't accept a file name as argument, the input has to be redirected from a file. It can be supplied via pipe.

e.g.

```
$ head -3 employee | tr '[a-z]' '[A-Z]' # convert lowercase [a-z] to uppercase [A-Z] given input via pipe.
111|AMIT|SALES|9000
112|ANIT|SALES|8000
113|RAJ|HR|7000
```

Examining File Contents :

od

many file contains non-printing characters, and most UNIX command not show them properly. od(octal dump) is used to display the ASCII octal value of a file's contents.
The –b option with od command display the octal value of each character in file.
The –c option used to print character o file.

e.g.

```
$ cat>odfile
the ^G character rings a bell.
$ catodfile
the  character rings a bell.
```

In above example cat command not able to print the proper content from file as it contain non-printing character.

```
$ od -b odfile
0000000 164 150 145 040 007 040 143 150 141 162 141 143 164 145 162 040
0000020 162 151 156 147 163 040 141 040 142 145 154 154 056 012
0000036
$ od -bcodfile
0000000 164 150 145 040 007 040 143 150 141 162 141 143 164 145 162 040
          t   h   e       \a      c   h   a   r   a   c   t   e   r
0000020 162 151 156 147 163 040 141 040 142 145 154 154 056 012
          r   i   n   g   s       a       b   e   l   l   .  \n
0000036
```

Tools for Mathematical Calculations :

bc – The Calculator

bc is a basic calculator program provide by unix. Its GUI version is xcalc in linux. Though bc is not GUI but its extremely powerful tool.
When you invoke bc without argument, the input has to be keyed in, each line terminated by pressing <enter>. After you have finished your work, use <ctrl – d>.
e.g.

```
$ bc
bc 1.04
12+5
17              # value displayed after computation.
```

<ctrl – d>
$
While working with float value if you want precise answers then you need to set the variable scale to a value of digits after the decimal point.
e.g.
$ bc
bc 1.04
scale=3
12.200 + 12.300
24.500
Another bc feature is that base conversion. You can convert one base value to another base value using bc. It supports binary, decimal, octal, hexadecimal base.
To use base conversion feature you have to set two variable. 1st is input base (ibase) & 2nd output base (obase).
e.g.
$ bc
bc 1.04
ibase=10
obase=2
3
11

bc also support function like sqrt, cosine,sine, tangent etc.

e.g.
$ bc
bc 1.04
sqrt(196)
14

When you use cosine – c(), sine –s() , tangent – t() you have to use bc with –l option.
e.g.
$ bc
bc 1.04
scale=2
s(3.14)
0
>
If you want to send the output to a file you can use '>' symbol it will redirect your output to a file you have specified after '>' symbol.
e.g.
$cat employee >emplist # output of cat command is redirected to emplist file.

<
When there is need that you need to take input from file rather than input device (in command such as awketc). then you can use '<' symbol to give input from a file to the command.

e.g.
$tee < employee # giving input from file to tee command.

Monitoring Input and Output

tee

UNIX provides a feature by which you can save the standard output in a file, as well as display them on terminal by using tee command. Tee uses standard input and standard output, which means it can be placed anywhere in a pipeline.
The additional feature it processes is that it breaks up the input into two components; one component is saved ina file and other components display on terminal..
e.g.
$ who | tee user # output display on terminal & save in user file.
Vdr tty10 Dec 12 01:42

$ cat user # content of user file.
Vdr tty10 Dec 12 01:42

Script

Script lets you to record your login session in a file named typescript (default) in the current directory. All the command and their outputs & the error messages are stored in the file for later viewing.
You can stop recording of script using exit command.

Syntax

Script [-a] [-f] [file]
-a # appends the output to file or typescript (default) file.
-f # flush output after each write.
File # specify file name in which your session record.

e.g.

```
$ script
Script started, output file is typescript

$ cal
Decil 2012
Su Mo Tu We Th Fr Sa
1  2  3  4  5  6  7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
$ exit                                          # end of session recording.

View session record from typescript file.
Script done, output file is typescript

$ cat typescript
Script started on Wed Dec 11 00:30:40 2012
$ cal
Decil 2012
Su Mo Tu We Th Fr Sa
1  2  3  4  5  6  7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
$ exit
Script done on Wed Dec 11 00:31:36 2012
```

Banner

You can create fancy objects on your screen with the banner command (not available in Linux). This command simply produces a banner of text you supplied with argument.
By default, the character # is used to draw banner of given argument. On each line it print maximum of 10 characters in each line.

e.g.

```
$banner wait
#   #   ##     #    #####
#   #  # #     #     #
#   # #   #    #     #
# ## # ######   #     #
## ## #   #    #     #
#   # #   #    #     #
```

By default banner use # character to print banner you can use any other character to print banner by using delimiter (-d) option.

```
$banner –d$ wait
```

echo

The echo command displays its argument. User can pass argument with both quoted and unquoted format. In echo command multiple space in argument is reduce to single space by default. With echo command option for interpretation of backslash you need to use option –e. in some unix system it takes by default.

Syntax

echo [options] [string]

options

```
-n      # Do not output the trailing newline
-e      # Enable interpretation of the backslash-escaped characters listed below
\\      # print backslash
\a      # alert (BEL)
\b      # backspace
\c      #suppress trailing newline
\f      # form feed
\n      # new line
\r      # carriage return
\t      #horizontal tab
\v      # vertical tab
```

e.g.

```
$ echo "learning\nlinux"
learning
linux

$ echo "learning\tlinux"
learninglinux

$ echo "learning\blinux"
```

Learninlinux

Print variable value using echo command.
e.g.
$ a=20
$ echo $a
20

Tools For Displaying Date and Time

Cal

With cal command, you can print calendar of current month.

```
$ cal
Decil 2012
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

You can print calendar of any year by passing year as with cal command.
$cal 2012 # print calendar for 2012 year

You can also print calendar of any specific month & year using cal command.

```
$ cal 4 2012
Decil 2012
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
8  9  10  11 12 13 14
15  16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

You can print Julian calendar with options as follows.

```
$ cal -j 2 2012
     February 2012
Su  MoTu  We  Th  Fr  Sa
32  33  34  35
36  37  38  39  40  41  42
43  44  45  46  47  48  49
50  51  52  53  54  55  56
57  58  59  60
```

date - current date and time

date displays the current data and time. A superuser can set the date and time.

Syntax

date [options] [+format]

Options

```
-u      # use Universal Time (or Greenwich Mean Time)
+       # format specify the output format
%a      # weekday abbreviation, Sun to Sat
%h      # month abbreviation, Jan to Dec
%j      # day of year, 001 to 366
%n      # <new-line>
%t      # <TAB>
%y      # last 2 digits of year, 00 to 99
%D      # MM/DD/YY date
%H      # hour, 00 to 23
%M      # minute, 00 to 59
%S      # second, 00 to 59
%T      # HH:MM:SS time
```

e.g.

```
$ date
Wed Dec 11 00:50:56 IST 2012
$ date +%a
Wed
$ date +%h
Dec
$ date +%j
102
$ date +%D
```

04/11/12
$ date +%y
12
$ date +%H
00
$ date +%T
00:52:48
$ date '+%y:%j'
12:102

time

time is very useful tool for programmer. Sometimes it's necessary to know how much drainage a command makes on system resources. The time command does this work. It accepts the command as its argument and executes the program and also displays time usage on terminal.

Syntax

time<command>
e.g.
$ time sort -o empemp
real 0m0.41s
user 0m0.00s
sys 0m0.06s
The real time shown is the clock elapsed time from the invocation of the command till its termination. The user time shows the time spent by the program in executing itself. Sys indicates the used by UNIX.

Communications

Tty

Tty command will display your terminal.

Syntax

tty options

Options

-l will print the synchronous line number.
-s will return only the codes: 0 (a terminal), 1 (not a terminal), 2 (invalid options) (good for scripts)

e.g.

$tty
/dev/tty11

news

The news command is normally invoked by any user to read any message that is sent by the system administrator. The latter does that by storing the contents of the message in the file in /usr/news.
news is meant to display all files that have been created in /usr/news since the last time the command was executed.
e.g.
$news
no news

write

The write command lets you have a two-way communication with any person who is currently logged in.
One user writes a message and then waits for the reply from the other.
write uses the login name of the recipient as argument, and the text of the message from the standard input.
This is how henry starts a dialogue with Charlie:
$write Charlie
Have you completed your program?
I have completed mine-henry
<crtl-d>
The message appears on charlie's terminal provided he is logged in. if he is not, the system responds with an error message.
write: Charlie is not logged in
write also accepts standard input:
write Charlie < henry.msg # takes input from file henry.msg

mesg

communication, single or two-way can be disconcerting to a user who might be watching the output of an important program on his/her terminal. In that case user can use the mesg command to prevent such intrusions. The command

$mesg n

prevents other people from writing to user's terminal. mesg y enables receipt of such messages.
If you want to know the status of your terminal for mesg you can simply type mesg without arguments.

```
$mesg
Is y                    # can receive messages
```

mail

unlike write and talk, they all enable sending of mail to a user even if even user is not logged in. when you receive a mail message , there are several things you can do.
View it on your terminal

- ♣ Save it in a file
- ♣ Delete it
- ♣ Reply to it
- ♣ Forward it to other.

You can send mail to any user by mail <username> as follows.
e.g.

```
$mail Charlie
Subject: New System
The new system will start functioning from next month.
<ctrl – d>
```

If Charlie is running mail program the sent message doesn't directly appear to charlie's terminal. it will wait for the Charlie to finish it's task then it flashes a message that.
You have new mail

user can see their mails by just simple mail command which shows all unread mail to his/her mailbox.
Mail's prompt is the **&.** Charlie can enter the number of mail message shown in the first column to retrieve it.

There are several internal command with mail prompt and their usage are shown below.

Command	Action
+	prints the next message
-	Prints previous message
N	prints message numbered N
h	prints header of all messages
d N	delete message N
w fname	Saves currents message without header to fname file.
m user	forward mail to user
r N	replies to sender of message N
q	quit mail program

finger

finger command when used without any argument it simply produces a list of all logged users.

```
$finger
Login       Name          Tty    Idle    Login Time          Where
Henry       Henry James   *01             Fri     08:56
Tulec       tatainfotech   03     16      Fri     09:26       camcst
```

The first field shows the login name, the second column says user's full name, the third field shows the terminal of the user, preceded by * sign means terminal doesn't have write permission. The fourth field shows the idle time and the last field shows the office location that is taken from /etc/passwd(set by user then display).
Unlike who, finger can also provide detail of single user.

```
$finger sumit
Login :sumit   (message off)   Name : sumit das
Office :ballygunj
Directory : /usr/sumit         Shell : /bin/ksh
No Plan.
```

Process Related Commands :

The sh command

Sh command run or process jobs through the shell.
With sh command you can process job. This job may be listed in file. You can use number of option with sh command.

Syntax

sh [-a] [-c] [-e] [-i] [-l] [-m] [-n] [-p] [-s] [-t] [-v] [-x]

Options	Meaning
-a	Export all variables assigned to.
-c	Pass the string argument to the shell to be interpreted as input. Keep in mind that this option only accepts a single string as its argument, hence multi-word strings must be quoted.
-e	If not interactive, exit immediately if any untested command fails.
-i	Force the shell to behave interactively.

-I	Ignore EOF's from input when interactive.
-m	Turn on job control (set automatically when interactive).
-n	If not interactive, read commands but do not execute them. This is useful for checking the syntax of shell scripts.
-p	Turn on privileged mode. This mode is enabled on startup if either the effective user or group id is not equal to the real user or group id.
-s	Read commands from standard input (set automatically if no file arguments are present). This option has no effect when set after the shell has already started running (i.e. with set).
-t	Exits after reading and executing one command.
-v	The shell writes its input to standard error as it is read. Useful for debugging.
-x	Write each command to standard error (preceded by a '+ ') before it is executed. Useful for debugging.

Exporting Variables

By default, any variable is available only in the shell in which it is defined. The following sequence of commands show this:

$a=20
$sh
$echo $a
$exit
$echo $a
20

In above commands after defining **a,** we invoked a sub-shell using **sh.** This is achieved by saying sh at the prompt. We shall stay in sub-shell until we leave it either by saying **exit** or **ctrl-d.**
Since**a** was define in the parent shell it no longer holds any identity in sub-shell. We can ensure that **a** is very much alive here in parent shell using echo command.
If we want variables to be available to all sub-shell we must export them from the current shell.
$export a
After exporting variable we can invoke a sub-shell now we can obtain value of **a** in sub-shell also.
To obtain a list of all exported variables we can simply say **export** at the shell prompt.
$export
Here all variables which have been exported are display along with their values. Note that this list contains variables that have been exported by the current shell and does not contain those that may have been inherited from parent shell.
Note:

1) A variable once exported from the parent shell becomes available to the sub-shell.
2) Once a variable is exported it remains exported to all sub-shells that are subsequently executed.
3) A variable once exported remain exported if we have to unexport it we have to first unset it.
4) A variable can be exported from parent shell to its sub-shell but reverse mechanism never works.
5) If sub-shell change the value of an export variable the value of the variable in parent shell remain unchanged.

Concept of Mounting &Demounting :

Mounting File System

Most of your hard disk partitions are mounted automatically in LINUX. When you install LINUX you were asked to create partitions and indicate the mount points for that partition. Mount points are often mentioned as being candidates for separate partitions include /, /boot, /home, /user and var. The root file system (/) is the catchall for directories that aren't in other mount points.
When you boot LINUX system all LINUX partitions reside on hard disk should typically be mounted. You can mount other file system or devices such as CD-ROM, floppy disks etc using mount command.

Supported file system on Linux

Ext3: The ext file systems are most common file systems used with Linux. The ext3 file system was is currently the default file system type for Linux system. The root file system is typically ext3. The ext3 file system is also referred to s Third Extended file system. It includes features that improve a file system's capability to recover & crashes as compared to old ext2 system.
Ext2: The default file system type for old version of Linux. The features are same as ext3 but it does not include new features described above in ext2.
Ext: The first version of the ext file system. It is not used often any more.
Is09660: this file system is used for standard CD-ROM. It's used when you mount CD-ROM.
Msdos: this is an MS-DOS file system. You can use this type to mount floppy disks that come from Microsoft operating system.
Vfat: this is Microsoft extended FAT (VFAT) file system.
Swap: this is used for swap partitions. Swaps are used to hold data temporary when RAM is currently used up. Data is swapped to the swap area then returned to RAM when it is need again.
Nfs: this is the network file system type of file system. File system mounted from another computer on your network use this type of file system.
The hard disk on your local computer & remote file system set up to mount automatically when you boot Linux. The definitions for which of these file system are mounted are contained in the etc/fstab file.

Using the mount command to mount file system

Most new LINUX system runs mount –a (mount all file system) each time you boot. For this reason, you would typically only use the mount command for special situations.

Synatx

Mount <device name><mount-point>

Here in device name you have to specify the name of device which you would like to mount. In mount-point you have to specify that in your file system at which point you like to mount your device.

e.g.

mount –t /dev/hda3 /oracle

here in above example the device hda3 will be mounted on mount point /oracle.

To mount all the devices specified in /etc/fstab file you can use mount –a command which will mount all devices.

Here below shows contain of /etc/fstab file.

#mount device	mount-point	File System Type	Mount Options
/dev/hda5	swap	swap	defaults
/dev/hda6	/	ext2	defaults
/dev/hda3	/oracle	ext2	defaults
/dev/hdc	/mnt/cdrom	iso9660	ro,no-auto,user

You can mount any device by either device name or mount point name specified in /etc/fstab file. Here are the example how you can mount device.

e.g.

mount /dev/hda3

mount /oracle

you can also specify the permission for the device that how device mount on system (e.g. read only) using –o options as shown in below example.

Mount –o ro /dev/hda4 /user/local

Here above device /dev/hda4 will be mounted on mount point /user/local as read only. You can also use options like exec(execute), rw(read, write), remount (remounting the permissions) with mount command options.

unmount

when your work with mounted device over you can use unmount command to unmount the file system. This command detaches the file system from it's mount point in your linux file system. To unmount you can use either device name or mount point (directory) name.

e.g

```
unmount /oracle          # mount point name
unmount /dev/hda3        # device (file system )name
```

Text Editing with vi Editor

vi editor

The vi editor (short for visual editor) is a screen editor which is available on almost all Unix systems. Vi is a screen editor where a portion of the file is displayed on the terminal screen and the cursor can be moved around the screen to indicate where you want to make changes. You can select which part of the file you want to displayed. vi has no menus but instead uses combinations of keystrokes in order to accomplish commands.

Statringvi

To start using vi at the Unix prompt type vi followed by a file name.

If you wish to edit an existing file, type in its name; if you are creating a new file, type in the name you wish to give to the new file.

vi <filename>

e.g.

vimyfile

Then hit Return. You will see a screen similar to the one below which shows blank lines with tildes and the name and status of the file.

```
~
"myfile" : new fie
```

Modes of operation in vi

1) Command mode:

- In this mode all the key pressed by the user are interpreted like editor commands.
- In command mode the keys that are hit are not displayed on the screen.
- To use this mode press **Esc key** from keyboard.

2) Insert mode:

- This mode permits insertion of new text, editing of existing text or replacement of existing text.
- Each of these operation can be performed only after changing over from command mode to insertion mode.
- To work in insert mode press i **key from keyboard.**

3) The ex mode:

- ♣ This mode permits user to give command at the command line. The bottom of vi editor is called the command line.
- ♣ To switch to ex mode press **Esc and :** from keyboard

Command for Inserting Text

Command	Function
a	Enter text input mode and appends text after the cursor
i	Enter text input mode and inserts text at the cursor
A	Enter text input mode and appends text at the end of current line
I	Enters text input mode and insert text at the beginning of current line
o	Enter text input mode by opening a new line immediately below the current line
O	Enter text input mode by opening a new line immediately above the current line.
R	Enter text input mode and overwrite from current cursor position onwards.

Commands for deleting text

Command	Function
x	Deletes the character at current cursor position
X	Deletes the character to the left of the cursor
dw	Deletes a word or (part of a word) from the cursor to the next space
dd	Deletes the current line
d0	Deletes the current line from the cursor to the beginning of the line
d$	Deletes the current line from the cursor to the end of line

Commands for quitting vi

Command	Functions
ZZ	Writes the buffer to the file and quits vi
:wq	Writes the buffer to the file and quits vi
:w filename :q	Writes the buffer to the filename(new) and quits vi
:w! filename :q	Overwrite the existing file(filename) with the contents of the buffer and
:q!	Quits vi whether or not changes made to the buffer were written to the file
:q	Quits vi if changes made to the buffer were written to a file

Commands for positioning Cursor in window

Positioning by character

Command	Function
h	Moves the cursor 1 character to the left
Backspace	Moves the cursor 1 character to the left
l	Moves the cursor 1 character to the right
Space bar	Moves the cursor 1 character to the right
0	Moves the cursor to the beginning of the current line
$	Moves the cursor to the end of the current line

Positioning by line

Command	Function
j	Moves the cursor down 1 line from it's present position in same column.
k	Moves the cursor up 1 line from it's present position in same column.
+	Moves the cursor down to the beginning of next line.
-	Moves the cursor up to the beginning of previous line.
Enter	Moves the cursor down to the beginning of next line.

Positioning by Word

Command	Function
w	Moves the cursor to the right to the first character of the next word
b	Moves the cursor back to the first character of the previous word
e	Moves the cursor to the end of the current word

Positioning in the window

Command	Function
H	Moves the cursor to the first line on the screen
M	Moves the cursor to the middle line on the screen
L	Moves the cursor to the last line on the screen

Positioning on a numbered line

Command	Function
G	Moves the cursor to the beginning of the last line in the file
nG	Moves the cursor to the beginning of the nth line in the file

Yank and paste

When you yank vi copies that objects into buffers without removing it from file. These objects can then be pasted to the another location or in the other file.

Yanking commands

Command	Function
yw	Yanks word from cursor position
yy	Yanks lines from cursor position
y0	Yanks line from cursor position to the beginning of the line
y$	Yanks line from cursor position to end of the line.

Paste commands

Command	Function
1p	Pastes last yanked buffer
2p	Paste last but 1 yanked buffer

Pico Editor

Pico (Pine composer) is a text editor for Unix and Unix-based computer systems. It is integrated with the Pine e-mail client, which was designed by the Office of Computing and Communications at the University of Washington.

From the Pine FAQ: "Pine's message composition editor is also available as a separate stand-alone program, called PICO. PICO is a very simple and easy-to-use text editor offering paragraph justification, cut/paste, and a spelling checker..."

Pico does not support working with several files simultaneously and cannot perform a find and replace across multiple files. It also cannot copy text from one file to another.

Cursor Movement:

You can move your cursor by using the arrow keys; you may also use any of the following editing commands:

Command	Functions
Ctrl-a	Beginning of current line
Ctrl-e	End of current line
Ctrl-v	Forward one screen
Ctrl-y	Backward one screen
Ctrl-f	Forward a character
Ctrl-b	Backward a character
Ctrl-p	Previous line
Ctrl-n	Next line

Saving a File:

You can save the file you are working on by using the **Ctrl-o** (write out) command. If you exit Pico by using the **Ctrl-x** (exit) command, you will be given a chance to save the file.

Deleting Text:

Command	Function
Ctrl-d	Delete the character the cursor is on
Ctrl-e	Delete to end of current line
Ctrl-k	Delete the line the cursor is on
Ctrl-u	Restore the last line that was deleted

Cutting and Pasting:

- ♣ You can mark text for cutting and pasting by using the **Ctrl-^** command.
- ♣ After entering this command, move the cursor to highlight the text you wish to cut.
- ♣ Use **Ctrl-k** (Cut) to cut the text. Then use **Ctrl-u** (Uncut) to paste it.

Searching:

- ♣ You can search for a given text by using the **Ctrl-w** command.
- ♣ The search is NOT case sensitive. After entering the command, you will be prompted for the text for which you want to search.
- ♣ If you press Enter at this point, Pico will repeat the last search.
- ♣ All searches start at the current cursor position and wrap around to the beginning of the file.
- ♣ To move to the last line in the file, type **Ctrl-w** (where is), **Ctrl-v** (last line). To move to the first line of the file type **Ctrl-w** (Whereis), **Ctrl-y** (first line).

Spell Checking:

Pico includes an spell check function which can be invoked with Ctrl-t. It will flag words it cannot findand give you a chance to edit them, but will not suggest a correct spelling or allow you to customize the dictionary

Pico Quick Reference	
Desired Function	**Command**
General Commands	
Get help	[Ctrl] g
Write the editor contents to a file	[Ctrl] o
Save the file and exit pico	[Ctrl] x
Spell check	[Ctrl] t
Justify the text	[Ctrl] j
Moving around in your file	
Move one character to the right	[Ctrl] f or the right arrow key
Move one character to the left	[Ctrl] b or the left arrow key
Move up one line	[Ctrl] p or the up arrow key
Move down one line	[Ctrl] n or the down arrow key
Move to the beginning of a line	[Ctrl] a
Move to the end of a line	[Ctrl] e
Scroll up one page	[Ctrl] y
Scroll down one page	[Ctrl] v
Get the position of your cursor	[Ctrl] c
Searching for text in your file	
Find [pattern]	[Ctrl] w
Cutting, pasting and deleting text	
Append another file at current cursor position	[Ctrl] r
Cut a line of text	[Ctrl] k
Paste a line of text	[Ctrl] u
Mark a block of text	[Ctrl] ^ (shift 6), then move cursor to the end of the