

GeeksZine

Free Open Source is Fun



Issue : November 2010

Website : www.richnusgeeks.com

Publisher : RichNusGeeks Consulting

Table of Content

Just two words	3
Going with the flow	4
Is that so?	5
Banishing a ghost	10
Surprising the master	15
Searching for Buddha	17
The last rap	21



Just two words, *an editorial*

Hello floss geeks and all! Sweet November brings another issue of the zine. As per our promise earlier regarding the colorful transformation of the zine, you'll notice a change in the otherwise white background of the zine pages. This change is another small step towards making GeeksZine an content and eye candy as well. Please stay tuned for more of these kinds of colorful changes in every coming issue.

This issue of the zine introduces a new regular section that presents some news bytes of interest to the floss geeks and all. The issue has contributed sections on distributed version control system Git and Shell internals. The zine also contains writeup on a simple but versatile automation tool PyMakeMeLazyNDumb along with some cool commands in tips and tricks section.

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

Going with the flow, *news bytes*

- Microsoft has withdrawn from the development of [IronPython](#) and [IronRuby](#) and has handed over responsibility to the open source community (complete news at <http://bit.ly/cylS5S>).
- At the 6th Annual AMD Technical Forum & Exhibition (TFE) 2010, AMD showcased for its ecosystem partners the first public demonstration of the forthcoming AMD Fusion Accelerated Processing Unit (APU) codenamed “Llano”, designed for notebook, ultrathin and desktop PCs. AMD demonstrated the accelerated single-chip processing muscle of Llano by simultaneously processing three separate compute-and graphics-intensive workloads (complete news at <http://bit.ly/aA1VLq>).
- The OpenOffice.org Project has unveiled a major restructuring that separates itself from Oracle and that takes responsibility for OpenOffice away from a single company. Driving home the changes, OpenOffice.org project is now The Document Foundation while the OpenOffice.org suite has been given the temporary name of LibreOffice (complete news at <http://bit.ly/bYgEjv>).
- IBM has announced its intention to join the OpenJDK project, the official open source Java runtime effort that is led by Oracle. IBM's move to take a more active role in OpenJDK could end the company's commitment to Apache's Harmony project (complete news at <http://bit.ly/d3OigZ>).
- Out of all of Google's acquisitions, the company's acquisition of the maker of the Android mobile operating system was its "best deal ever," a company executive said at an industry conference yesterday (complete news at <http://bit.ly/a9n1Q0>).

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

Whenever we are developing random stuff or working on small or large projects we need a fair amount of control on our code to maintain the code history, important marks etc. For these kinds of things version controlling is used.

There are numerous examples of them like Mercurial, Bazaar, Subversion etc. In this article we will discuss about GIT which is a free and open source distributed version control. It was originally developed by Linus Torvalds for the source control management of Linux kernel. We would not discuss it in depth but will try use the basic commands and get a hands-on git.

Installing git : GIT can be easily installed using the following :

apt-get install git-core, my distro is ubuntu so used the above command.

You can even download precompiled packages from <http://git-scm.com/download> .

Add user to git :

```
git config --global user.name "UserName"  
git config --global user.email "EmailId"
```

Initializing git :

```
git init
```

Checking Status:

```
konarkmodi@ubuntu:~/konark-git$ git status
```

Output :

```
# On branch master  
#  
# Initial commit  
#  
# Untracked files:
```

```
# (use "git add <file>..." to include in what will be committed)

#
# testing-git
# testing-git~
```

nothing added to commit but untracked files present (use "git add" to track). This shows that we are on the branch master, committed nothing, untracked files with there names.

Adding files to git :

```
konarkmodi@ubuntu:~/konark-git$ git add .
```

'.' here mean all the files in the folder, alternate to this is writing filenames.

Now check the status :

```
konarkmodi@ubuntu:~/konark-git$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
# (use "git rm --cached <file>..." to unstage)
#
# new file: testing-git
# new file: testing-git~
#
```

you can clearly see the files which were untracked are now under changes to be committed

Ignoring files in git :

There are instances where we don't want to include some files in git. It is cumbersome to write each files name to ignore multiple files. So the easier way is to create a file .gitignore and put all the filenames into this. The file names in .gitignore will not be committed once you do this.

Committing in git :

```
konarkmodi@ubuntu:~/konark-git$ git commit
```

this opens a text editor window where in you can enter the message corresponding to the commit. Another alternative to this is :

```
konarkmodi@ubuntu:~/konark-git$ git commit -m 'Your message here'
```

Branching in git :

We can maintain multiple repositories of a single repository in git. As we saw in our example above the branch name was master that is the default branch now lets create another branch and name it test-branch :

```
konarkmodi@ubuntu:~/konark-git$ git branch test-branch
konarkmodi@ubuntu:~/konark-git$ git branch * master test-branch
```

the asterisk denote the current branch. To checkout to test-branch we would use the following command :

```
konarkmodi@ubuntu:~/konark-git$ git checkout test-branch
Switched to branch "test-branch"
```

now we can work on the test-branch. To log back to master branch we use the same :

```
konarkmodi@ubuntu:~/konark-git$ git checkout master
Switched to branch "master"
```

The above commands give you a brief idea of how to add, commit files using git. Now lets move on to the commands that help you.

Checking commit log :

There are times where you would want to check all the commits made, for that use the following command:

```
konarkmodi@ubuntu:~/konark-git$ git log
```

```
commit e690a99d8c03c85337a7395ba613d83a35358baf
```

```
Author: Konark <modi.konark@gmail.com>
```

```
Date: Sat Oct 30 09:17:30 2010 -0700
```

first commit

this shows when the commit is made with along with the message of the commit. In case of multiple commits it is showed as newest to the oldest.

To see complete list of diffs use the following command:

```
konarkmodi@ubuntu:~/konark-git$ git log -p
```

Blaming :

When we need to check which user made changes to a file we can track that in git using the following command:

```
konarkmodi@ubuntu:~/konark-git$ git blame testing-git
^e690a99 (Konark 2010-10-30 09:17:30 -0700 1) konark modi line 1
1ab028cf (Konark 2010-10-30 23:32:59 -0700 2) konark modi line 2
1ab028cf (Konark 2010-10-30 23:32:59 -0700 3)
^e690a99 (Konark 2010-10-30 09:17:30 -0700 4)
```

this shows that file two times the file was changed and both the time user was Konark. Uptil now we'd been working with a single project, now lets assume there are multiple people working on the same project.

Create a clone of the project :

To ease the understanding, lets assume there are two persons Konark and KV.

```
KV : git clone /home/konark/testing-git test
```

this creates a new directory with the name test. Now when you would edit a file and commit changes they happen on your folder. Now when Konark wants to update the files with KV's changes he would type :

```
Konark : git pull /home/kv/test master
```

this would automatically update the files. To view log etc same commands are used as were used in local folders. All the above was just a glimpse of how useful and easy is git for all of us. Below is a list of very helpful resources tips and tricks :

1. **gitk** : Visual git repository
2. **Crash course for GIT** : <http://git-scm.com/course/svn.html>
3. **github.com** : Public repository to collaborate your work online.

Konark K Modi
twitter : konarkmodi



Banishing a ghost, *transition to floss*

We talked about GNU/Linux command line basics in October 2010 issue of GeeksZine. This time we are going to dig a little deeper and find out how shell actually works? So let's get started :

What is a Shell ?

Shell is the software installed on your system that works as command-line interpreter i.e. it takes the commands you type, interpret those and hands over those to the operating system to execute. There are many different shells available to choose from in Linux.

C Shell : C shell was created by Bill Joy, while he was a student at University of California, Berkeley. Released under BSD license, the syntax of C shell is similar to C programming language hence the name C shell.

TC Shell : An improved version of C shell, provides some additional features like command line completion and editing. In most of GNU/Linux systems /bin/csh is linked to /bin/tcsh so starting of C shell instead starts tc shell.

```
$ls -l /bin/csh
```

```
lrwxrwxrwx. 1 root root 4 Oct 31 08:34 /bin/csh -> tcsh*
```

Korn Shell (ksh) : Korn shell was developed by David Korn at AT&T Bell Laboratories. It contains many features of C shell and Bourne shell.

Bourne Shell (sh) : The default shell in many Unix systems, developed by Stephen Bourne of AT&T Bell Laboratories.

GNU Bourne-Again Shell(bash) : Bash shell is the default shell in most of the GNU/Linux distributions. It's a free and open source clone of bourne shell. It was developed by Brian Fox for the GNU project. It's the most feature rich shell available and is compatible with bourne shell. To find out all the shells installed on your system type *cat /etc/shells*. Your default shell is defined in */etc/passwd* file.

```
$cat /etc/passwd | grep sumit
sumit:x:500:500:Sumit Rai:/home/sumit:/bin/bash
```

here my default shell is bash.

To switch to a different shell on your system just type the name of that shell (as in /etc/shells) and hit enter. Since bash is out default shell in most of the GNU/Linux systems so we will only discuss bash here.

How Bash Shell interprets a command?

Normally we think that when we type a command the shell looks for the command in all the directories in PATH environment variable, but when you type a command the bash shell goes through the following sequence :

1. Redirection
2. Aliases
3. Expansion
4. Shell Function
5. Shell builtin command
6. HASH table
7. PATH variable

Now let's first discuss these one by one :

1. Redirection : Consider the example given in the following snapshot :

```
[sumit@localhost tmp]$ ls
file1 file2 file3
[sumit@localhost tmp]$ ls > out.txt
[sumit@localhost tmp]$ cat out.txt
file1
file2
file3
out.txt
[sumit@localhost tmp]$
```

here files file1, file2, file3 exists in current directory as seen from the output of ls command but when we redirect the output to out.txt file, the out.txt file should contain the output of ls command i.e. file1 file2 file3 but it also includes out.txt. This happens because before ls is executed

redirection is done and file out.txt is created in current directory (we can't redirect output to a file that doesn't exist).

2. Aliases : After Redirection shell moves to aliases.

```
[sumit@localhost tmp]$ alias ls=cat
[sumit@localhost tmp]$ ls out.txt
This is out.txt file.
[sumit@localhost tmp]$ \ls out.txt
out.txt
[sumit@localhost tmp]$
```

Here I have defined *alias ls=cat*, now when I type *ls out.txt*, *ls* is replaced with *cat* and contents of *out.txt* file is displayed. To make the shell ignore alias use precede the command name with backslash (“\”).

3. Expansion : According to bash man page there are seven kind of expansions; brace expansion, tilde expansion, parameter and variable expansion, command substitution, arithmetic expansion, word splitting, and pathname expansion. Discussing all of them is beyond the scope of this article so please refer to bash man page (*man bash*) for more information. If the command you typed contain any variable name, wildcards (*, ? etc) shell expands them. e.g.

```
bash-4.1$ alias ls="/bin/ls"
bash-4.1$ function ls () { echo "I am a function" ; }
bash-4.1$ ls
file1 file2 file3
bash-4.1$ \ls
I am a function
bash-4.1$ unalias ls
bash-4.1$ ls
I am a function
bash-4.1$
```

Here *\$HOME* become */home/sumit* (my home directory) after expansion and *file** is expanded to *file1*, *file2*, and *file3*.

4. Function : Like many programming languages bash allows you to define functions. A function in bash may contain multiple commands. You can execute a function just by typing its name. e.g. look in the figure above, here I have defined an alias by name “*ls*” and a function by the same name. Since alias gets priority over function, when I first type *ls* and press enter */bin/ls* is executed. Next I have used backspace to ignore alias.

After deleting alias using unalias command, since there is no alias left ls corresponds to function ls.

5. Shell builtin commands : Some commands are a part of bash code itself, they are called shell builtin commands. They are given priority over HASH table and directories in PATH variable. e.g. echo is a shell builtin command, you can use type command to determine how a given keyboard will be interpreted.

```
bash-4.1$ type echo
echo is a shell builtin
bash-4.1$ which echo
/bin/echo
bash-4.1$ type ls
ls is a function
ls ()
{
    echo "I am a function"
}
bash-4.1$ alias ls="echo GNU/Linux"
bash-4.1$ type ls
ls is aliased to `echo GNU/Linux`
bash-4.1$ unalias -a
bash-4.1$ unset ls
bash-4.1$ type ls
ls is /bin/ls
bash-4.1$
```

Here its clear that echo is shell builtin command, echo also exists in /bin/ folder but if you type echo and hit enter it will be ignored since builtin command are given priority. Notice how the output of “type ls” changes after defining the alias by the same name. Once we delete alias and function ls, bash searched hash table, since ls is not in hash table is look in PATH and its found in /bin/ls.

6. HASH table : The concept of hash table is similar to caching in GNU/Linux, shell stores the full path of all the executed commands to speed things up. Now in the snapshot below I have started a new shell, after that I execute ls command (since there is no alias or function by the same name it executes /bin/ls because /bin is in PATH). Then after executing echo and Firefox, I have used hash command to look at hash table. The first column “hits” display cache hits and “command” column displays full path of the command. Note that since echo is internal

command, its not displayed in hash table. After I execute ls one more time its hit column changes from 1 to 2.

```
bash-4.1$ bash
bash-4.1$ ls
file1 file2 file3
bash-4.1$ echo "GNU/Linux"
GNU/Linux
bash-4.1$ firefox
bash-4.1$ hash
hits    command
  1     /usr/bin/firefox
  1     /bin/ls
bash-4.1$ ls
file1 file2 file3
bash-4.1$ hash
hits    command
  1     /usr/bin/firefox
  2     /bin/ls
bash-4.1$
```

7. PATH environment variable : Last of all shell searches for a given command in directories listed in PATH environment variable. You can look at the contents of PATH by typing :

`$echo $PATH`

to add any directory to PATH variable use

`$export PATH=$PATH:/path/to/dir,`

where /path/to/dir is the absolute path to the directory you want to add. For more information you can read bash man page (`$man bash`).

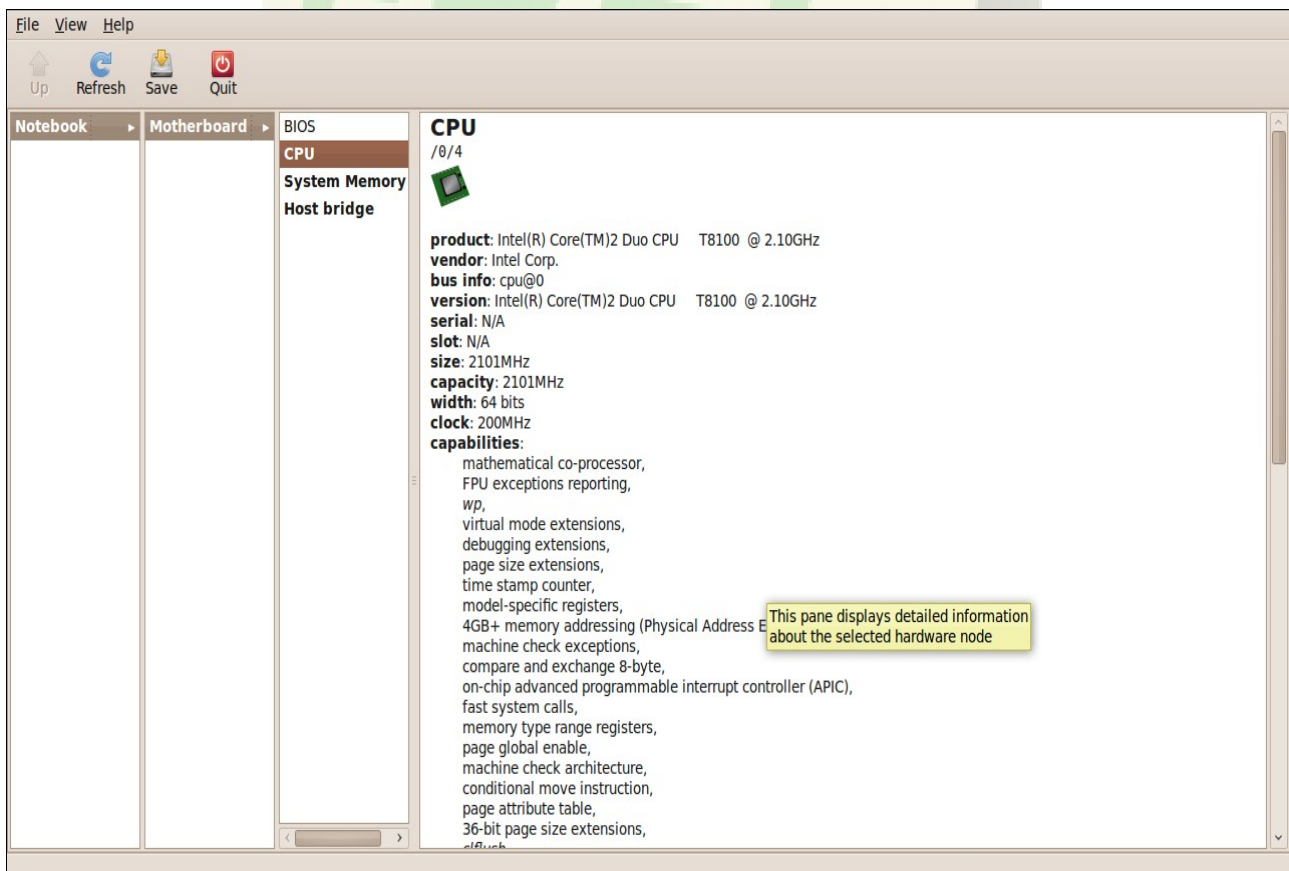
Sumit Rai

sumit.tux@gmail.com

Surprising the master, *cool tricks*

- There is a small utility available for GNU/Linux systems known as *lshw* (hardware lister) that detects and provides information about the hardware configuration of a machine. This utility is available in different formats through its web page <http://ezix.org/project/wiki/HardwareLiSter> and it comes preinstalled with Ubuntu. There is also a GUI front end to the *lshw* and that can be installed by command `sudo apt-get install lshw-gtk`.

To detect your hardware configuration, type *lshw* in a text console and it presents a detailed information about your hardware configuration. If you run it with `sudo` or `su` privileges then you'll get more detailed and correct hardware info. You could also format the output of the utility into a nice html page through command `lshw -html > filename.html`. Run `lshw -X` to launch the GUI frontend of *lshw*. Try to explore more through `man lshw` and enjoy using it. The screenshot of its frontend is shown below :



- Most of the *IX (Unix, GNU/Linux etc.) users need some help regarding the description of various commands, apis etc. without knowing the exact command or API name to search. You can use command *apropos* to search for all the related information corresponding to a search term. It provides a list of various command and APIs where it finds the search term. So next time, when you have only few term in your head to search for something then give this command a shot. An example screenshot of *apropos* is shown below :

```
richnusgeeks@ankur-laptop:~/Development$ apropos stack
curl_multi_info_read (3) - read multi stack informationals
pcrestack (3)           - Perl-compatible regular expressions
bluetooth-properties (1) - GTK dialog for managing properties of the Linu...
bluetooth-wizard (1)   - GTK wizard for setting up devices with the Linux B...
hciattach (8)          - attach serial devices via UART HCI to BlueZ stack
longjmp (3)            - non-local jump to a saved stack context
pam_debug (8)          - PAM module to debug the PAM stack
pthread_attr_getstack (3) - set/get stack attributes in thread attributes...
pthread_attr_getstack (3posix) - get and set stack attributes
pthread_attr_getstackaddr (3) - set/get stack address attribute in thread...
pthread_attr_getstackaddr (3posix) - get and set the stackaddr attribute
pthread_attr_getstacksize (3) - set/get stack size attribute in thread at...
pthread_attr_getstacksize (3posix) - get and set the stacksize attribute
pthread_attr_setstack (3) - set/get stack attributes in thread attributes...
pthread_attr_setstack (3posix) - get and set stack attributes
pthread_attr_setstackaddr (3) - set/get stack address attribute in thread...
pthread_attr_setstackaddr (3posix) - get and set the stackaddr attribute
pthread_attr_setstacksize (3) - set/get stack size attribute in thread at...
```

Searching for Buddha, *thoughts about floss technologies*

Good computer users and programmers are lazy and dumb. Lazy because they believe in automation as repetition is boring for them and dumb because they are not ever satisfied with what others try to tell them. There is an upcoming automation tool in offering for all the lazy and dumb geeks known as PyMakeMeLazyNDumb. The aim of this tool is to provide all those lazy and dumb people a simple but super automation tool through which they become more lazy.

The PyMakeMeLazyNDumb tool is a combination of various simple but powerful python applications that automate some important repetitive activities encountered by the geeks on day to day basis. The release pattern for the PyMakeMeLazyNDumb shall be in form of the release of few of these python applications at a time. These tools are developed in standard python so these are cross platform and the users don't need to install any external packages to use these applications. The applications are released under GNU GPL v3 and current version of these tools is 0.0.2 . You can download and try these tools from :

<http://code.google.com/p/pymakemelazyndumb>
<http://pymakemelazyndu.sourceforge.net/>

These tools are completely free and open-source and you can modify those according to your needs. Currently there are two applications released, pygenericroutines.py and pycreateflswthdr.py . There are many more tools coming up soon and please contact the owner at the project pages if you want to contribute to this project. Now let's explore these python applications.

Pygenericroutines.py

There are some very common activities/operations in our computing life we all wanna automate on a daily basis e.g. backing up files, testing the existence of files, creation of nonexistent directories, testing status of Internet in a machine, parsing command line, parsing configuration files, setting up logging etc. These routines are more important if you want to

create some automation tools but writing them again and again is like a burden to the programmers. The `pygenericroutines.py` is a python module that provides these kind of common methods you require for building automation tools and many other things.

We clear it through an example, suppose you need the command line parsing in your python applications. You can write it first time, second time but what about writing it for 10 -15 times? Isn't it the wastage of time and efforts? So here comes the `pygenericroutines.py` to the rescue. All you have to do is just import `PyGenericRoutines` class from `pygenericroutines` module and use the desired methods in your program. It's like making use of libraries in other programming languages like C, C++ etc. Output wise you won't see any difference but modularity and programming efforts wise you will feel a significant difference. You just have to add the following line in your code (preferred at the top) to make use of the methods available :

```
from pygenericroutines import PyGenericRoutines
```

all the methods provided by `PyGenericRoutines` class return `False` in case of failure and `True` in case of success. The `pycreateflswthdr.py` application described next is an live example of this. In fact, all the applications making `PyMakeMeLazyNDumb` are heavily using these generic methods. You can take a look at their code and you will realize how easy it is to use the generic methods.

Pycreateflswthdr.py

Good programmer always knows how to write the code but great programmers knows how to write code they and others can track and understand later on too. So as a good software engineering practice, the first step before you start writing an application is to put an information header in your files. This information header is necessary to track and give others a chance to understand your code more easily. Its like a brief overview of your code. I mean something like this (taken from `pygenericroutines.py` and initially generated through `pycreateflswthdr.py` only) by information header :

```
#####
# File name : pygenericroutines.py
# Purpose : A Python class to expose some generic routines for other
#           python automation tools.
# Usages : add "from pygenericroutines import PyGenericRoutines" in the
#           python file(s) and use class.
# Start date : 13/10/2010
# End date : 13/10/2010
# Author : Ankur Kumar Sharma <richnusgeeks@gmail.com>
# Download link : http://www.richnusgeeks.com
# License : GNU GPL v3 http://www.gnu.org/licenses/gpl.html
# Version : 0.0.2
# Modification history :
#####
```

This header almost provides you with every information regarding the code of your application. Purpose, usage, dates all at one place. Suppose there are thousands of files at your place and you want to locate a particular one then this can help you a lot. Just by searching for any field in the information header of the desired code you can locate it quickly. Also any other person can quickly get the overview of your work just by browsing through the information header.

Pycreateflswthdr.py provides you the facilities to create one or multiple new files with the information header like shown above on the top of your file. It can also insert the information headers at the top of the already existing files. So whenever you want to start your project just type (you need to put both pygenericroutines.py and pycreateflswthdr.py in the same location) :

```
python pycreateflswthdr.py file1.ext1 file2.ext2 file3.ext3 ....
```

and headers will be there on the top. You can control the directory where the files are created using *-d* option along with the path of the desired directory. To append the information header to the existing files there is *-a* option. You can see the application help using *-h* option. The application creates a back up with *.bak.timestamp* extension if you apply this on your

existing files so you don't have to worry about losing your existing work. In case of any error or warning messages, the messages are appended also to a log file activity.log in the working directory.

If you use the applications to create new files then it will create the files automatically. Currently this application creates and appends to various files having .c, .h, .cpp, .hpp, .java, .py, .pl, .rb, .lua, .php, .sh, .mak, .sql extensions with appropriate commented information headers. So use this simple but effective application next time when you want to generate files you or others could track and understand quickly later on. Give PyMakeMeLazyNDumb tool a try and save your time to hack and enjoy more with FLOSS.

Ankur Aggarwal

ankur.aggarwal2390@gmail.com

coolankur2006@gmail.com

www.twitter.com/ankurtwi



FREE

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 releasing all the sections of the zine in Attribution-ShareAlike license <http://creativecommons.org/licenses/by-sa/3.0/> except News Bytes and Cool Tricks and you are free to use and distribute the information with proper 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.

