**Unix/Linux Tutorial:**

Notes: kernel is OS,

OS/kernel- that allocates system resources and coordinate all the details:

Unix system has following 4 key bsics:

* Kernel
* Shell
* Command and utilities
* Files and directories

List of key commands:

* $whoami 🡪 who the user is.
* $login
* $passwd 🡪 change password for current user
* $ls –l 🡪 list of directory and files available in the directory
* $pwd 🡪 present working directory

System poweroff priveledge are provided only to super user or root users or system admin.

**1.1 File Management**

All data in UNIX is organized into files.

All files are organized into directories.

All directories are organized into a tree like structure called filesystem.

**Files<<directories(folders)<<filesystem**

**\*UNIX directories are equivalent to folders.**

**In UNIX there are 3 basic types of files**

* Ordinary files(files that contains data or program instruction)
* Directories( directories store both ordinary and special files, = folders in windows and mac)
* Special files(file that provide access to hardware, CD-ROM, Modem, adapter)

**Listing files:** ls –l

drwxrwxr-x 2 amrood amrood 4096 Dec 25 09:59 uml

-rw-rw-r-- 1 amrood amrood 5341 Dec 25 08:38 uml.jpg

-1st column🡪 file type and permission

-2nd column🡪 no of memory block taken by the file or directory

-3rd column🡪owner of the file

4th column🡪represent the group of the owner

5th column🡪 file size in bytes

6th column🡪 date created or modified.

7th column🡪 file or directory name.

If it starts with d 🡪 its directory, if with - 🡪 Regular files or l 🡪 symbolic link files.

**Metacharacter**

\*🡪 to match 0 or More character. (for eg: ls ch\*.doc) **file that starts with ch and ends with .doc**

**Ls \*.doc 🡪 gives you all doc files on that directory**

? 🡪 matches with single character. Eg: ls ?.doc

**Hidden Files:**

Files that starts with period (.) are hidden files. Mostly this type of files are configuration files

Eg .profile

.kshrc

How to look for invisible files: ls –a

* Single dot(.) represent current directory
* Double dot(..) represent parent directory

**Creating New Files**

Vi filename

TO Edit: press ‘I’

TO get out of Edit mode: Press ‘esc’

TO get out in files completely: Press ‘shift+ZZ’

\*can move up and down only with arrow key id the file is in edit mode. That’s ‘I’

**Reading Files(cat)**

Cat filename

Cat –b filename 🡪 display line no too.

**Word Count in a File(wc)**

Wc filename

Eg: wc chakra.txt

2 14 97 chakra.txt

Lines, words, letter and filename respectively.

**Copying Files(cp)**

Cp source\_file destination\_file

Eg:

Cp chakra.txt cp.txt

**Rename File(mv)**

mv old\_name new name

eg: Mv cp.txt cpn.txt

**Deleting Files(rm)**

Rm filename

Eg: rm cp.txt

\*CAN delete multiple files with added filename

Rm filename1.txt filename2.txt filename3.txt

**Standard Unix Stream**

On normal circumstance every UNIX program has 3 stream(files) opened for it when it start up:

* Stdin 🡪 standard input, file description =0;
* Stdout 🡪 standard output, file description =1;
* Stderr 🡪standard error, file description=2;

**1.2 Directory management**

**Home Directory(~)**

* **Cd ~**

**To go to specific usernames Home directory use**

* **Cd ~username**

**To go to the last directory use**

* **Cd –**
* **Cd / 🡪root and cd ~ 🡪home**

**Absolute/Relative Pathname**

* Absolute pathname always begins with / eg: /c/Users/passwd
* Relative path name never begins with / eg: c/Users

**\*\*current working directory(pwd)**

**Listing Directories(ls)**

Ls dirname

Eg: ls /C/Users/etc

**Creating a New Directory(mkdir)**

Mkdir mydir

Eg: mkdir /tmp/test-dir

**Creating parent directory(mkdir –p)**

In some cases when you create a directory it might show errors because it might not be associated with any parent directory. So while creating a parent directory always use –p

Eg: mkdir –p/tmp/amrood/test 🡪this creates all the required parent directory.

**Removing Directories(rmdir)**

Rmdir dirname

Eg: rmdir /dirname1/dirname2

\*\* to remove the directory make sure its empty which means there should not be any files or sub-directory inside the directory

\*\*rmdir provides no output if its successful.

**Change Directories(cd)**

Cd dirname

Cd /usr/local/bin

\*\*can define any relative or absolute path with cd and get inside that directory from any place.

\*\*can use cd ../../dirname/dname 🡪 to get into any path.

**Renaming Directories(mv)**

Mv olddir newdir

\*the file with . represent current working directory

\* the file with .. represent one level above the current working directory.

**1.3 File Permission/Access Mode**

The permission are broken into three groups: read (r), write (w) and execute (x)

For example:

-**rwx**r-xr--

*\*The first(2-4) character represent the permission of file owner. That read, write and execute(rwx)*

*\*The second(5-7) char represent the permission to group to which the file belong. That’s read and execute(r-x) but not write permission.*

*\* The last group of three characters (8-10) represents the permissions for everyone else*

\*\*File access mode(read, write and execute)

\*\*directory access mode((read, write but nothing as execute for directories)

**Changing permission**

* To change file or directory permission use **chmod(change mode)**
* There are two ways to use chmod- symbolic and absolute

**Using chmod in symbolic mode**

(+) 🡪 add new permission

(-)🡪 remove the designated permission

= 🡪 assign permission

**example:**

**ls –l testfile**

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

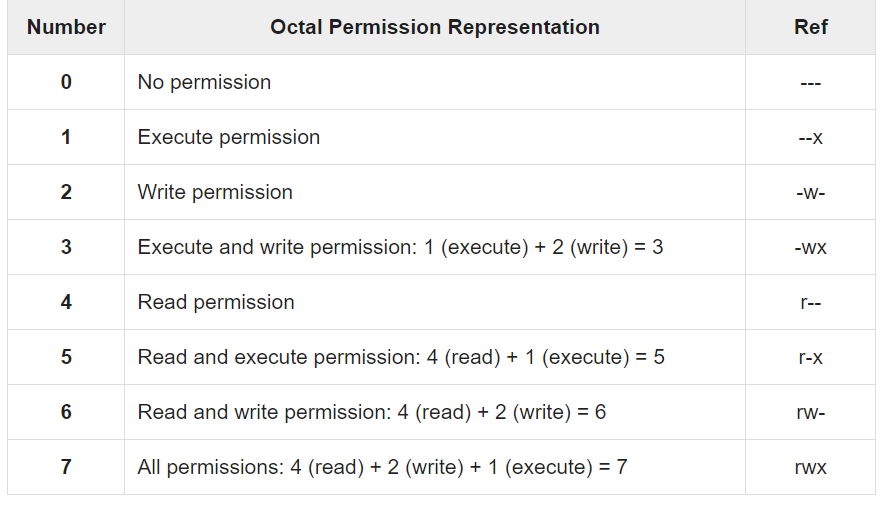
Lets change the permission using chmod.

1. Chmod o+wx testfile
2. Chmod u-x testfile
3. Chmod g=rx testfile

\*we can combine all this command on one line like:

Chmod o+wx, u-x, g=rx testfile

**Using chmod in absolute mode**



Example:

$ chmod 755 testfile

$ls -l testfile

-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 743 testfile

$ls -l testfile

-rwxr---wx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 043 testfile

$ls -l testfile

----r---wx 1 amrood users 1024 Nov 2 00:10 testfile

\*\*\*Owner(2-4),group(5-7), everyone(8-10)

Always starts with –

-rwxr-xr-x

**Changing Owner AND Group**

**Command:**

**Chown- “change owner”-** changes the ownership of the files.

**Chgrp- “change group”-** changes the group ownership of the files.

Example:

Chown user filename

Or

Chown amrood filename

\*here user can be username or userId. Above example user is changed to amrood.

**Changing group:**

**Example:**

Chgrp group filename

Or

Chgrp special filename

\*here group can be any system group or group id.

\* here it change group name group to special.

\*\*SUID AND SGID

\*\*The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners’ **execute** permission normally resides.

For example, the command −

$ ls -l /usr/bin/passwd

-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd\*

$

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

* The owner of the sticky directory
* The owner of the file being removed
* The super user, root

To set the SUID and SGID bits for any directory try the following command −

$ chmod ug+s dirname

$ ls -l

drwsr-sr-x 2 root root 4096 Jun 19 06:45 dirname

$

**Unix Scripting-Environments**

* Environment are set using variables and “echo” command is used to test the output of the variables.
* Example:

$TEST=”UNIX Programming”

$echo $TEST

**Setting up the terminal type**

If your terminal is set incorrect the output might look strange or you might not be able to interact with SHELL properly.

To avoid this: most user set their terminal to lowest denominator using”

**$TERM=vt100**

**Setting up the PATH**

When you type any command on command prompt, the shell has to locate the command before it can be executed.

The PATH variables specify the location in which the shell should look for command.

$PATH=/bin:/usr/bin

\*here each of the individual entries are separated by colons, which specify directories.

**PS1 and PS2 Variables:**

**PS1=”=>”**

**=>**

**=>**

**=>**

**YOUR PROMPT will become =>**

**More escape character, look for additional reference**

You can make the change yourself every time you log in, or you can have the change made automatically in PS1 by adding it to your **.profile** file.

When you issue a command that is incomplete, the shell will display a secondary prompt and wait for you to complete the command and hit **Enter** again.

The default secondary prompt is **>** (the greater than sign), but can be changed by re-defining the **PS2** shell variable −

**$echo “this is a**

**>test”**

**This is a test**

**$**

**The example below gives redefined PS2 with a customized prompt.**

**$PS2=”secondary prompt->”**

**$echo “this is a**

**Secondary prompt->test”**

**This is a test**

**$**

**ENVIRONMENTAL VARIABLES**

**PATH**

Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.

$ ECHO $HOME

/ROOT

$ ECHO $PATH

/usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin

**BASIC UTILITIES**

**PRINTING FILES:**

Before printing files on Unix system, it’s better to reformate the content.

Syntax:

**Pr option(s) filename(s)**

**Some pr options that can be added for formatting are: -k, -d, -h”header” etc.**

**Now to print use: lp command:**

**Example:**

**Lp filename**

**To add no of copies use -nNum**

**To choose printer use -dprinter**

**Ipstat 🡪 show printing on queue**

**Ipstat -o 🡪 show only printing u owe in queue**

**To cancel and remove use -> remove will cancel all queue on that specific printer**

**$cancel printer\_name**

**Send Email Using UNIX command**

$mail [-s subject] [-c cc-addr] [-b bcc-addr] to-addr

Example:

$mail -s "Test Message" admin@yahoo.com

\*after the above syntax press CONTROL + D and start typing the message. To STOP press . (dot)

\*we can attached a file using following command(include filename)

**$mail -s "Report 05/06/07" admin@yahoo.com < demo.txt**

**To check email: simply type**

**$mail**

**UNIX Pipes and Filters**

Grep COMMAND:

\*The grep commands search for files or lines in a file that has a certain pattern.

Example:

$ls -l | grep “Aug”

PIPE (|) symbol can be used to combine more than one command.

\*given a filename, it will look into that specific file to match the search pattern.

\*other additional options can be used along with grep command. Example

**$ls -l | grep -i "carol.\*aug"**

**-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros**

**$**

**\*here it matches either uppercase or lower case (-i)**

**SORT COMMAND**

The sort commands arrange the text alphabetically or numerically.

Example: $sort food

\*the above code will arrange alphabetically by default. There are additional options that can be added with sort to get more detailed output.

-n 🡪 sorts numerically

-r 🡪 reverse the order of sort

+x 🡪 ignores x fields while sorting.

**\*\*more than 2 commands can be linked.**

**Examples:**

**$ ls -l | grep “Aug” | sort +4n**

This pipe sorts all files in your directory modified in August by the order of size, and prints them on the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

\*can add more in above command if there are more lines to read.

**$ ls -l | grep “Aug” | sort +4n| more**

**UNIX Processes**

Whenever you issue a command in Unix, it creates, or starts, a new process. When you tried out the **ls** command to list the directory contents, you started a process. A process, in simple terms, is an instance of a running program.

The operating system tracks processes through a five-digit ID number known as the **pid** or the **process ID**. Each process in the system has a unique **pid**.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

**Starting a Process**

When you start a process (run a command), there are two ways you can run it −

* Foreground Processes
* Background Processes

### **Foreground Processes**

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

You can see this happen with the **ls** command. If you wish to list all the files in your current directory, you can use the following command −

$ls ch\*.doc

This would display all the files, the names of which start with **ch** and end with **.doc** −

ch01-1.doc ch010.doc ch02.doc ch03-2.doc

ch04-1.doc ch040.doc ch05.doc ch06-2.doc

ch01-2.doc ch02-1.doc

The process runs in the foreground, the output is directed to my screen, and if the **ls** command wants any input (which it does not), it waits for it from the keyboard.

While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

### **Background Processes**

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand (**&**) at the end of the command.

$ls ch\*.doc &

This displays all those files the names of which start with **ch** and end with **.doc** −

ch01-1.doc ch010.doc ch02.doc ch03-2.doc

ch04-1.doc ch040.doc ch05.doc ch06-2.doc

ch01-2.doc ch02-1.doc

Here, if the **ls** command wants any input (which it does not), it goes into a stop state until we move it into the foreground and give it the data from the keyboard.

That first line contains information about the background process - the job number and the process ID. You need to know the job number to manipulate it between the background and the foreground.

Press the Enter key and you will see the following −

[1] + Done ls ch\*.doc &

$

The first line tells you that the **ls** command background process finishes successfully. The second is a prompt for another command.

## Listing Running Processes(ps)

It is easy to see your own processes by running the **ps** (process status) command as follows −

$ps

PID TTY TIME CMD

18358 ttyp3 00:00:00 sh

18361 ttyp3 00:01:31 abiword

18789 ttyp3 00:00:00 ps

One of the most commonly used flags for ps is the **-f** ( f for full) option, which provides more information as shown in the following example −

$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first\_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second\_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

## Stopping Processes

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when the process is running in the foreground mode.

If a process is running in the background, you should get its Job ID using the **ps** command. After that, you can use the **kill** command to kill the process as follows −

$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first\_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second\_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

$kill 6738

Terminated

Here, the **kill** command terminates the **first\_one** process. If a process ignores a regular kill command, you can use **kill -9** followed by the process ID as follows −

$kill -9 6738

Terminated

## Parent and Child Processes

Each unix process has two ID numbers assigned to it: The Process ID (pid) and the Parent process ID (ppid). Each user process in the system has a parent process.

Most of the commands that you run have the shell as their parent. Check the **ps -f** example where this command listed both the process ID and the parent process ID

**UNIX/LINUX COMMUNICATION**

When you work in a distributed environment, you need to communicate with remote users and you also need to access remote Unix machines.

There are several Unix utilities that help users compute in a networked, distributed environment.

## The ping Utility

The **ping** command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not.

The ping command is useful for the following −

* Tracking and isolating hardware and software problems.
* Determining the status of the network and various foreign hosts.
* Testing, measuring, and managing networks.

### **Syntax**

Following is the simple syntax to use the ping command −

$ping hostname or ip-address

The above command starts printing a response after every second. To come out of the command, you can terminate it by pressing **CNTRL + C** keys.

## The ftp Utility

Here, **ftp** stands for **F**ile **T**ransfer **P**rotocol. This utility helps you upload and download your file from one computer to another computer.

The ftp utility has its own set of Unix-like commands. These commands help you perform tasks such as −

* Connect and login to a remote host.
* Navigate directories.
* List directory contents.
* Put and get files.
* Transfer files as **ascii**, **ebcdic** or **binary**.

### **Syntax**

Following is the simple syntax to use the ping command −

$ftp hostname or ip-address

The above command would prompt you for the login ID and the password. Once you are authenticated, you can access the home directory of the login account and you would be able to perform various commands.

The following tables lists out a few important commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **put filename**  Uploads filename from the local machine to the remote machine. |
| 2 | **get filename**  Downloads filename from the remote machine to the local machine. |
| 3 | **mput file list**  Uploads more than one file from the local machine to the remote machine. |
| 4 | **mget file list**  Downloads more than one file from the remote machine to the local machine. |
| 5 | **prompt off**  Turns the prompt off. By default, you will receive a prompt to upload or download files using **mput** or **mget** commands. |
| 6 | **prompt on**  Turns the prompt on. |
| 7 | **dir**  Lists all the files available in the current directory of the remote machine. |
| 8 | **cd dirname**  Changes directory to dirname on the remote machine. |
| 9 | **lcd dirname**  Changes directory to dirname on the local machine. |
| 10 | **quit**  Helps logout from the current login. |

It should be noted that all the files would be downloaded or uploaded to or from the current directories. If you want to upload your files in a particular directory, you need to first change to that directory and then upload the required files.

### **Example**

Following is the example to show the working of a few commands −

$ftp amrood.com

Connected to amrood.com.

220 amrood.com FTP server (Ver 4.9 Thu Sep 2 20:35:07 CDT 2009)

Name (amrood.com:amrood): amrood

331 Password required for amrood.

Password:

230 User amrood logged in.

ftp> dir

200 PORT command successful.

150 Opening data connection for /bin/ls.

total 1464

drwxr-sr-x 3 amrood group 1024 Mar 11 20:04 Mail

drwxr-sr-x 2 amrood group 1536 Mar 3 18:07 Misc

drwxr-sr-x 5 amrood group 512 Dec 7 10:59 OldStuff

drwxr-sr-x 2 amrood group 1024 Mar 11 15:24 bin

drwxr-sr-x 5 amrood group 3072 Mar 13 16:10 mpl

-rw-r--r-- 1 amrood group 209671 Mar 15 10:57 myfile.out

drwxr-sr-x 3 amrood group 512 Jan 5 13:32 public

drwxr-sr-x 3 amrood group 512 Feb 10 10:17 pvm3

226 Transfer complete.

ftp> cd mpl

250 CWD command successful.

ftp> dir

200 PORT command successful.

150 Opening data connection for /bin/ls.

total 7320

-rw-r--r-- 1 amrood group 1630 Aug 8 1994 dboard.f

-rw-r----- 1 amrood group 4340 Jul 17 1994 vttest.c

-rwxr-xr-x 1 amrood group 525574 Feb 15 11:52 wave\_shift

-rw-r--r-- 1 amrood group 1648 Aug 5 1994 wide.list

-rwxr-xr-x 1 amrood group 4019 Feb 14 16:26 fix.c

226 Transfer complete.

ftp> get wave\_shift

200 PORT command successful.

150 Opening data connection for wave\_shift (525574 bytes).

226 Transfer complete.

528454 bytes received in 1.296 seconds (398.1 Kbytes/s)

ftp> quit

221 Goodbye.

$

## The telnet Utility

There are times when we are required to connect to a remote Unix machine and work on that machine remotely. **Telnet** is a utility that allows a computer user at one site to make a connection, login and then conduct work on a computer at another site.

Once you login using Telnet, you can perform all the activities on your remotely connected machine. The following is an example of Telnet session −

C:>telnet amrood.com

Trying...

Connected to amrood.com.

Escape character is '^]'.

login: amrood

amrood's Password:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* \*

\* WELCOME TO AMROOD.COM \*

\* \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Last unsuccessful login: Fri Mar 3 12:01:09 IST 2009

Last login: Wed Mar 8 18:33:27 IST 2009 on pts/10

{ do your work }

$ logout

Connection closed.

C:>

## The finger Utility

The **finger** command displays information about users on a given host. The host can be either local or remote.

Finger may be disabled on other systems for security reasons.

Following is the simple syntax to use the finger command −

Check all the logged-in users on the local machine −

$ finger

Login Name Tty Idle Login Time Office

amrood pts/0 Jun 25 08:03 (62.61.164.115)

Get information about a specific user available on the local machine −

$ finger amrood

Login: amrood Name: (null)

Directory: /home/amrood Shell: /bin/bash

On since Thu Jun 25 08:03 (MST) on pts/0 from 62.61.164.115

No mail.

No Plan.

Check all the logged-in users on the remote machine −

$ finger @avtar.com

Login Name Tty Idle Login Time Office

amrood pts/0 Jun 25 08:03 (62.61.164.115)

Get the information about a specific user available on the remote machine −

$ finger amrood@avtar.com

Login: amrood Name: (null)

Directory: /home/amrood Shell: /bin/bash

On since Thu Jun 25 08:03 (MST) on pts/0 from 62.61.164.115

No mail.

No Plan.

**Vi Editor**

In this chapter, we will understand how the vi Editor works in Unix. There are many ways to edit files in Unix. Editing files using the screen-oriented text editor **vi** is one of the best ways. This editor enables you to edit lines in context with other lines in the file.

An improved version of the vi editor which is called the **VIM** has also been made available now. Here, VIM stands for **Vi IM**proved.

vi is generally considered the de facto standard in Unix editors because −

* It's usually available on all the flavors of Unix system.
* Its implementations are very similar across the board.
* It requires very few resources.
* It is more user-friendly than other editors such as the **ed** or the **ex**.

You can use the **vi** editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

Starting the vi Editor

The following table lists out the basic commands to use the vi editor −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **vi filename**  Creates a new file if it already does not exist, otherwise opens an existing file. |
| 2 | **vi -R filename**  Opens an existing file in the read-only mode. |
| 3 | **view filename**  Opens an existing file in the read-only mode. |

Following is an example to create a new file **testfile** if it already does not exist in the current working directory −

$vi testfile

The above command will generate the following output −

|

~

~

~

~

~

~

~

~

~

~

~

~

"testfile" [New File]

You will notice a **tilde** (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

You now have one open file to start working on. Before proceeding further, let us understand a few important concepts.

Operation Modes

While working with the vi editor, we usually come across the following two modes −

* **Command mode** − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
* **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

**Hint** − If you are not sure which mode you are in, press the Esc key twice; this will take you to the command mode. You open a file using the vi editor. Start by typing some characters and then come to the command mode to understand the difference.

Getting Out of vi

The command to quit out of vi is **:q**. Once in the command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or use **:wq** and return.

The easiest way to **save your changes and exit vi** is with the ZZ command. When you are in the command mode, type **ZZ**. The **ZZ** command works the same way as the **:wq** command.

If you want to specify/state any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type **:w filename2** and return.

Moving within a File

To move around within a file without affecting your text, you must be in the command mode (press Esc twice). The following table lists out a few commands you can use to move around one character at a time −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **k**  Moves the cursor up one line |
| 2 | **j**  Moves the cursor down one line |
| 3 | **h**  Moves the cursor to the left one character position |
| 4 | **l**  Moves the cursor to the right one character position |

The following points need to be considered to move within a file −

* vi is case-sensitive. You need to pay attention to capitalization when using the commands.
* Most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2j** moves the cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Remember that you must be in the command mode (**press Esc twice**). The following table lists out a few commands to move around the file −

Given below is the list of commands to move around the file.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **0 or |**  Positions the cursor at the beginning of a line |
| 2 | **$**  Positions the cursor at the end of a line |
| 3 | **w**  Positions the cursor to the next word |
| 4 | **b**  Positions the cursor to the previous word |
| 5 | **(**  Positions the cursor to the beginning of the current sentence |
| 6 | **)**  Positions the cursor to the beginning of the next sentence |
| 7 | **E**  Moves to the end of the blank delimited word |
| 8 | **{**  Moves a paragraph back |
| 9 | **}**  Moves a paragraph forward |
| 10 | **[[**  Moves a section back |
| 11 | **]]**  Moves a section forward |
| 12 | **n|**  Moves to the column **n** in the current line |
| 13 | **1G**  Moves to the first line of the file |
| 14 | **G**  Moves to the last line of the file |
| 15 | **nG**  Moves to the **nth** line of the file |
| 16 | **:n**  Moves to the **nth** line of the file |
| 17 | **fc**  Moves forward to **c** |
| 18 | **Fc**  Moves back to **c** |
| 19 | **H**  Moves to the top of the screen |
| 20 | **nH**  Moves to the **nth** line from the top of the screen |
| 21 | **M**  Moves to the middle of the screen |
| 22 | **L**  Move to the bottom of the screen |
| 23 | **nL**  Moves to the **nth** line from the bottom of the screen |
| 24 | **:x**  Colon followed by a number would position the cursor on the line number represented by **x** |

Control Commands

The following commands can be used with the Control Key to performs functions as given in the table below −

Given below is the list of control commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **CTRL+d**  Moves forward 1/2 screen |
| 2 | **CTRL+f**  Moves forward one full screen |
| 3 | **CTRL+u**  Moves backward 1/2 screen |
| 4 | **CTRL+b**  Moves backward one full screen |
| 5 | **CTRL+e**  Moves the screen up one line |
| 6 | **CTRL+y**  Moves the screen down one line |
| 7 | **CTRL+u**  Moves the screen up 1/2 page |
| 8 | **CTRL+d**  Moves the screen down 1/2 page |
| 9 | **CTRL+b**  Moves the screen up one page |
| 10 | **CTRL+f**  Moves the screen down one page |
| 11 | **CTRL+I**  Redraws the screen |

Editing Files

To edit the file, you need to be in the insert mode. There are many ways to enter the insert mode from the command mode −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **i**  Inserts text before the current cursor location |
| 2 | **I**  Inserts text at the beginning of the current line |
| 3 | **a**  Inserts text after the current cursor location |
| 4 | **A**  Inserts text at the end of the current line |
| 5 | **o**  Creates a new line for text entry below the cursor location |
| 6 | **O**  Creates a new line for text entry above the cursor location |

Deleting Characters

Here is a list of important commands, which can be used to delete characters and lines in an open file −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **x**  Deletes the character under the cursor location |
| 2 | **X**  Deletes the character before the cursor location |
| 3 | **dw**  Deletes from the current cursor location to the next word |
| 4 | **d^**  Deletes from the current cursor position to the beginning of the line |
| 5 | **d$**  Deletes from the current cursor position to the end of the line |
| 6 | **D**  Deletes from the cursor position to the end of the current line |
| 7 | **dd**  Deletes the line the cursor is on |

As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2x** deletes two characters under the cursor location and **2dd** deletes two lines the cursor is on.

It is recommended that the commands are practiced before we proceed further.

Change Commands

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **cc**  Removes the contents of the line, leaving you in insert mode. |
| 2 | **cw**  Changes the word the cursor is on from the cursor to the lowercase **w** end of the word. |
| 3 | **r**  Replaces the character under the cursor. vi returns to the command mode after the replacement is entered. |
| 4 | **R**  Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting. |
| 5 | **s**  Replaces the current character with the character you type. Afterward, you are left in the insert mode. |
| 6 | **S**  Deletes the line the cursor is on and replaces it with the new text. After the new text is entered, vi remains in the insert mode. |

Copy and Paste Commands

You can copy lines or words from one place and then you can paste them at another place using the following commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **yy**  Copies the current line. |
| 2 | **yw**  Copies the current word from the character the lowercase w cursor is on, until the end of the word. |
| 3 | **p**  Puts the copied text after the cursor. |
| 4 | **P**  Puts the yanked text before the cursor. |

Advanced Commands

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

Given below is the list advanced commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **J**  Joins the current line with the next one. A count of j commands join many lines. |
| 2 | **<<**  Shifts the current line to the left by one shift width. |
| 3 | **>>**  Shifts the current line to the right by one shift width. |
| 4 | **~**  Switches the case of the character under the cursor. |
| 5 | **^G**  Press Ctrl and G keys at the same time to show the current filename and the status. |
| 6 | **U**  Restores the current line to the state it was in before the cursor entered the line. |
| 7 | **u**  This helps undo the last change that was done in the file. Typing 'u' again will re-do the change. |
| 8 | **J**  Joins the current line with the next one. A count joins that many lines. |
| 9 | **:f**  Displays the current position in the file in % and the file name, the total number of file. |
| 10 | **:f filename**  Renames the current file to filename. |
| 11 | **:w filename**  Writes to file filename. |
| 12 | **:e filename**  Opens another file with filename. |
| 13 | **:cd dirname**  Changes the current working directory to dirname. |
| 14 | **:e #**  Toggles between two open files. |
| 15 | **:n**  In case you open multiple files using vi, use **:n** to go to the next file in the series. |
| 16 | **:p**  In case you open multiple files using vi, use **:p** to go to the previous file in the series. |
| 17 | **:N**  In case you open multiple files using vi, use **:N** to go to the previous file in the series. |
| 18 | **:r file**  Reads file and inserts it after the current line. |
| 19 | **:nr file**  Reads file and inserts it after the line **n**. |

Word and Character Searching

The vi editor has two kinds of searches: **string** and **character**. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the last line of the screen, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place −

* The **/** command searches forwards (downwards) in the file.
* The **?** command searches backwards (upwards) in the file.

The **n** and **N** commands repeat the previous search command in the same or the opposite direction, respectively. Some characters have special meanings. These characters must be preceded by a backslash (**\**) to be included as part of the search expression.

|  |  |
| --- | --- |
| **Sr.No.** | **Character &Description** |
| 1 | **^**  Searches at the beginning of the line (Use at the beginning of a search expression). |
| 2 | **.**  Matches a single character. |
| 3 | **\***  Matches zero or more of the previous character. |
| 4 | **$**  End of the line (Use at the end of the search expression). |
| 5 | **[**  Starts a set of matching or non-matching expressions. |
| 6 | **<**  This is put in an expression escaped with the backslash to find the ending or the beginning of a word. |
| 7 | **>**  This helps see the '**<**' character description above. |

The character search searches within one line to find a character entered after the command. The **f** and **F** commands search for a character on the current line only. **f** searches forwards and **F** searches backwards and the cursor moves to the position of the found character.

The **t** and **T** commands search for a character on the current line only, but for **t**, the cursor moves to the position before the character, and **T** searches the line backwards to the position after the character.

Set Commands

You can change the look and feel of your vi screen using the following **:set** commands. Once you are in the command mode, type **:set** followed by any of the following commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **:set ic**  Ignores the case when searching |
| 2 | **:set ai**  Sets autoindent |
| 3 | **:set noai**  Unsets autoindent |
| 4 | **:set nu**  Displays lines with line numbers on the left side |
| 5 | **:set sw**  Sets the width of a software tabstop. For example, you would set a shift width of 4 with this command — **:set sw = 4** |
| 6 | **:set ws**  If *wrapscan* is set, and the word is not found at the bottom of the file, it will try searching for it at the beginning |
| 7 | **:set wm**  If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: **:set wm = 2** |
| 8 | **:set ro**  Changes file type to "read only" |
| 9 | **:set term**  Prints terminal type |
| 10 | **:set bf**  Discards control characters from input |

Running Commands

The vi has the capability to run commands from within the editor. To run a command, you only need to go to the command mode and type **:!** command.

For example, if you want to check whether a file exists before you try to save your file with that filename, you can type **:! ls** and you will see the output of **ls** on the screen.

You can press any key (or the command's escape sequence) to return to your vi session.

Replacing Text

The substitution command (**:s/**) enables you to quickly replace words or groups of words within your files. Following is the syntax to replace text −

:s/search/replace/g

The **g** stands for globally. The result of this command is that all occurrences on the cursor's line are changed.

Important Points to Note

The following points will add to your success with vi −

* You must be in command mode to use the commands. (Press Esc twice at any time to ensure that you are in command mode.)
* You must be careful with the commands. These are case-sensitive.
* You must be in insert mode to enter text.