

# Ubuntu - Nano editor, links, file permission, GNU compiler

Usman Wajid  
usman.wajid88@gmail.com

## 1 NANO Editor

There are many text editors available for Linux. At the moment you will have access to the nano editor. Nano is an advanced text editor provided by GNU. Simply typing nano on the shell will give you the editor. You may also start your nano by explicitly mentioning the file you want to work on. This file may be already existing or you may be creating a new one.

```
nano <filename>
```

Near the end of your screen you will see a list of shortcuts. The ones which you should get yourself familiar with are as such:

CTRL+X for exit

CTRL+O for saving

CTRL+W for searching

CTRL+K for cutting

CTRL+U for pasting

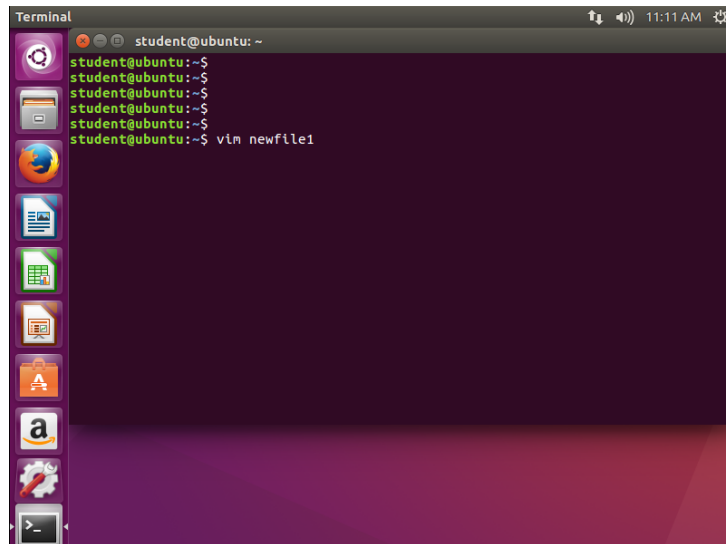
CTRL+C for displaying cursor position

Other commands are listed at the bottom of the text-editor window.

## 2 vim editor

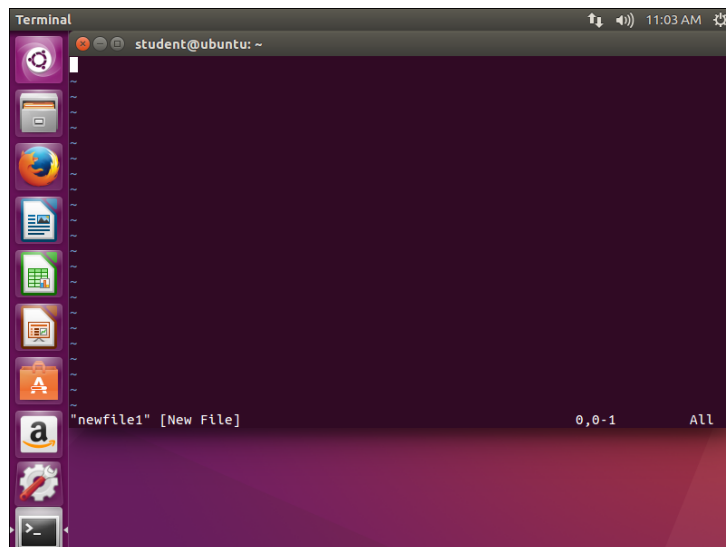
Vim is a powerful text editor used in CLI (command line interface). Linux uses a lot of configuration files, you'll often need to edit them and vim is a great tool to do so. Alternatives to vim are the command-line editors nano and joe. You can enter into the vim editor by using the **vim** command,

```
vim newfile1
```



A terminal window titled "Terminal" with a dark purple background. The window shows a series of shell prompts "student@ubuntu: ~\$" followed by the command "vim newfile1". The terminal is open on the Ubuntu desktop, with a sidebar on the left containing icons for various applications like the Dash, Home, Files, Firefox, LibreOffice, and Amazon. The system clock in the top right corner shows 11:11 AM.

```
student@ubuntu: ~  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$ vim newfile1
```



The same terminal window now shows the vim editor interface. The status bar at the bottom indicates the file is "newfile1" [New File], the cursor is at line 0, column 1, and the mode is "All". The terminal is still open on the Ubuntu desktop, with the sidebar and system clock visible. The system clock now shows 11:03 AM.

```
newfile1" [New File] 0,0-1 All
```

## Modes in vim editor

There are three basic modes that are widely used in the vim editor,

### 2.1 Insert mode

The Insert mode lets you insert text in a document. The shortcut is: "i" (insert text where the cursor is) or "o" (insert text at the beginning of the following line).

- To toggle into the insert mode press **ALT + i** or **ALT + O**
- To go out of the insert mode press **escape**

### 2.2 Visual Mode

The visual mode permits the user to select the text as you would do with a mouse but using the keyboard instead of the mouse. Useful to copy several lines of text for example. The shortcut is: "V".

- cut/delete: **d**
- undo: **u**
- paste: **p**

### 2.3 Command Mode

When you are in another mod you can use the escape key (sometimes you'll need to hit it twice) to come back to command mod at any time. Some of the commands in the command mode are as follows,

- save: **:w**
- save and exit: **:wq**
- exit: **:q**
- force: **!** (example **:w! :q!**)
- vertical split: open a document and then type **:vsplit /path-to-document/document** and this will open the specified document and split the screen so you can see both documents. Use **ctrl + w + (h/j/k/l)** to switch between vertical windows
- copy: **y**
- copy a line: **yy**
- paste: **p**
- cut: **d**
- cut a line: **dd**

### 3 Links

Links are the equivalent of Shortcuts in Windows. The syntax of creating a link to a file or directory is as such:

```
ln existingfile linkname
```

```
$ls
```

```
$mkdir newdir
```

```
$echo "hello world" > newdir/newfile
```

```
$cat newdir/newfile
```

```
$ln newdir newdir_hardlink
```

```
$ln -s newdir newdir_softlink
```

```
$ls ~lh
```

```
$ls newdir_softlink/
```

```
$cat newdir_softlink/newfile
```

```
$mv newdir dirnew
```

```
$ls ~lh
```

```
$ls newdir_softlink/
```

```
$mv dirnew newdir
```

```
$ls ~lh
```

```
$ls newdir_softlink/
```

```
$cat newdir_softlink/newfile
```

```
$echo "hellow world" > newdir_softlink/newfile
```

```
$cat newdir_softlink/newfile
```

```
$cat newdir/newfile
```

```
$ls ~lh
```

```
$echo "hello world" > newfile
```

```
$ls -lh

$cat < newfile

$ln newfile newfile_hardlink

$cat newfile_hardlink

$echo "Hello dear" > newfile_hardlink

$cat < newfile

$cat < newfile_hardlink

$ls ~lh

$mv newfile filenew

$cat < filenew

$cat < newfile_hardlink

$rm -rf newdir newdir_softlink newfile newfile_hardlink

$ls ~lh
```

Look carefully at the contents of both filepointer and dirpointer. Changing one will automatically change the other. Also look at how the directory pointers appear using `ls -lh`.

## 4 More | Less

Sometimes a user may give a command which generates so much output that it scrolls very fast. As a result, the top information simply flows out of the screen whereas only the low end information is visible. For example, use the following command

```
ls /bin -lh
```

We can view the lost information using the `more` or `less` commands. Try both of them first.

```
ls /bin -lh | more
```

To quit, press `q`.

```
ls /bin -lh | less
```

The syntax of both above commands are such that we are specifying two commands in one go. The first command starts with `ls`, the second command starts with `more` or `less`. Both these commands are joined by the pipe symbol `|`. With `more`, you are able to browse down the display 1-page at a time using spacebar. With `less`, you are able to browse up and down the display 1-line at a time using up and down arrow keys. An alternative is using output redirection that you have covered in earlier sections. Usage would be as such:

```
ls /bin -lh > newfile.txt
nano newfile.txt
```

## 5 Searching

By default, searching for files or directories is performed using the find command. The syntax of find command is as such:

```
find <where> -name <what>
```

So, if I want to find all pdf files in / directory, I will give:

```
find / -name *.pdf
```

## 6 File Permissions

All files and directories in Linux have an associated set of owner permissions that are used by the operating system to determine access. These permissions are grouped into three sets of three bits. The sets represent owner, group, everyone else whereas the bits represent read, write, execute. So overall, we have 9 permissions, as shown in the following table:

	Read	Write	Execute
<b>Owner</b>	1	1	1
<b>Group</b>	1	1	0
<b>Others</b>	0	0	1
	$2^2$	$2^1$	$2^0$

Representation of file permissions

If we look at the first row for Owner, we see three 1's. Which specify that the Owner has read, write, and execute permissions on a file or directory. Since all of the bits are in allow mode, we can add them up together to get  $2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7$ . Similarly, the second row for Group, i.e., users within the same group as that of the file owner, have read and write, but no execute permissions. This will be translated only as  $2^2 + 2^1 = 4 + 2 = 6$ . And lastly, every other user can only execute files and have no permission to read or write to them. This will be translated as  $2^0 = 1$ . So the file permissions would be 761. To give the permission of 761 to a file, we would use the command:

```
chmod 761 <filename>
```

You can view the file permissions using the command:

```
ls -lh
```

## 7 GNU Compiler Collection

Programs written in C on linux are compiled using the gcc compiler. Programs written in C++ are compiled using the g++ compiler. Both these compilers are provided by GNU under the label GCC. Any C program written for linux should have the extension of .c (e.g., hello.c), whereas a C++ program should have an extension of .cpp (e.g., hello.cpp). When compiling, the general syntax of command is as follows:

```
gcc first.c -o first
g++ second.cpp -o second
```

A breakdown is as such:

- gcc | g++ is the compiler itself.
- first.c | second.cpp will be the filename with extension of your source code.
- -o is the output filename. It is the argument for making an executable file with the name you specify. Without this, the source code will be compiled into an executable file named a.out first is the name of executable file which you pass to -o argument.

So considering that we have a source code with the name first.c, and compile it with the above command, we can execute it using:

```
./first
```

Where ./ specifies the current directory, and first is the name of executable file which you passed as an argument to the compiler. Try it out using the following code:

```
#include <stdio.h>

main() {
    printf("hello, world\n");
}
```