Linux Notes

Vim:

Vim is a text editor that is an upgraded version of the Vi editor and is more compatible with Vi. The most usage of vi editors is to create a new file, edit an existing file, or just read a file. Vim editor is more useful in editing different kinds of plain text. This command is more used in editing programs.

There are many up gradations done to Vi editor like multiple windows can be opened at a time, multi-level undo option and buffers, syntax highlighting, filename completion, command-line editing, on-line help, and many more.
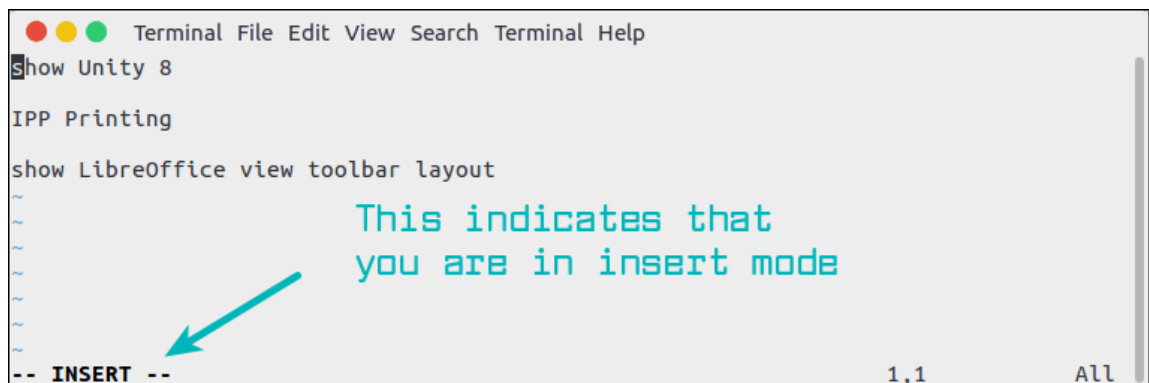
Vim has three modes:

1. <u>Command Mode</u>: When you start Vim, you are placed in Command mode. In this mode, you can move across the screen, delete text and copy text.

2. <u>Insert Mode</u>: You cannot write text in command mode. To write text (or let's say insert text) into a file, there is a dedicated insert mode. When you want to write something on a file, you must enter the insert mode.

3. <u>Visual mode</u>: Somewhat like command mode but here, you can use the arrow keys to select text across lines (instead of directly working on words and lines based on cursor position).

Once you know the Vim modes, let's see some basic Vim commands for various purposes.

Entering insert mode in Vim

Always remember, i stands for insert mode. You press the 'i' key to enter the insert mode.

When you enter the insert mode, you'll see — INSERT — at the bottom of the editor/terminal screen. This indicates that you are in insert mode.

Vim Insert Mode

But "i" is not the only way to enter the insert mode. There are several other commands to enter the insert mode in Vim. The cursor position is important while entering the insert mode.

i – New text will appear before the cursor

a – New text will appear after the cursor

I – New text will appear at the beginning of the current line

A – Next text will appear at the end of the current line

o – A new line is created after the current line

O – A new line is created before the current line

Going back to command mode in Vim

You start in command mode; you go in to insert mode. If you want to move back to the command mode, press Esc (escape) key.

When you have entered your text, I advise hitting Esc key to enter command mode. This way, you won't enter any new text unknowingly, involuntarily. Tip: If you cannot remember anything else, just remember to press 'i' for insert mode and Esc key for command mode. This should be sufficient.

Moving around in Vim

When you are in Vim, you cannot use your mouse to click at in point in the screen like normal text or code editors. This is why you need to know the movement commands to a certain line or word or position in Vim.

Vi/Vim purists will suggest using h, j, k and l keys for moving up, left, right and down respectively when you are in the command mode. While this is applicable to both Vi and Vim editors, I don't prefer using these weird key-combinations.

When the Vi editor was first developed, most keyboards didn't have arrow keys. This is why h,j,k,l keys were used for the movement.

If you are new to Linux or to Vim, you may opt for the arrow keys. No harm done.

However, if you want to be a bit more proficient with Vim, you may want to memorize a few Vim shortcuts for easily moving around the screen.

H – Move to the top of the screen. Note that it doesn't always mean moving to the first line in the file. It's just for moving to the top of the visible screen.

L – Move to the bottom of the screen. Note that it doesn't always mean moving to the last line in the file. It's just for moving to the bottom of the visible screen.

M – Move to the middle of the screen.

[[– Move to the first line of the file. You can also use gg here.

]] – Move to the last line of the file. You can also use G here.

nG – Move to line number n. Note that you won't see anything on the screen while typing the line numbers.

Tip: You can display line numbers in Vim by going into the command mode and typing :set number

As you can see, with these additional movement commands, it will be easier for you to move around in a big text file. You can learn more movement commands like {and} for moving back and forth paragraph, w for moving word by word etc.

You can also use 'repeater modifiers' like nG you just saw with most commands in Vim. For example, if you use 5w, it will move 5 words. If you use 6H, it will move to the 6th line from the top of the screen.

Repeater modifiers also work with 'i' and this is where many people make mistakes. If you have accidentally pressed a number like 7 before pressing 'i' to go into insert mode, whatever text you type will be added 7 times. You won't see it on the screen immediately until you start saving the document.

Undo your changes in Vim:

Vim allows you to undo and redo your changes. You can also redo your changes. It's all applicable in the same session, of course. Once you have saved your changes, you cannot undo it.

To undo a change, go to command mode and press 'u' key. You can press it several times for multiple undo actions.

If you want to redo a change, press Ctrl+r key combination in the command mode. You can press it several times for multiple redo actions.

Deleting in Vim

Apart from undoing your changes, you might also want to delete some text from the file. In Vim, you can always use the Delete key for deleting a character but there are a few key combinations for better handling the deletion in Vim.

x – Delete the character at the current cursor position just like a delete key

X – Delete the character before the current cursor position like a backspace key. Note that the backspace key itself doesn't work in Vim.

dw – Delete word. Actually, it deletes from the current cursor position till the end of the current word plus the white space after it.

dd – Delete the current line.

d$ – Delete from the current cursor position till the end of the line.

dG – Delete from the current cursor position till the end of the file.

Note: There are no cut commands in Vim because when you delete something, the deleted text is placed into the buffer. In other words, delete commands are the cut commands.

How to Delete All Lines of a File in Vim [Quick Tip]?

'Esc+gg+dG' is what you need to do for deleting all the lines of the file in Vim.

Copy-Pasting in Vim

You might wonder how to copy paste in Vim. This is a legitimate concern because you won't always type in the texts.

There are two kinds of copy and paste in Vim. One that deals with the buffer and one that deals with the 'external' text.

When you are in command mode, you can either use your Linux terminal shortcuts for copying the text or the following key combinations for copying text:

yw – Copy word. Actually, it copies from the current cursor position till the end of the current word plus the white space after it.

yy – Copy current line.

y$ – Copy from the current cursor position till the end of the line.

yG – Copy from the current cursor position till the end of the file.

Suppose you used one of the delete commands discussed above. The deleted text is placed into the buffer. You can paste this text from the buffer using these two paste commands:

p – Paste the contents of the buffer before the cursor position

P – Paste the contents of the buffer after the cursor position

Note that if you have deleted an entire line, the contents will be placed before or after the current line with the above-mentioned paste commands.

The paste commands only work with the Vim buffer. What about the text you copied from some other file? In those cases, you can use the standard copy and paste key combinations of your Linux Terminal.

In Ubuntu and many other Linux distribution's default terminal, Ctrl+Shift+C is used for copying and Ctrl+Shift+V is used for pasting. In some other terminals, selecting a text copies it and you can paste it using the right click.

Searching for text in Vim

Finding a particular text is an important function of a text editor. You can search for text in the file in Vim using '/'.

In the command mode, use '**/**' and then type your search text and press enter. You'll see whatever you are typing in the bottom left of the screen.

It will do a forward searching for the searched term from your cursor position. You can also use '?' and then type your search term and press enter to perform a backward search. Note that the search is case sensitive.

If there is more than one match for your search text, you can jump to the next position by pressing n key. If you want to go back to the previous match, press N. Basically, with n and N you can cycle through all the matches. Tip: You can use \c option to run a case-insensitive search in Vim. Example: /\cMy_Search.

Search and Replace in Vim

Vim provides a substitute command (:s) for searching and replacing text. It relies heavily on regular expressions (regex) for search and replace.

You can use it in the following fashion:

:%s/foo/bar/g

The above command will replace all the 'foo' with 'bar' in the entire file. The 'g' at the end is responsible for extending the search and replace function on all the matches. Otherwise, only the first match is replaced.

:s/foo/bar/g

The ':s' command will do the exact same function as above but only in the current line instead of the entire file.

By default, the search is case sensitive. To make it case insensitive, you can use the flag 'i' along with g.

:%s/foo/bar/gci

I added an additional 'c' flag here. Actually, you might not feel comfortable with the idea of your entire matching text getting replaced in the entire document within seconds. This is where 'c' flag helps a lot. With this confirm flag, Vim will ask you to confirm if you want to perform a replacement on each match.

In confirmation mode, you'll be presented with the following option: replace with UU (y/n/a/q/l/^E/^Y)?

Let me explain the choices here:

y – YES, replace this match

n – NO, don't replace this match and move to the next one

a – Replace ALL matches

q – QUIT without replacing any match

l – Replace this match and quit as if it was the LAST match

^E – Scroll the screen up

^Y – Scroll the screen down

Tip: If you can't remember everything else, just remember :%s/foo/bar/gci will try to replace all the matches in the entire file with your confirmation.

Saving and Quitting Vim

You have just learned the basics of Vim commands. It's time to save your work and exit Vim. To save or exit Vim, you must enter the command mode first by press Esc key. And then you can use the following options:

:w – Save the changes but don't exit

:wq – Save and quit

:q – Just Quit (if you have unsaved changes, you'll see this warning: E37: No write since last change (add ! to override))

:q! – Force quit (it will discard any unsaved changes)
:w! – Force save.

File Permissions:

```
ubuntu@ip-172-31-30-185:~$ ls -l
total 36
drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 28 07:20
bash_scripts
-rw-rw-r-- 1 ubuntu ubuntu   22 Jul 28 05:32 demo1
-rw-rw-r-- 1 ubuntu ubuntu    0 Jul 28 05:23 demo2
-rw-rw-r-- 1 ubuntu ubuntu   46 Jul 28 06:55 demo3.txt
-rw-rw-r-- 1 ubuntu ubuntu   27 Jul 28 07:00 demo4.txt
-rw-rw-r-- 1 ubuntu ubuntu   27 Jul 28 07:01 demo5.txt
-rw-rw-r-- 1 ubuntu ubuntu   22 Jul 28 05:25 directory
-rw-rw-r-- 1 ubuntu ubuntu   22 Jul 28 05:32 documents
drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 28 05:17 file
-rw-rw-r-- 1 ubuntu ubuntu   46 Jul 28 07:07 marks.txt
ubuntu@ip-172-31-30-185:~$
```

Above Box, in blue colours are Directories files & Remaining are regular files.

Here, drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 28 07:20 bash_scripts.

Where d –Directory

```
Owner ----------> Group

-rw-rw-r-- 1 ubuntu ubuntu   22 Jul 28 05:32 demo1

Regular file        Other
```
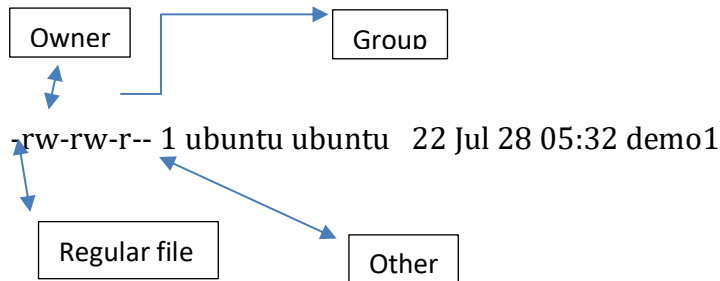
Permissions:

Permissions can be modified by Two types. They are 1) Numerical

2)Alphabet

Chmod:- it is a cmd to change permissions in Linux.

Eg:- chmod 777 demo1

```
ubuntu@ip-172-31-30-185:~$ chmod 777 demo1
ubuntu@ip-172-31-30-185:~$ ls -l

Before cmd is
-rw-rw-r-- 1 ubuntu ubuntu    22 Jul 28 05:32 demo1

After cmd is
-rwxrwxrwx 1 ubuntu ubuntu    22 Jul 28 05:32 demo1
ubuntu@ip-172-31-30-185:~$
```

For shortcut, we have to know some data about how change permissions by numerical or Alphabet notation's.

Owner/ User – u, Group – g , Others – o

Read – r (or) 4, Write – w (or) 2, Execute – x (or) 1

Add – '+', Remove – '-' , Assign – '='

Permission string	Octal code	Meaning
rwxrwxrwx	777	Read, write, and execute permissions for all users.
rwxr-xr-x	755	Read and execute permission for all users. The file's owner also has write permission.
rwxr-x---	750	Read and execute permission for the owner and group. The file's owner also has write permission. Users who aren't the file's owner or members of the group have no access to the file.
rwx------	700	Read, write, and execute permissions for the file's owner only; all others have no access.
rw-rw-rw-	666	Read and write permissions for all users. No execute permissions for anybody.
rw-rw-r--	664	Read and write permissions for the owner and group. Read-only permission for all others.
rw-rw----	660	Read and write permissions for the owner and group. No world permissions.
rw-r--r--	644	Read and write permissions for the owner. Read-only permission for all others.
rw-r-----	640	Read and write permissions for the owner, and read-only permission for the group. No permission for others.
rw-------	600	Read and write permissions for the owner. No permission for anybody else.
r--------	400	Read permission for the owner. No permission for anybody else.

Eg: chmod u+x demo1

  chmod u+x-w demo2

```
ubuntu@ip-172-31-30-185:~$ chmod u+x demo1
ubuntu@ip-172-31-30-185:~$ chmod u+x-w demo2
ubuntu@ip-172-31-30-185:~$ ls -l
total 36
drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 28 07:20
bash_scripts
-rwxrwxrwx 1 ubuntu ubuntu   22 Jul 28 05:32 demo1
-r-xrw-r-- 1 ubuntu ubuntu    0 Jul 28 05:23 demo2
```

Creating and Deleting User Accounts

To create a new standard user, use the useradd command. The syntax is as follows:

useradd <name>

The useradd command utilizes a variety of variables, some of which are shown in the table below:

Option	Description	Example
-d <home_dir>	home_dir is used as the value for the user's login directory	useradd <name> -d /home/<user's home>
-e <date>	the date when the account expires	useradd <name> ** -e <YYYY-MM-DD>
-f <inactive>	the number of days before the account expires	useradd <name> -f <0 or -1>
-s <shell>	sets the default shell type	useradd <name> -s /bin/<shell>

User database

Local user information is stored in the plain-text /etc/passwd file: each of its lines represents a user account, and has seven fields delimited by colons.

account:password:UID:GID:GECOS:directory:shell

Where:

- account is the user name. This field cannot be blank. Standard *NIX naming rules apply.
- password is the user password.

> **Warning:** The passwd file is world-readable, so storing passwords (hashed or otherwise) in this file is insecure. Instead, Arch Linux uses shadowed passwords: the password field will contain a placeholder character (x) indicating that the hashed password is saved in the access-restricted file /etc/shadow. For this reason, it is recommended to always change passwords using the passwd command.

- UID is the numerical user ID. In Arch, the first login name (after root) for a so called normal user, as opposed to services, is UID 1000 by default; subsequent UID entries for users should be greater than 1000.
- GID is the numerical primary group ID for the user. Numeric values for GIDs are listed in /etc/group.
- GECOS is an optional field used for informational purposes; usually it contains the full user name, but it can also be used by services such as finger and managed with the chfn command. This field is optional and may be left blank.
- directory is used by the login command to set the $HOME environment variable. Several services with their own users use /, but normal users usually set a directory under /home.
- shell is the path to the user's default command shell. This field is optional and defaults to /bin/bash.

Example:

```
java:x:1001:1001::/home/java:/bin/sh
```