

# GeeksZine

*Free Open Source is Fun*



Issue : October 2010

Website : [www.richnusgeeks.com](http://www.richnusgeeks.com)

Publisher : RichNusGeeks Consulting

# Table of Content

Just two words ..... 3

Is that so? ..... 4

Banishing a ghost ..... 6

Surprising the master ..... 11

Learning the hard way ..... 12

Searching for Buddha ..... 16

The last rap ..... 19

## **Just two words, *an editorial***

Hello everybody, here comes October and also a new issue of the zine. First of all, a small change in the punch line of the zine with an additional word “Free”. Richard Stallman always emphasizes the use of Free with Open Source Software and really it matters as “Open” doesn't mean necessarily Free. I heartfelt this vital difference in last few months a lot.

We are very sorry for not publishing the zine last month due to some unforeseen reasons. But we assure you all that we are working hard on this regularity thing and are trying to publish the zine twice a month in the very near future. This issue of the zine is having a new section and a new contributor too. Please welcome Konark Modi to the contributor/reviewer panel of GeeksZine. We are also planning to add many new sections one by one every month so we hope that those would make the zine more interesting and appealing to a large community of floss geeks. This issue of the zine contains some cool tricks with few interesting shell commands to extract more fun and profit from your GNU/Linux box. Also there are contributed sections on Shell Scripting, CDargs and Grep along with an interesting comparison between C and Python.

This zine is from geeks, for the geeks and of the geeks. So if anyone of you think of any weird section and stuff to add to zine or want to provide feedbacks then please drop us a line. We would be more than indebted to add your section and stuff to the zine and work upon your feedbacks. So all the geekheads, we hope to see you every month with more improved GeeksZine and add more fun to this world with free open source software.

Keep hacking

Ankur Kumar

RichNusGeeks Consulting

geekszine@yahoo.in

geekszine@gmail.com

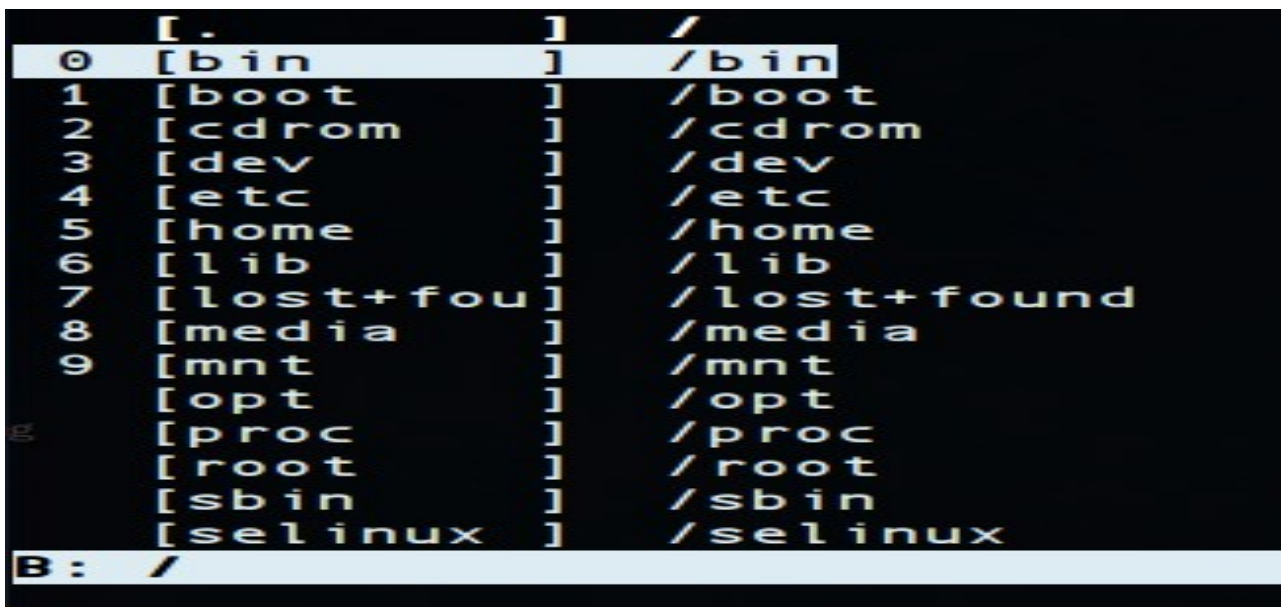
## Is that so?, *for the first timers to floss*

Are you tired of typing full path names to move to directories in your system? I found this utility known as CDargs a few days ago which helps you bookmark directories you most frequently visit. Ubuntu users can get it from synaptic package manager and CDargs download page (<http://www.skamphausen.de/cgi-bin/ska/CDargs>) also provides RPMs and source code if you wish to compile yourself.

After you have installed CDargs you need to set up the shell functions that are included with it. To set up CDargs function with bash you need to edit your .bashrc file in your home directory to include the following:

```
source /DR/cdargs-bash.sh
```

replace DR with the path to the file cdargs-bash.sh . Debian users can find the file under /usr/share/doc/cdargs/examples. If you have installed the RPM or compiled from source, the path will be different. Now run the same command above to make the shell functions take effect immediately. Now type cv to test the cd browser; you will see all the subdirectories in your present directory.



```
0 [.] ] /
1 [bin ] /bin
2 [boot ] /boot
3 [cdrom ] /cdrom
4 [dev ] /dev
5 [etc ] /etc
6 [home ] /home
7 [lib ] /lib
8 [lost+found ] /lost+found
9 [media ] /media
[mnt ] /mnt
[opt ] /opt
[proc ] /proc
[root ] /root
[sbin ] /sbin
[selinux ] /selinux
B: /
```

There are two modes for cv browser and list that are indicated by B and L in the lower bar which shows your current directory. In the browser mode

you can navigate the browser mode using arrow keys or emacs, vi movement keys. Cv numbers only first 10 directories in your current directory and to see hidden directories press the period key. Press the tab key to toggle between list and browser mode.

If you are starting CDargs for the first time, you won't have any directories bookmarked. Press 'a' to add the current directory in the bookmarks list with the directory name without the path as the bookmark name. For example you have a directory /usr/local/include/linux then on bookmarking the bookmark name will be 'linux' .If you want to give your bookmark another name type 'A' instead of 'a' to bookmark and you will be prompted for a name. Press 'e' to edit the bookmarks file in your default text editor. Bookmarks are stored in the .cdargs file in your home directory. To know about other commands press 'H' or '?'. Press 'Esc' or 'ctrl+D' to come out from browser.

Ok! so now use **cv BOOKMARK\_NAME** to go to the directory associated with that bookmark or type **cdb** to see all the bookmarked directories and choose one of them. If you are in a directory which you want to bookmark type 'ca' and type 'ca name' to give bookmark name you want. To copy or move files to your bookmarked directories use 'cpb' and 'mvb' commands for example :

cpb filename bookmark and mvb filename bookmark.

All The Best! I hope this utility will make your cli experience better.

**Anirudhh Dev**

[www.twitter.com/rudi009](http://www.twitter.com/rudi009)

## Banishing a ghost, *transition to floss*

Terminal plays an important role in GNU/Linux. Well! It's the time to get started with it. There are two types of user interfaces, graphical user interfaces (GUI) and command line interface (CLI). In GUI, you use mouse to give instructions and you type commands to control your computer in command line. GNU/Linux provides two ways of exploring command line, you can either start a terminal emulator or switch to virtual console.

### Terminal Emulator

Terminal emulator is a program that provides access to command line when you are in GUI. It opens up a window in which you can type commands. To start a Terminal Emulator look in the application menu.



### Virtual Consoles

To switch to virtual console press **<Ctrl><Alt><F1>** i.e. press and hold control key then press and hold Alt, now press F1 key. It will give you a log in prompt where you can type in your user name and press enter. Now type your password to login. While typing the password you won't see it on screen. You can type commands after logging in. There are six virtual consoles by default. To switch to second one press **<Ctrl><Alt><F2>** and so on. To switch back to GUI press **<Ctrl><Alt><F7>**.

```
Fedora release 13 (Goddard)
Kernel 2.6.33.3-85.fc13.i686 on an i686 (tty1)

localhost login: _
```

## Normal users and root user

When you open up a terminal or login into virtual console it will show you something like `[username@hostname~]$` e.g. my laptop shows `[sumit@sumit-laptop ~]$`. The \$ sign means that you are normal user and you only have limited powers. A root user has many more powers than a normal user and as a root user you change all the critical system settings. There are some commands that only a root user can run. To become a root user press enter after typing `su -` (if you are using Slackware, Fedora or Red Hat based distribution) or `sudo -i` (if you are using Debian based distribution like Ubuntu).

It will ask for your root password please enter it and also it is important to note that while you are typing the password it will not be displayed on the screen for security reasons press enter after your are done typing the password. For Ubuntu this password is same as your user password, Fedora and others asks you to set root password during installation.

After you become a root, the prompt will look something like `[root@hostname~]#` . Now you are root user. Since a root user can make critical changes and even delete important system files so be careful with what you type as a root user. Only login as root user when necessary.

## Basic Linux Commands

In GNU/Linux the general syntax of any command is **command name** **[options]** **[arguments]**. Options usually start with `-` e.g. in command `ls -l /home`, `-l` is option and `/home` an argument. The things that are written in

[] brackets means that they are optional i.e. you may or may not use them its your choice but you may use them to modify the behavior of command according to your needs. If you don't use them some defaults options and/or arguments will be used.

### **1. pwd (print working directory)**

To find out where you are in file system tree i.e. current directory type :

```
[user@host~]$pwd
```

```
/home/sumit
```

So by looking at the output, we can say that I am currently in `/home/sumit` directory.

### **2. ls (list directory contents)**

Now you know that you are currently in `/home/sumit`. Now you can type `ls` to list all the files and directories in `/home/sumit` directory. This command also accepts arguments and options like most of linux commands. Using `ls` with option `-l` i.e. typing `ls -l` will give you more detailed output. If you type `ls /home`, the command will display all the files in `/home` folder. Here `/home` is taken as argument.

### **3. cd (change directory)**

You can you use `cd` to change you current directory (`pwd`). While using `cd` you can either specify relative path or absolute path. Relative path depends on your current directory (output of `pwd` command) i.e. their result depend upon your current location in file system tree, absolute paths always start from `/` and their result does not depend on your current directory. Let's say your current directory is `/home/sumit` and `/home/sumit` contains a folder named `Documents`.

I can type `cd Documents` to go to `Documents` directory (relative path) or type `/home/sumit/Documents`. The former only works when my current directory is `/home/sumit`. If you type `cd ..` then it will take you to the parent

directory i.e. if you are in /home/sumit/ typing cd.. will take you to /home, now typing cd .. once more will take you one step higher to / directory.

```
sumit@localhost: ~ 75x21
[sumit@localhost ~]$ pwd
/home/sumit
[sumit@localhost ~]$ cd ..
[sumit@localhost home]$ pwd
/home
[sumit@localhost home]$ cd ..
[sumit@localhost /]$ pwd
/
[sumit@localhost /]$ cd /home/sumit/
[sumit@localhost ~]$ pwd
/home/sumit
[sumit@localhost ~]$ ls
Desktop/ Documents/ Downloads/ Music/ Pictures/ Public/ Videos/
[sumit@localhost ~]$ cd Documents/
[sumit@localhost Documents]$ pwd
/home/sumit/Documents
[sumit@localhost Documents]$ cd
[sumit@localhost ~]$ pwd
/home/sumit
[sumit@localhost ~]$
```

#### 4. mkdir (make directories)

You can use mkdir command to create new directories, here you can use absolute or relative paths just like cd command.

[user@hostname~]\$ mkdir dirname (create a directory named mkdir in current directory)

[user@hostname~]\$ mkdir /home/sumit/test2 (creates directory /home/sumit/test2 using absolute path).

#### 5. Using Tab auto completion

You can use auto completion to save time when typing commands or filenames. If you are typing mkdir command, just type mkd and press tab, it will auto complete the command, if it doesn't press tab twice and it will list all the commands that has mkd as its first three characters. Similarly you can type ls /ho and press tab is will auto complete the command to ls /home.

## 6. Using History

All the commands you type are stored in history. You can move b/w history by using upper and downward arrow keys on your keyboard and press enter to execute. Although you can also type command **history** and you will see that the commands you types in the past with a number. Want to run the command directly without typing again, type “!**number**” and the command on that number will get executed.

I hope it should be enough to get you started with the terminal.

**Sumit Rai**

[sumit.tux@gmail.com](mailto:sumit.tux@gmail.com)

## Surprising the master, *cool tricks*

- I/O redirection is a very interesting feature of shell. But sometimes we want to redirect the output of a command to some file as well as we want to see that output on the standard output (by default, text console screen). Shell provides a command known as **tee** for this purpose. For example, type the following command on a text console to redirect the output of `ls -lhrt` command to a file as well as to the standard output :

**ls -lhrt | tee lslhrt.txt**

you could explore more about `tee` by typing **man tee** in a text console.

- Most of the shell commands provide a recursive option to act upon a recursive hierarchy of the directories. For example if you want the listing of the subdirectories in the current directory then you can type:

**ls -R -l ... ,**

if you want to remove all the directories and their contents recursively then you can type:

**rm -R ... ,**

if you want to search a pattern in all the files recursively then you can use `grep` (explained in the next section of the zine) like:

**grep -R ... ,**

so the recursive option saves us all the labor otherwise required to apply the various operations on subdirectories and files contained in those.

## Learning the hard way, *cool floss software tools*

There are many instances when you need to search something in a file and at times this might be a daunting task. There is a very useful utility known as GREP that is provided with most of the Unix based operating systems. It's also known as g/re/p (global regular expression). The synopsis of the grep command is :

```
grep [options] pattern filename
```

Now let us try an understand grep in more detail via examples.

### **Eg file: sample-file.txt**

```
An apple a day keeps the doctor away
APPLE
apple123
orange
```

### **1. General usage of grep :**

#### **a. Searching string in a file**

```
grep 'string-to-search' file-to-search
grep apple sample-file.txt
output: An apple a day keeps the doctor away
APPLE
apple123
```

#### **b. Counting the number of matches the pattern would return**

```
grep -c 'string-to-search' file-to-search
grep -c apple sample-file.txt
output: 3
```

#### **c. Put the contents of a match in another file**

```
grep 'string-to-search' file-to-search > file-to-write
grep apple sample-file.txt > results.txt
Output : cat results.txt
```

An apple a day keeps the doctor away  
APPLE  
apple123

**d. Case sensitive search (by default, case sensitive)**

```
grep -i 'string-to-search' file-to-search  
grep -i APPLE sample-file.txt  
output: APPLE
```

**e. Invert selection**

```
grep -v 'string-to-search' file-to-search  
grep -v apple sample-file.txt  
output: orange
```

**f. Exact match**

```
grep -w 'string-to-search' file-to-search  
grep -x 'string-to-search' file-to-search  
grep -w apple sample-file.txt  
output: APPLE
```

**g. Line nos. for search (example of php programs) with filename (-H)**

```
grep -n 'function-to-search' *.php  
grep -n -H 'init()' *.php
```

This would list all the filenames with line numbers where it encounters the function `init()`. E.g: searches for a function inside JavaScript file

```
grep function *.js
```

**2. Using Regular Expressions in grep :**

|                       |   |
|-----------------------|---|
| grep apple files      | { search files for lines with 'apple' } |
| grep '^Apple' files   | { 'Apple' at the start of a line }      |
| grep 'Apple\$' files  | { 'Apple' at the end of a line }        |
| grep '^apple\$' files | { lines containing only 'apple' }       |
| grep '[Aa]pple' files | { search for 'Apple' or 'apple' }       |

```
grep 'B[oO][bB]' files {search for BOB, Bob, BOb or BoB }
grep '^$' files         {search for blank lines}
grep '[0-9][0-9]' file  {search for pairs of numeric digits}
```

### 3. Efficiently searching files using grep (using switches like A,B,C) :

At times there is a need to check the records around the match too. Grep gives us this power to search around the result too using switches A,B,C.

```
grep -A3 'apple' file (Displays three results after each match for Apple)\
grep -B3 'apple' file (Displays three results before each match for Apple)
grep -C3 'apple' file (Displays three results around(both below and above)
                    each match for Apple)
```

### 4. Searching for multiple patterns :

At times you want multiple patterns to be searched, that can be easily achieved in two ways:

#### a. Matching more than one pattern using pipe

```
grep 'abc \| 123' sample_file.txt
```

#### b. Matching more than one pattern using file

Put all the patterns to be used in a file and then GREP command with the switch -f:

```
Pattern_File:pattern_file.txt
```

```
test
abc
123
```

```
Sample File:sample_file.txt
```

```
def
abc
123
```

```
grep -f pattern_file.txt sample_file.txt
output:
```

abc  
123

## 5. Using colors to highlight the results :

Playing with environment variable to highlight the match with diff. colors  
The GREP\_COLOR environment variable controls which color is used. To change the color from red to something else, set GREP\_COLOR to a numeric value according to this chart:

|    |        |
|----|--------|
| 30 | black  |
| 31 | red    |
| 32 | green  |
| 33 | yellow |
| 34 | blue   |
| 35 | purple |
| 36 | cyan   |
| 37 | white  |

For example, to have matches highlighted in a shade of green:

```
GREP_COLOR=32; export GREP_COLOR; grep pattern myfile
```

This would highlight the results in the color set in the environment variable.

## 6. Using g/re/p in editors :

To use GREP in editor like vi and vim you can use the syntax g/regular-expression/p.

**Konark Modi**

<http://twitter.com/konarkmodi>

## Searching for Buddha, *thoughts about floss technologies*

Python is often compared to other interpreted languages such as Java, JavaScript, Perl, Tcl, or Smalltalk. Comparisons to C++, Common Lisp and Scheme can also be enlightening. In this article, I am gonna compare the python with the most renowned and important language “C”.

### 1. Working Style

Python is an interpreter based language. It means that it reads your code line by line and then gives you the output. If the interpreter finds an error it stops working over there and gives you the error. You can understand this by :

source code → interpreter → output

C is a compiler based language. You have to compile the whole code before you execute it. That's the main reason behind its speed. You make a small change in your code, you have to compile it again. It tells you the errors in compile time. So the scheme for compiler based language goes like this :

source code → compiler → object code → executor → output

### 2. Code Length

Python is known for its short code as compare to another languages. Its code is 2-3 times shorter than the similar C language source code. Let's look at this example where I am going to print the numbers from 0-999 using both languages. So have a look at the code length then :

#### **In Python :**

```
i=0
```

```
while i<100:
```

```
    print i
```

```
    i=i+1
```

# number of lines = 4

## In C :

```
#include<stdio.h>
int main()
{
    int i=0;
    while (i<100)
    {
        printf(“%d\n”,i);
        i=i+1;
    }
}
```

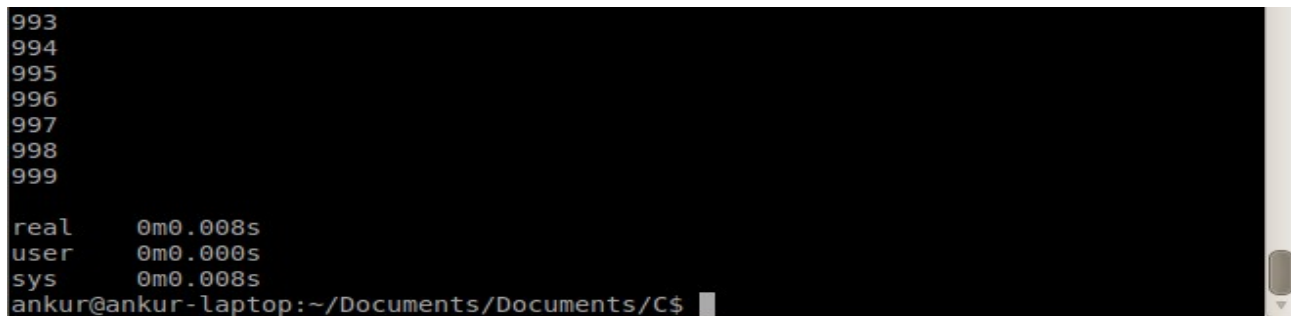
#number of lines = 10

So you can see the code ratio of python and C that's 2:5. Python is known for the flexibility of the source code. So in short, you have to code less to get the same result in python.

## 3. Performance

C is known for its unmatched speed. Its compiled nature makes it one of the fastest language in the world. However python can match its speed to some extent using its the psyco module. I execute the same code i.e. printing number from 0-999 and found my results as follow :

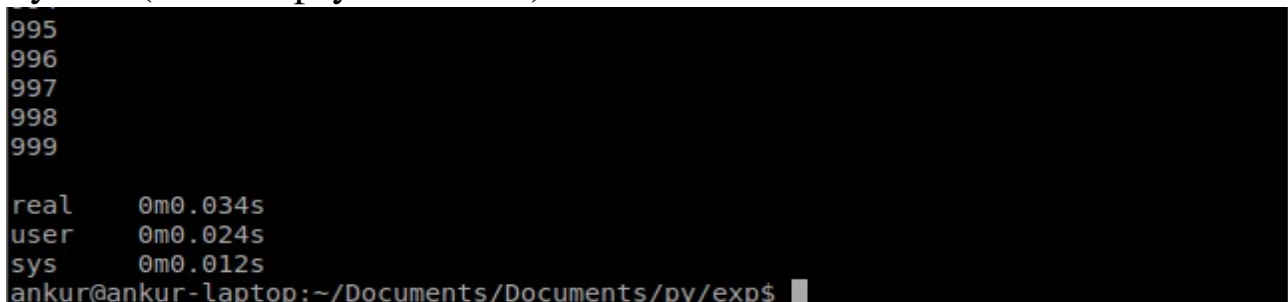
C :

A terminal window showing the output of a C program. The output consists of the numbers 993, 994, 995, 996, 997, 998, and 999, each on a new line. Below the numbers, the timing information is displayed: real 0m0.008s, user 0m0.000s, and sys 0m0.008s. The prompt shows the user is ankur@ankur-laptop in the directory ~/Documents/Documents/C\$.

```
993
994
995
996
997
998
999

real    0m0.008s
user    0m0.000s
sys     0m0.008s
ankur@ankur-laptop:~/Documents/Documents/C$
```

Python (without psyco module) :

A terminal window showing the output of a Python program. The output consists of the numbers 995, 996, 997, 998, and 999, each on a new line. Below the numbers, the timing information is displayed: real 0m0.034s, user 0m0.024s, and sys 0m0.012s. The prompt shows the user is ankur@ankur-laptop in the directory ~/Documents/Documents/py/exp\$.

```
995
996
997
998
999

real    0m0.034s
user    0m0.024s
sys     0m0.012s
ankur@ankur-laptop:~/Documents/Documents/py/exp$
```

## Python (with psyco module)

```
993
994
995
996
997
998
999

real    0m0.028s
user    0m0.020s
sys     0m0.008s
ankur@ankur-laptop:~/Documents/Documents/py/exp$
```

|        | C        | Python<br>(without psyco) | Python<br>(with psyco) |
|--------|----------|---------------------------|------------------------|
| Real   | .008 sec | .034 sec                  | 0.028 sec              |
| User   | .000 sec | .024 sec                  | 0.020 sec              |
| System | .008 sec | .012 sec                  | 0.08 sec               |

Though the execution speed of C is unmatched but still python can match it up to certain level through its psyco module.

**Ankur Aggarwal**

[coolankur2006@gmail.com](mailto:coolankur2006@gmail.com)

[www.twitter.com/ankurtwi](http://www.twitter.com/ankurtwi)

## **The last rap, *an epilogue***

All the thoughts and the information presented in this zine are based upon the various freely and openly available resources on the internet and the personal experiences. So we don't guarantee the fitment of the opinions and the softwares mentioned for some particular purposes. Please try the information provided in the zine on your risk only and we are not responsible for any damage and loss caused by that.

We are putting this work in Public Domain and you are free to use and distribute the information anyway you like, with or without any attribution. If you like the work then we encourage you to share it more and more in various forms.

This entire document was produced with the FLOSS using OpenOffice.org 3.1 on Ubuntu 9.10 64-bit Desktop edition.