LINUX Command Line

FAST AND EASY

Matthew Gimson

LINUX COMMANDS, BASH SCRIPTING TRICKS, LINUX SHELL PROGRAMMING TIPS AND BASH ONE-LINERS

Linux Command Line:

FAST and EASY!

-Linux Commands, Bash Scripting Tricks, Linux Shell Programming Tips and Bash One-Liners

By Matthew Gimson

Table of Contents

Disclaimer

While all attempts have been made to verify the information provided in this book, the author does assume any responsibility for errors, omissions, or contrary interpretations of the subject matter contained within. **The information provided in this book is for educational and entertainment purposes only. The reader is responsible for his or her own actions and the author does not accept any responsibilities for any liabilities or damages, real or perceived, resulting from the use of this information.**

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Introduction

Linux is a common operating system for computers, and especially the server computers. The funny thing with it is that people view Linux as a very complex operating system which is not the case. This is why most computer novice users don’t like using Linux on their computers. However, this is not the case. Just like the Windows OS, Linux is an easy-to-use operating system. All one needs to have is interest in learning it. The operating system itself is very secure, meaning that hackers will find it almost impossible to hack into it.

The various distributions of Linux use almost the same commands. These commands can be written using the same syntax and they run successfully on various distributions of Linux. However, some of them will need to be altered slightly so that they can be compatible. You should identify the right distribution for yourself and your computer and install it on your system. The right choice for you in this case will depend on the distribution that you will feel comfortable to work with. Other than installing the operating system directly onto your hard disk, you can choose to run it on virtual box or VMware player. This can help in saving on the available computer space, hence important.

Chapter 1- Definition

Linux is an open-source operating system for computers. It was developed to provide a Unix-like free and open source operating system for computers. The kernel is the main component of Linux operating system and its first version was released in the year 1991 by Linus Torvalds. People rarely use Linux operating system on desktop computers. This operating system has shown a wide use on server computers. The reason is due to its high level of security. For instance, those who have worked with server computers running Linux, you must have realized that the operating system supports no graphics.

After booting up the computer, what you are provided with is just a blinking cursor. You then have to provide commands to the terminal. This is of great advantage when it comes to ensuring security of the system. Note that graphics and any other additional feature added to the operating system are an advantage to the hacker as they can gain access into the system via this feature. If you don’t the commands used on the command line, then definitely you be stuck. However, the desktop versions of Linux supports graphics.

There are various distributions of Linux, commonly known as Linux Distros. These include Ubuntu, Khali, Red Hat, Fedora, Mint, Centos, SUSE and others. These Distros exhibit numerous similarities although there are fewer differences amongst themselves. This involves even the commands used on their terminals. For enterprise distributions of Linux such as Red Hat, a subscription fee must be paid for you to enjoy the services that they offer. This includes even updating of the OS.

The development of Linux began in the year 1983 when Richard Stallman thought of the idea of developing a completely free Unix-like operating system. This also marked the beginning of the Gnu project. By the year 1990, the essential components of this OS such as the compilers, libraries, shell and the text editors were created. Other complex components of this operating system such as drivers for devices, kernel and the daemons were developed later.

The development of the Linux Kernel began in the year 1991. This was after Richard Stallman attended University of Helsinki. The user interface of Linux operating systems supports either graphics or command line. In desktop versions, both the graphical user interface and the command line are supported. The GUI is made the default although you can still access the command line.

Chapter 2- Linux Command Line Commands

Let us explore the Linux commands ranging from the basic ones which are necessary for you to begin to the complex ones. To open the command line, right click on your Desktop and select "*Open Terminal*". If this is not available, find Applications at the top of your desktop. After clicking on this, select *Accessories*. Choose "*Open terminal*". Open a text editor of your choice such as *Gedit* or *Vim*. To open the *Gedit* text editor, just search for it from the search bar. Type "*text editor*" on this search bar and you will find it. To open *Vim*, commonly known as *vi editor*, on the command line, type the command:

vi filename

where *filename* is the name of the file. I have called my file *myfile*. On pressing the enter key, the *vi editor* will be opened. Now you can add some text of your choice to the file. Note that to change from command mode in *vi editor*, you have to press the letter *i*, otherwise, you will not be able to add any text to the file.

The text should be as follows:

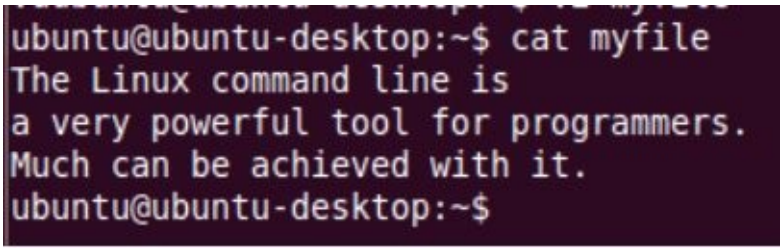
The Linux command line is
a very powerful tool for programmers.
Much can be achieved with it.

You can then save the file to somewhere where you can easily access such as on the desktop. In *vi editor*, just press the *Esc* key followed by typing *wq*. This will save the file.

On the command line, type the following command and then press the enter key:

cat myfile

On pressing the enter key, the text you added on your file will be displayed on the standard output. This is illustrated in the figure below:



This shows that the *cat* command displays the contents of a file on the standard output. However, use it to display the contents of a short file.

Changing Directories

Sometimes, you might need to change your working directory. This can be achieved via the terminal using the *cd* (change directory) command as shown below:

cd /home

The command above will change the working directory to home directory and relative to root due to the use of the forward slash, that is, /. Regardless of the current working directory, the above command will be executed. Type the following command:

cd httpd

After pressing the enter key, the full working directory will become */home/httpd*, meaning that we have changed to the *httpd* directory but relative to the */home* directory. To change back to the user's home directory, use the following command:

cd ~

The user's home directory is the /home/username directory, where the username will be the name you have used on the computer. Notice that we have used the symbol ~, known as the tilde symbol. In Linux, it symbolizes the user's home directory.

Copying file

In Linux, the command *cp* (copy files), is used for copying files. In copying, a duplicate of the original file is made at the location being specified, meaning that the initial file is not deleted. To make a duplicate a duplicate of the file *myfile*, run the following command:

cp myfile myfile2

The above command will create a duplicate of the file *myfile* and name it *myfile2*. The files will be similar but with a difference in their naming. However, it is risky to run the above command. This is because in case the file *myfile2* already exists in the directory, it will be overwritten without a warning. To take care of this, the *–i* option should be used as shown below:

cp –i myfile myfile2

If the file *myfile2* already exists in the directory, then we warned before it is overwritten, which is a very good idea.

You might need to copy all files contained in a certain directory to another directory. This can be done using the following command:

cp –dpr originaldirectory finaldirectory

The files will then be copied from the *originaldirectory* to the *finaldirectory.* They will not be deleted from the latter. The *–dpr* option is very important as it plays the following role:

-d- for preservation of links.

-p- for preservation of file attributes.

-r- for recursive copying.

If you don’t specify the above options, the default ones will be applied. This is not recommended as you might need to preserve the links and file attributes which might not be done with the default options.

Each mounted file system has used some space on the disk. To know the amount of this space for each file system, use the command:

df

The *less* command

This command is almost similar to the "*more*" command with the difference in that, with this command one will be able to move the page up and down through the file. Consider the command below:

less myfile

After pressing the enter key, the contents of the file *myfile* will be displayed.

ln command

To create a symbolic link to your file, use the command *ln*. Type the command shown below:

ln –s myfile mlink

The above command will create a symbolic link called *mlink* to link to the file *myfile*. To show that two files are different and contain different inodes, use the command

ln –i myfile mlink

locate command

If you need to search in a database, use the *locate* command. Consider the command shown below:

slocate –u

The command will create a database and name it *slocate*. The problem is that the process might take a long time so patience will be required. But the command is important since you can't search for a file without having run it. Ever heard of cron? This is used for scheduling tasks. This will run the above command periodically so you might not be needed to do so. Type the command shown below and run it:

locate whois

The command above will search for all the files contained in your system and whose names contain the string "whois".

logout command

This command just logs out the currently logged in user. Open the terminal and type the following:

logout

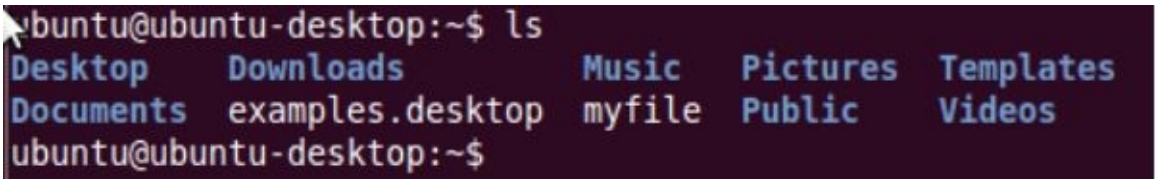
Once you have typed the above command, just press the enter key. You will notice that you will be logged out of the system and then finally you will be on the login screen.

ls command

Use this command to list the files which are contained in a particular directory, *ls* means list. The command has many options associated with it, which needs to be well understood. Type the command shown below and the press the enter key:

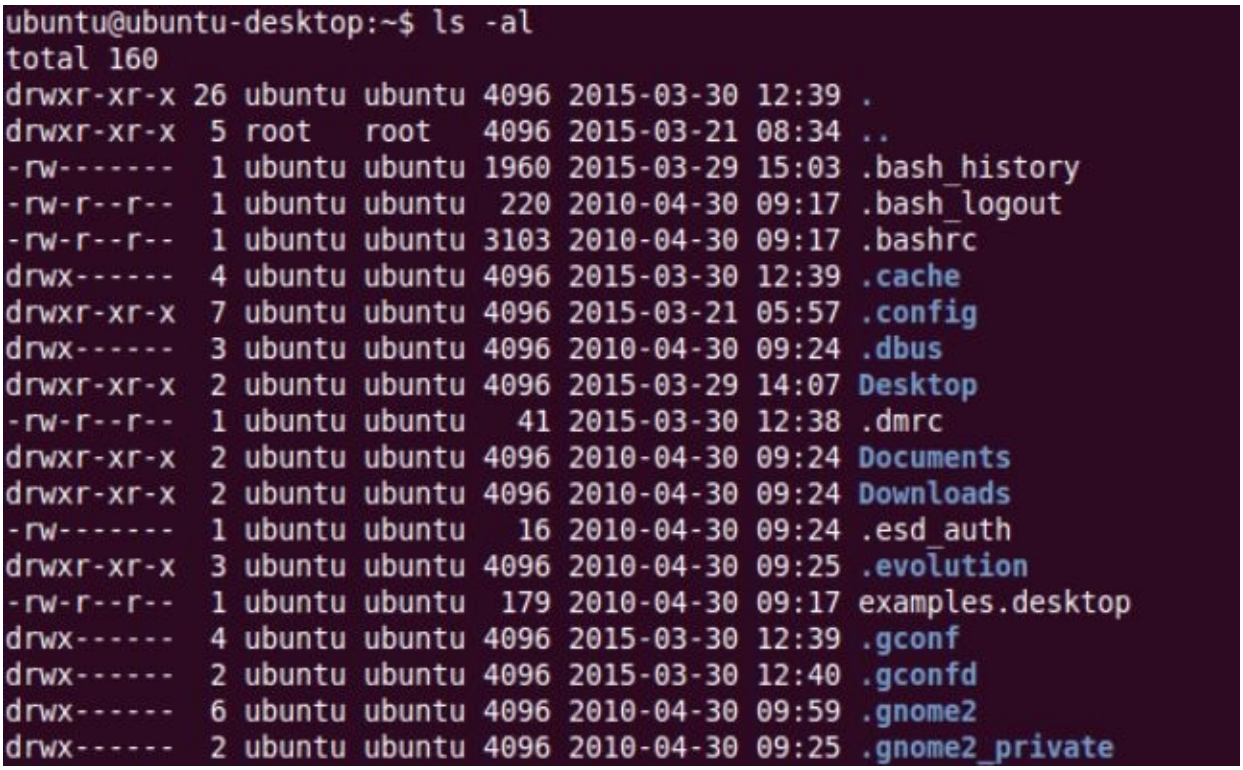
ls

The command will list all the files which are contained in your current directory.



The above figure shows the files contained in my directory. However, if this directory contains files whose name starts with a dot (.), they will not be listed with this command. You also notice that with the above command, only a few details about the file are provided. Suppose you want to know the size of the file, the day it was made and the ownership permissions associated with it, you can achieve this with the following command:

ls –al



As you can see in the figure above, the output is more detailed. The first part starts with either a “d” or a – (dash), where “d” means that it is a directory, while “-“ means that it is a file. The “*rwx*” stands for permissions, where *r* is for read permission, *w* is for write permission whereas *x* means execute permission. The size of the file and the date of creation are also shown.

more command

With this command, the contents of a file will be sent to the screen. This will be done one page at a time. The command is also applicable on piped output.

Type and run the command shown below:

more /etc/profile

The above command will output the contents of the file */etc/profile*. This file contains details about the users of the system. On pressing the enter key, the following will be the output:

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

if [ "$PS1" ]; then
  if [ "$BASH" ]; then
    PS1='\u@\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
        . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
```

Note that this is a built-in file so it comes with the OS itself. The file also contains too much content which cannot fit on a single page. With the above command, these will be displayed one page at a time. Type the command below and run it:

ls –al | more

The output will be the files contained in that directory. This output will then be piped through the more command. Since the output is large, it will be listed one page at a time.

mv command

The command *mv* is used for moving or renaming files, *mv* stands for move. Type the command shown below:

mv –i myfile file

The command will rename the file *myfile* to file, meaning that it will have been moved. This is the simplest way on how to rename files in Linux. Consider the following command:

mv /directory1/myfile

The above command will move the file *myfile* from the directory "*directory1"* to the current working directory. This is very simple and it shows the power of the *mv* command.

The question is, what is working directory? This is the directory you are currently working in. To know the directory you are working in, use the following command:

pwd

Just press the enter key as usual. The output will be the directory you are working on. This command is used for that purpose in case you don't know the directory you are in. *pwd* means print working directory.

shutdown command

Use this command whenever you want to shut down your system. Run the command shown below:

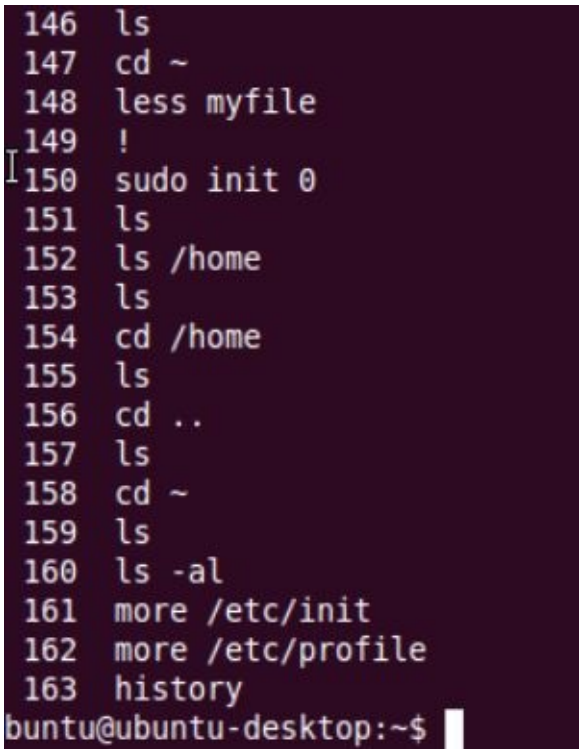
shutdown –h now

On running the above command, you will notice that the system will halt immediately, that is, it will shut down. Consider the next command:

shutdown –r now

The above command shuts down the system and then boots it again, meaning that it is used to reboot the system.

You might need to check the commands which you have run previously. The reason might be that you have changed something and now you want to undo the change. To see the list of these commands, run the history command as shown in the figure below:



The above figure shows the commands which I have run recently.

sudo command

sudo stands for *super user do*. With this command, any user can execute his or her commands as a super user. The *sudoers* file defines all this. Sensitive commands, especially those which alter the file systems of the Linux OS can only be executed using this command. The *su* command enables you to login as a super user while the *sudo* command borrows the privileges of a super user.

To update the system, you must use the *sudo* command as shown below:

sudo apt-get update

If this is the first time for you to run the command, you will be prompted to enter the *sudo* password. To run the above command, you must be connected to the internet, otherwise, you will get an error. In Linux, the updates are fetched from what we call repositories. This is where the Linux development team uploads any latest updates regarding to various ditros of Linux.

To upgrade the system, run the following command:

sudo apt-get upgrade

The above command will upgrade the version of OS that you are using on your system. Again, you must be connected to the internet. Notice that in Red Hat Linux, this command is substituted by the *yum* command, so the *sudo* command is not supported in Red Hat Linux.

***mkdir* command**

mkdir stands for *make directory*. It is used for creating new directory. If the folder representing the directory is in existence, you will get an error informing of the same. Let me create a directory and call it *myfiles*.

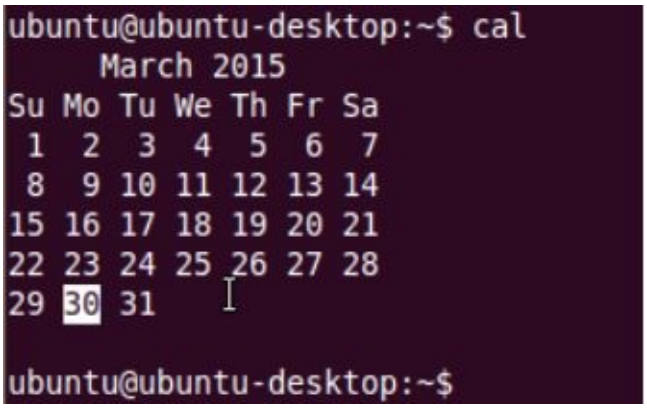
mkdir myfiles

The above command shows that I will create a directory named *myfiles*. This can be illustrated in the figure shown below:

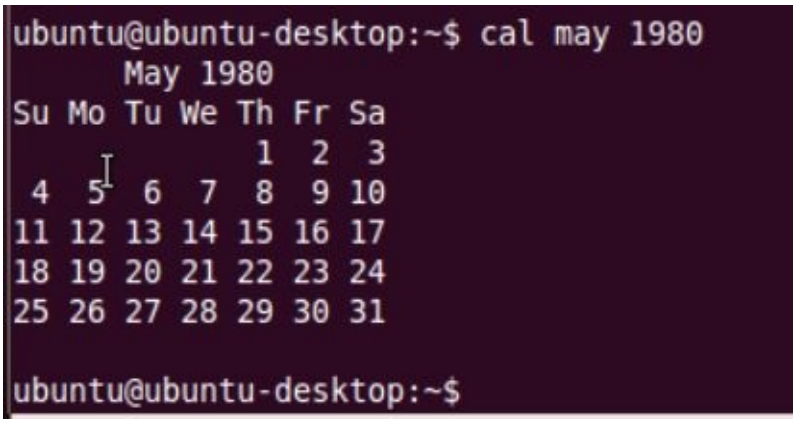
Notice after creating the directory then I use the *list* command, the directory will be listed in the available directories. This shows that it has been created. If the above command fails to run, precede it by the *sudo* command and provide for the password if prompted to do so.

cal command

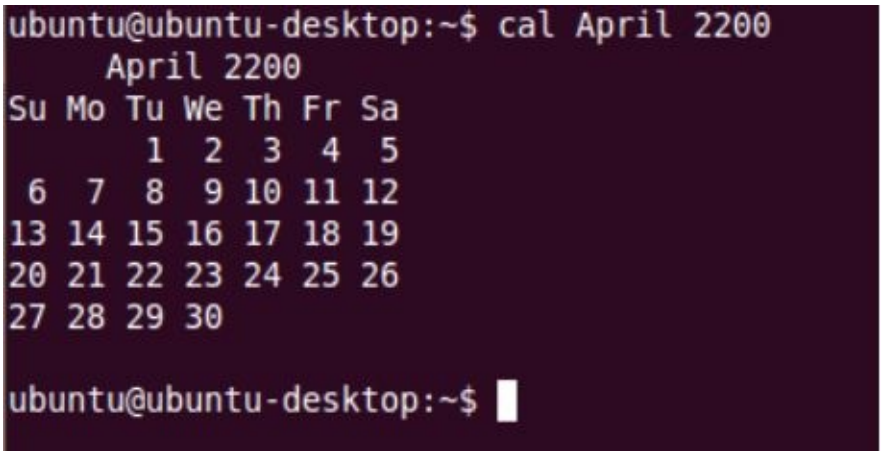
This command shows the month of a particular year. This year can be the current one, the past or even the advancing one. Just open the command *prompt* and type the command *cal*. Press the enter key:



As seen, it shows the date of the day, and it is correct. Now, let us show the month May of the year 1980, which is a past year:

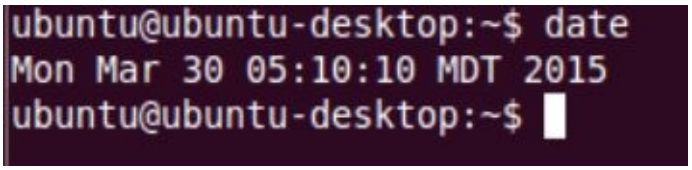


You can also show the month a future year as shown below:


ubuntu@ubuntu-desktop:~$ cal April 2200
April 2200
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
ubuntu@ubuntu-desktop:~$

date command

In case you need to display the current time and date, use this command as shown below:



On executing the *date* command, the current date and the time will form the output as shown in the figure above. However, you might find that the date on your system is wrong.

It is possible to set it via the command line as shown below:

date –set=’1 April 2015 19:30’

Once you have run the command shown above, the date will be set to *1st April 2015* and the time will be set to *19:50 hours*.

grep command

Sometimes, you might need to search for a pattern or a string from a certain file. In most cases, the various Distros of Linux come installed with this command.

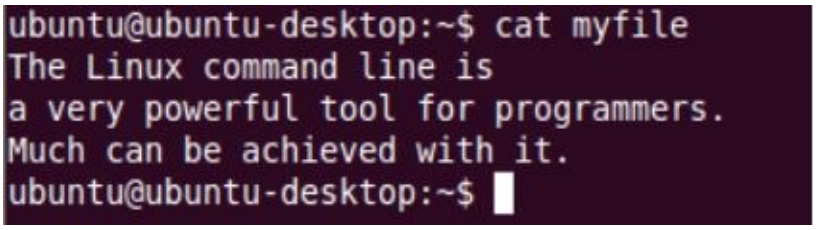
However, it is also possible to install it in case you find it not installed using the following command:

```
sudo apt-get install grep
```

If you are using Red Hat, install it using the command:

```
yum install grep
```

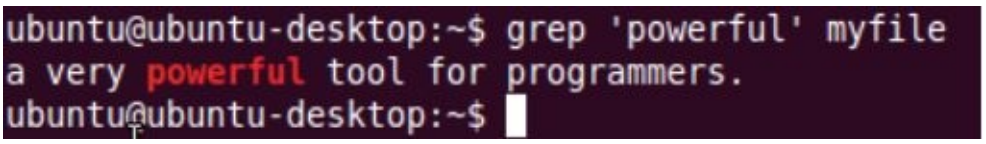
Remember that we created our file and named it *myfile* with the following content:



We want to search through it using the *grep* command. To search for the word *powerful* in the file, the following approach should be used:

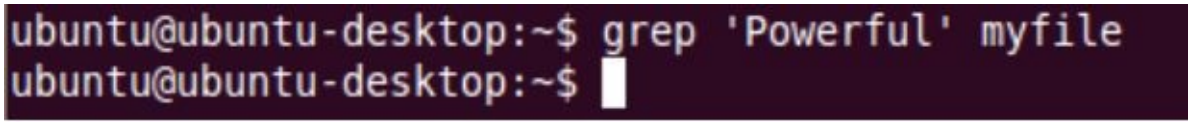
grep ‘powerful’ myfile

This is illustrated using the figure shown below:



The command outputs the line of the file with the search word and makes the search word red in color. This differentiates it from the rest of the words. Notice the search is case sensitive.

Let us search for the word ‘*Powerful’* rather than ‘*powerful’:*



As shown in the above output, there exists no word ‘*Powerful*’ in the file. This is because we have made the *p* uppercase. However, it is possible to make the search case insensitive. This can be achieved using the *–i* option as shown below:

```
ubuntu@ubuntu-desktop:~$ grep -i 'Powerful' myfile
a very powerful tool for programmers.
ubuntu@ubuntu-desktop:~$
```

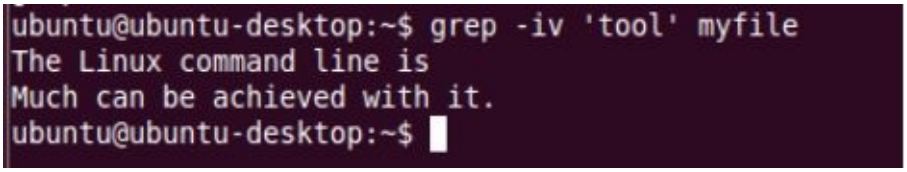
The –i makes the words '*powerful*' and '*Powerful*' the same, hence we will get our result.

To perform a multiple search, which means that you will search for several words in a file at once, we can still use the *grep* command. This is demonstrated below.

```
ubuntu@ubuntu-desktop:~$ grep -ie 'command' -e 'powerful' myfile
The Linux command line is
a very powerful tool for programmers.
ubuntu@ubuntu-desktop:~$
```

Notice that in the figure above, we are searching for two words, that is, *command* and *powerfu*l. The *–e* option makes it possible for us to search for more than one word at once. Don't confuse the purpose of the *–i* option as it only makes the search case insensitive.

The *grep* command can also be used to mean the opposite of your specification. This is illustrated below:



From the figure shown above, we have searched using the word '*tool*' as our search criteria. The command will only return the lines without the word '*tool*'. This explains why the second line of the file is not part of the output. The result has been achieved using the *–v* option, which returns the opposite of the search criteria. The *–i* is for case insensitive search, so let it not confuse you.

To know the number of the line of the file containing the word '*powerful*', use the command shown below:

grep –in 'powerful' myfile

```
ubuntu@ubuntu-desktop:~$ grep -in 'powerful' myfile
2:a very powerful tool for programmers.
ubuntu@ubuntu-desktop:~$
```

Consider the command shown below and its output:

grep –iB1 'can' myfile

The command above outputs the following:

```
ubuntu@ubuntu-desktop:~$ grep -iB1 'can' myfile
a very powerful tool for programmers.
Much can be achieved with it.
ubuntu@ubuntu-desktop:~$
```

The command outputs the exactly one line which is above the line with our search criteria '*can*'. This is because we have used *1*. If we used two, the output would be as follows:

```
ubuntu@ubuntu-desktop:~$ grep -iB2 'can' myfile
The Linux command line is
a very powerful tool for programmers.
Much can be achieved with it.
ubuntu@ubuntu-desktop:~$
```

tail command

A very popular command in Linux which is used to give the last part of a file. It gives only the last 10 lines of the file that you specify. If you need to get the last 10 lines of the file *myfile*, use the following command:

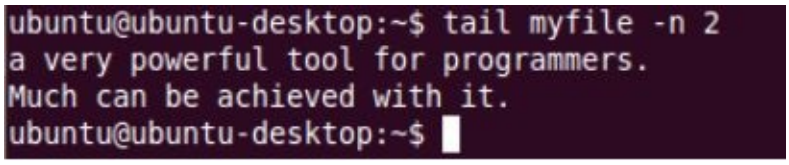
tail myfile

However, we have added only three lines to the file. This means that the result of the above command will be the three lines of the file. If it had more than 10 lines, then it would output only the last 10 lines. You should also specify the extension of the file name. A good example is if it is a *c* file, add a *.c* extension to the file name.

However, the last 10 lines of the file is the default setting of the *tail* command. One has the choice of specifying the number of lines that they want to form the output.

Consider the command below and its output:

tail myfile –n 2



As shown, we have specified that we want to output only the last 2 lines of the file *myfile*. This is what forms our output. The *–n* option when used with this command specifies the number of lines that will form the output.

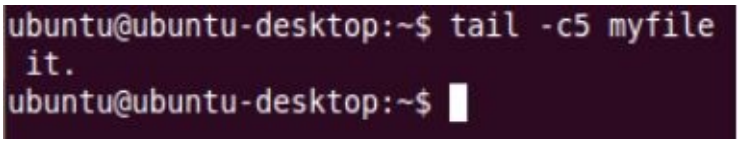
Consider the command shown below:

tail -f myfile | grep 24.13.152.12

Notice that we are piping the output from the *tail* command into the *grep* command. The command above can be used to monitor updates being made on the file *myfile* in real time, that is, just as they happen. The last 10 lines of the file and any new lines added to it will be piped to the *grep* command. The *grep* command then has the task of outputting these on the standard window. Note that only the lines with the above specified address, that is, *24.13.152.12* will be printed on the standard output.

The command can be used to display the last part of the file in relation to its size rather than the number of lines. The size is usually specified in bytes. To print the last 5 bytes of the file *myfile*, use the following command:

tail –c5 myfile



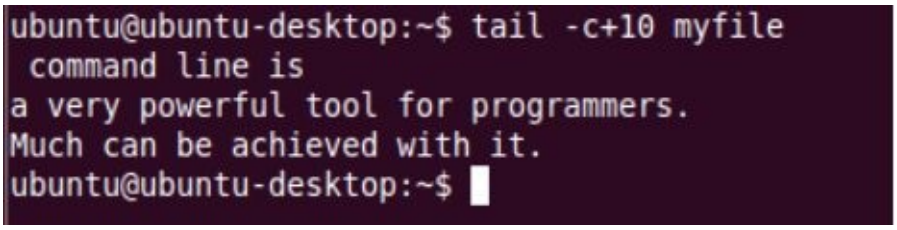
The output implies that only the word "*it*" and the *full stop (.)* form the last 5 bytes of the file. It is also possible to combine the above option with the *plus (+)* symbol to print the bytes from a certain byte.

Consider the command shown below:

tail –c+10 first

The above will print the last bytes of the file, starting from the 10th byte. The output will

be as shown below:

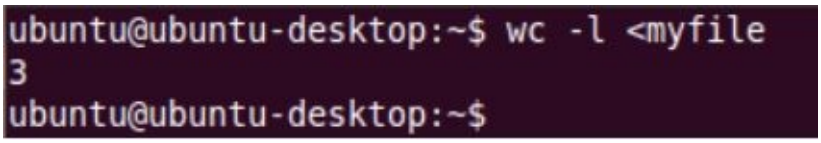


wc command

wc stands for *wall count*. This command is used to count the lines contained in a certain file. To know the number of lines contained in the file *myfile*, run the following command:

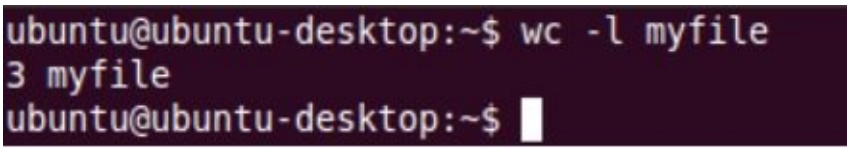
wc –l <myfile

The command outputs the following:



The command gives the number of lines contained in the file *myfile*, which are only three lines. In case you also need to get the name of the file, use the following command:

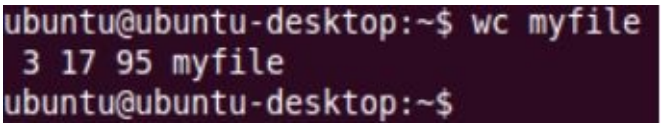
wc –l myfile



Run the command shown below:

wc myfile

The following will form the output from the file:



The command will show the number of lines in the file, the number of words contained in the file, the size of the file in bytes and the name of the file. These have been displayed in the order mentioned, where 3 is the number of lines in the file, 17 is the total number of words in the file, 95 is the size of the file in bytes and finally we have the file name. However, some people might easily be confused by the above.

It becomes easy when you split the above output as shown below:

wc –c myfile– this will output the total bytes making up the file.

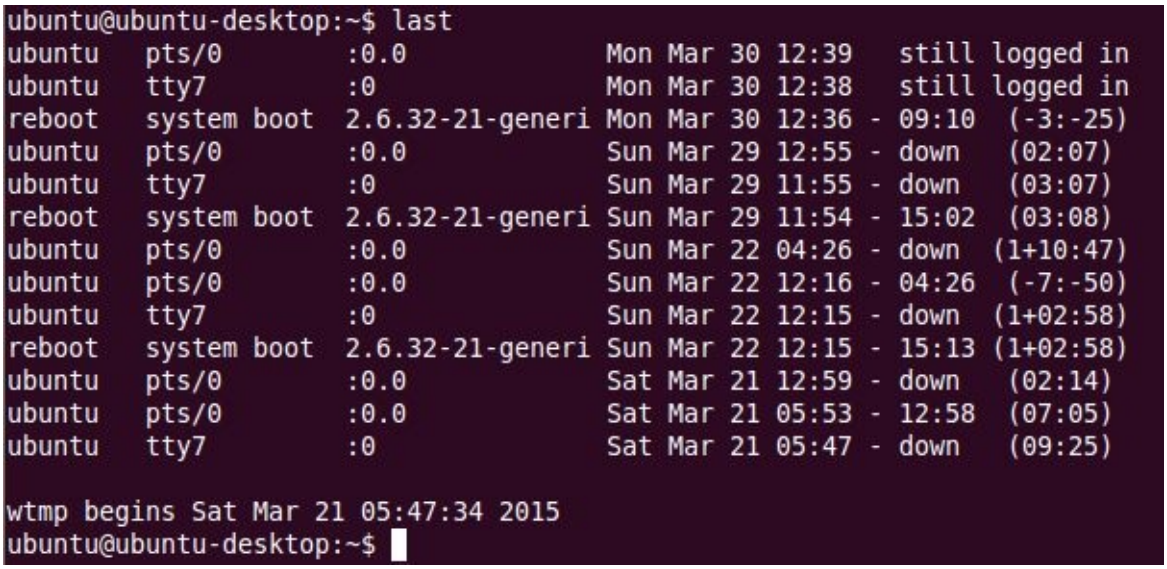
wc –w myfile– outputs the number of words contained in the file.

wc –m myfile– will output the number of characters making up the file.

If you can’t memorize the order used in the first command, then it is advisable you use the simple and alternative option above.

last command

This command is useful when you want to know the users who have recently logged into your system. This is very useful in the industry for security purposes. A user won't deny having logged into the system at a certain time. Open the command line and type in the command *last*. Observe the output:



The command shows the name of the user and other details including the terminal they were logged on and the time in which they logged in. In case any of the users is still in, you will be notified of it. The difference between when the user was lastly logged in and logged out will also be shown. Notice that only the last 100 logins will be displayed.

The command is also associated with the following options:

1. **–R**- this will make sure that the host name is not displayed.
2. **–d** – this is used when doing remote login. The *ip* address which identifies the host name is translated into the hostname itself.
3. **–a** – this will display the name of the host in output's last column.
4. **–x**- shows the shutdown history and how changes have been made on the run levels.
5. **–i**- displays the *ip* address of the host login in remotely. The *ip* address is usually in dots and number format.
6. **–F**- the times and dates of full logins and logouts into the system ware displayed.
7. **–w**- the names of the user and the domain name are displayed as the output.

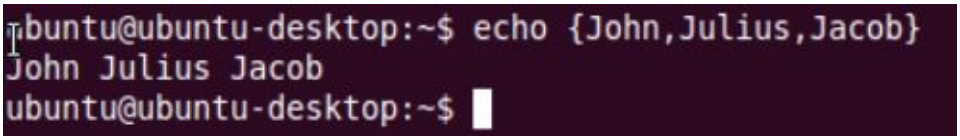
Chapter 3- Bash scripting Tricks

Bash stands for *Bourne Again shell*, and in most Linux distros, it forms the default shell. Many Unix and Linux users really like it due to its user-friendliness. In this chapter, we are going to learn about the Bash shell.

Brace Expansion

A list of strings is used as arguments in this case. These strings are separated using comas. There should be no space after the comma. Example:

echo {John,Julius,Jacob}



From the figure, you can see the output. Just the arguments we provided in the braces.

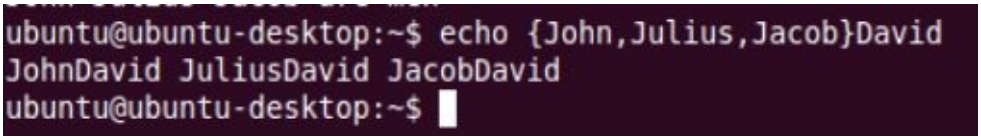
Now, type the following on the command line:

echo John,Julius,Jacob

The output will just be the same as in the first example above. I am sure that you are seeing no usefulness of the braces. Consider the example shown below:

echo {John,Julius,Jacob}David

Maybe the three are sons of David. Observe the output:



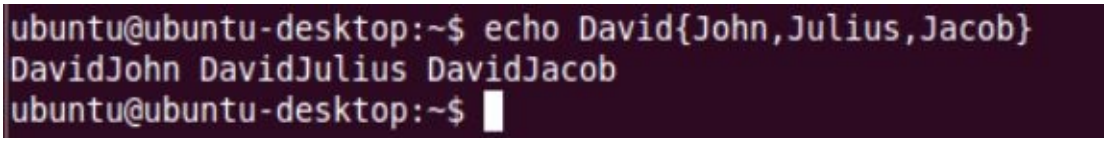
You have now seeing how useful the braces are in bash scripting. They are very useful

when the list placed in the braces should occur after, inside or before another string. In the case above, these should occur before the string David.

The above can be implemented in another way as follows:

echo David{John,Julius,Jacob}

The following will form the output of the above script:

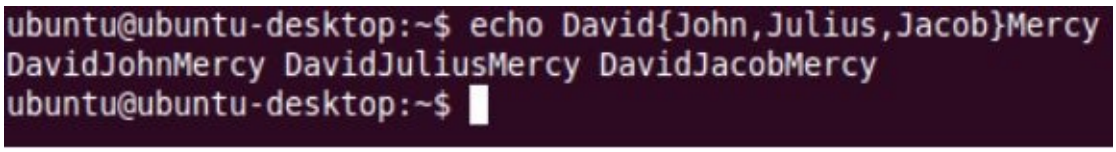


As you can see, the list specified inside the braces occurs after the string “David”.

It is also possible to place the list in between two strings. This is illustrated below:

echo David{John,Julius,Jacob}Mercy

The following will be the output:



The words in the list have been sandwiched between the two strings that we have specified. Again, make sure that you don’t include space at all. There are no spaces between the arguments nor between the braces and the strings. Including this might lead to an error or an undesired output.

However, if you need to use a space, make use that you also use double quotes as shown

below:

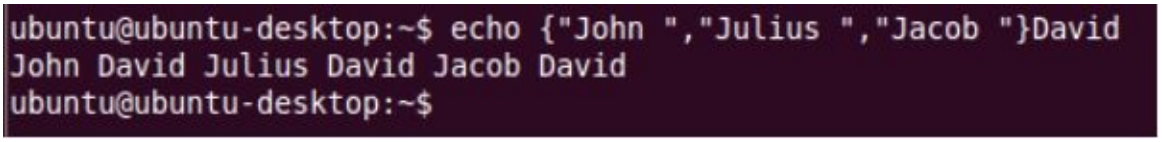
echo “{John,Julius,Jacob} David”

The following will be the output of the above:

You might need such like output, so that’s how to get it. Consider the example given below:

echo {“John “,“Julius “,“Jacob “} David

The above will give the following as the output:



Braces can also be nested, whereby braces are placed within other braces as shown below:

echo {{a,b,c},a,b,c}

The following will be the output of the above example:

```
ubuntu@ubuntu-desktop:~$ echo {{a,b,c},a,b,c}
a b c a b c
ubuntu@ubuntu-desktop:~$
```

Consider the example shown below:

echo {{6,7,8}1,2,3)

The output will be as follows:

```
ubuntu@ubuntu-desktop:~$ echo {{6,7,8}1,2,3}
61 71 81 2 3
ubuntu@ubuntu-desktop:~$
```

The question is where is this trick that is brace expansion applied. Of course, when making a backup of a file, you will need to use this trick hence it is very important. An example of this is given below:

cp /etc/httpd/conf/httpd.conf{,.backup}

In the above example, we are backing up the *httpd.config* file.

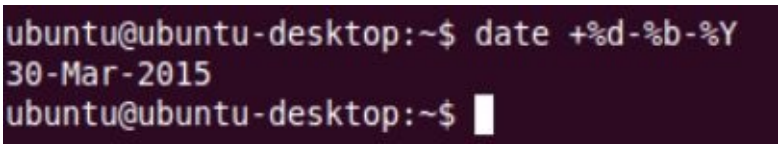
Command Substitution

This is also another useful bash scripting trick. It includes the use of the $ sign and parenthesis on the standard output to enclose any command which is aimed at generating output. This trick is applied when assigning some value to a variable.

You can assign the date to a variable as follows:

date +%d-%b-%Y

The above will give you the current date as shown below:



This can also be achieved using the following:

echo $today

today=$(date +%d-%b-%Y)

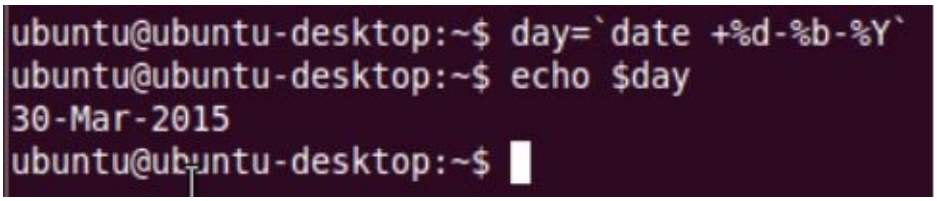
The output will just be the same. To get information about multiple *RPM* packages simultaneously, you can also use command substitution.

Command substitution can also be achieved after surrounding a command using back quotes. This can be illustrated using the example shown below:

day=`date +%d-%b-%Y`

echo $day

The above will give the following as the final result:



The current date will be displayed. With the above style, commands can easily be nested and they will become easy to read.

The output from a certain command can also be directed to a certain file. This can be done as shown below:

find / -name file > result.txt

In the above example, we are redirecting the output from the command *find* to the file *result.txt.*

However, you will notice that error messages will be produced. If you are not interested in these, you can also redirect them to a file. A *2* in front of the redirection symbol can assist in this.

They can also be redirected to */dev/null* if we are not interested in them at all. This is illustrated below:

find / -name file 2> /dev/null

The location of the file "*file*" will be shown without the error messages, *2* stands for the standard error output stream. Most Linux commands redirect their errors here, whereas the correct output which is send to the standard output is represented using *1*.

Consider the commands shown below:

find / -name file > result.txt

find / -name file 1> result.txt

Some could think that the above commands are different. However, they are just the same. The *1* represents the normal standard output.

You may need to save both, that, the error message and the standard output to a file. Cron jobs are usually used for this purpose. This can be achieved as shown below:

find / -name file > result.txt 2> result.txt

Notice that both output streams have been directed to the same file. However, the above is tiresome. It can be achieved with much simplicity as shown below:

find / -name file > result.txt 2>&1

In the example just above, we have used an ampersand to tie the standard error stream and the standard output stream. After this, the standard error will go to wherever the standard output goes. Again the syntax used in the example above as it is very important.

You might also be interested in piping the output to another command. This is shown below:

find -name file.sh 2>&1 | d /tmp/result2.txt

The above line will work just as it is expected. Now consider the line shown below:

find -name file.sh | d /tmp/result2.txt 2>&1

On running the above line, you will notice that it won’t work as it is expected. This is in contradiction to the other line.

Command History

With Bash, you can search through the commands that have been run recently. You can then use the up and down arrow keys so as to navigate through these commands. If the command you are looking for was executed between the last 10-20 executed commands, then this will be easy for you.

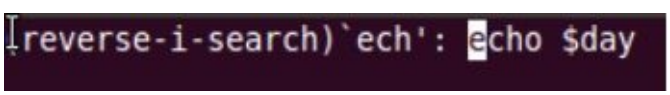
If you want to search interactively, meaning that suggestions for what you are searching for will be provided, just press *Ctrl-R*.

The prompt will then be changed to the following:

(reverse-i-search)`':

You can then start typing the letters of your command. You will notice that you will be provided with suggestions. However, the suggestions will only include the recently types and related commands.

In my case, after pressing the *Ctrl-R*, and then typing *ech*, the following is the suggestion:



He above shows that the command I ran recently containing the words *ech* is the *“echo day”* command.

To execute the command, just press the enter key and it will be executed. If you need to do some editing to the command, just press the left or right arrow keys. The command will be placed on the normal prompt and then you will be able to edit it.

Loops

With *Bash*, it is possible for you to create loops on the command line. However, this is not suitable is the code is large or too complex. On the command line, loops can be written in two different ways. The first way involves separating the different lines of code using a semicolon. To create a loop that will back up all the files contained in a certain directory, this can be done as follows using this method:

for doc in *; do cp $doc $doc.bak; done

The above line of code will back up all the files in the directory. Rather than separating the lines using a semicolon, a secondary prompt can be used. This can be done by pressing the enter key once you are through with each single line. Once you have pressed the enter key, Bash will understand that you want to create a loop and it will provide you with a secondary prompt.

You can then enter your second line. This is illustrated below:

```
for doc in *
> do cp $doc $doc.bak
> done
```

Sometimes, your Linux system can run out of memory. This means that you will not be

able to execute your commands including the simple ones such as *ls* and the ones for changing directories. If *ls* fails and you want to vie the files contained in a particular directory, then this can be achieved as follows:

for doc in *; do echo $doc; done

This will solve the *ls* problem. This shows the power of bash scripting.

Chapter 4- Linux shell programming

In Linux, it is very easy to create shell scripts. With shell programming, we can group multiple commands in a chain and then execute them to obtain the desired output. Let us begin by writing out first script. Open your editor of choice; I am using the *vi editor*. Add the following shell program to the file:

```
#!/bin/bash
Echo “Hello World”
```

The above is an example of a shell program. Save the file and give it the name hello.sh. Notice that we have used the *.sh* extension to imply that it is a shell program. Now, the next step is to make the script executable. This can be achieved as follows:

Open the command prompt and type the following command:

```
chmod 744 hello.sh
```

After running the above command, an executable of the file hello.sh will be created, *chmod* stands for change mode.

744 refers to the permissions, and we have used this to add an execute permission to the file. Since we have made the program executable, we now need to execute it.

This can be achieved by running the following command on the terminal:

./hello.sh

After running the above command, the following output will be observed on the standard output:

The command outputs a text written "*Hello World*". Remember that this is the text that we command the echo command to output.

#!/bin/bash is called the *shebang* is used to precede shell programs.

To add comments to your program, precede them with a pound sign (#).

Consider the shell program shown below:

#! /bin/bash

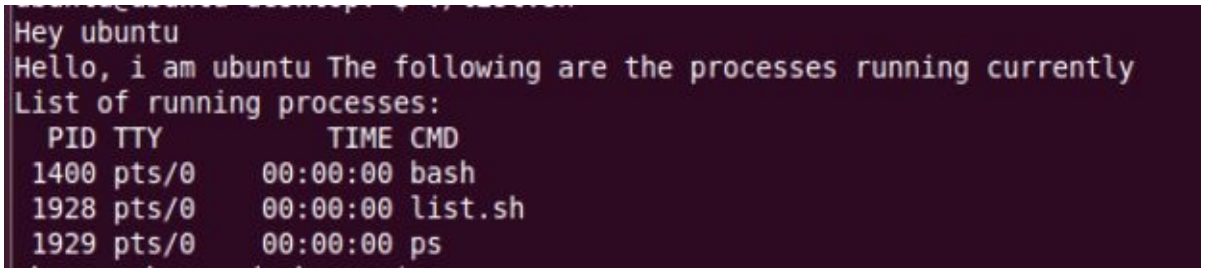
echo “Hey $USER”

echo “Hello, i am” $USER “The following are the processes running currently”

echo “List of running processes:”

ps

You can write and run the program. The following output will be observed after running the program:



As shown in the above figure, the program outputs your name. It also shows the list of processes which are currently running on your system.

Now, let us write a more interactive program. Write the code shown below and run it:

#! /bin/bash

echo “Enter your name”

read name;

echo “welcome Mr. /Mrs. $name “;

echo “Thanks Mr./Mrs. $name. You have provided us with your names”;

echo “————————————————”

echo “Mr. /Mrs. $name, good bye friend”

After running the above program, the following output will be observed:

As seen, you are prompted to provide your name. This has been achieved by use of the read command, which prompts users to provide their input. This input has then been displayed alongside some other text.

Variables in Shell

The two types of shell variables are:

1. System variables- these are created and maintained by the OS. They are defined in upper case.
2. User Defined Variables- the user creates and maintains these. They are defined in lower case letters.

The following are some of the system variables in shell:

BASH- the shell name.

HOME- our home directory

LOGNAME- our login name.

PATH- our setting for the path.

USERNAME- the currently logged in user.

COLUMNS- number of columns making up our screen.

OSTYPE- the type of the OS we are running.

PWD- the current working directory.

The above shows only a sample of the system variables. In case you want to print any of the above variables, use the echo and the $ sign command as shown below:

echo $HOME

echo $PWD

The next part is on how to work with user defined variables.

Use the following syntax to define a user defined variable:

variable name=value

The name refers to the name to be used to refer to the variable. The value is what it is. This is usually a number or a string. Example:

num=10

name=Mary

Tips when naming variable

1. There should be no spaces on either sides of the equal sign. Example:

num=12- this is right

num= 12- this is wrong

num =12- this is wrong

num = 12- this is wrong

2. The name of the variable should begin with an underscore or of the alphanumeric characters. This should then be followed by any of the alphanumeric characters. The following are examples of valid variables:

PWD

SYSTEM_VERSION

num

3. The names of variables are case sensitive. Example:

Num=2

num=6

nUm=9

nuM=10

NUM=90

The above shows that the above variable names are not equal. To print any of the above, also make sure that you specify correctly. For example, to print 10, use the following code:

echo $nuM

4. To define null variables, do the following:

num=

num=””

To print it, use the following code:

E=echo $num

However, note that nothing will be displayed since the variable is null.

5. Some characters should not be used to name variables. These include the

asterisk (*) and the question mark (?).

Loops in Shell

A loop represents a group of instructions executed repeatedly.

for Loop

This loop follows the following syntax:

for { name of the variable} in {list }

do

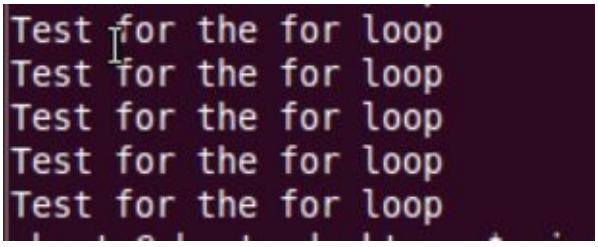
execute the items in the list one after another until finished.

done

Consider the program shown below:

```
for j in 1 2 3 4 5
  do
  echo “Test for the for loop”
  done
```

After running the above program, you will observe the following as the output:



What the program does is that it begins by defining the variable *j* and sets its value from 1 to 5. The line *echo "Test for the for loop"* of the program will then be executed 5 times since the value of *j* will increment up to a maximum of 5. This explains the source of the output.

while Loop

The while loop in shell takes the following syntax:

while [condition]

do

1st command

2nd command

3rd command

..

….

done

As long as the condition that you have specified is true, the loop will execute.

Case statement

It is also common in Linux. One can use it rather than the *if-else* statement.

It takes the following syntax:

case $name_of_the_variable in

1st pattern) command

…

..

command;;

2nd pattern) command

…

..

command;;

nth pattern) command

…

..

command;;

***) command**

…

..

command;;

esac

Notice that we start with *case* but we end with an *esac* statement. Consider the example shown below:

```
if [ -y $1 ]
then
  r=”the vehicle is unknown”
elif [ -x $1 ]
then
  r=$1
fi

case $r in
   “lorry”) echo “For $r 10 per k/m”;;
   “merc”) echo “For $r 80 per k/m”;;
   “corolla”) echo “For $r 15 per k/m”;;
   “motorbyke”) echo “For $r 5 per k/m”;;
   *) echo “we can’t obtain a $r for you”;;
esac
```

if-else-fi for decision making

The command *if-else-fi* can be used in shell programming for the purpose of decision making. The programmer specifies a condition and once met, a command is executed, if not met, the other command is executed. It takes the following syntax:

if condition

then

the condition is zero

execute commands until you meet the else statement

else

if the condition is false

    execute commands until you find the fi statement

fi

Consider the example shown below:

```
#!/bin/bash
if [ $# -eq 0 ]
then
echo “$0 : an integer must be supplied”
exit 1
fi
if t $2 -gt 0
then
echo “$2 is a positive number”
else
echo “$2 is a negative number”
fi
```

If the command line is not given, then an error will be printed, that is, an integer must be supplied. The next part involves checking the number of arguments that have been passed. If we passed any argument, then the *“if* “will turn out to be false and true otherwise. The *exit 1* statement will terminate the program after a successful execution.

Chapter 5- Bash One-liners

Working with files

1. Truncating the size of a file to *0* (emptying) - the redirection command is used in this case. If the file exists, then its size is truncated to zero whereas if it does not exist, it is created. Note the operator opens the file for writing.

$ > file

To create a file containing a content of choice or replacing the file's contents, use the following command:

$ echo "string to replace" > file

The string that you specify in between the quotes will act as the replacement string.

2. Adding a string to your file- the on-liner uses the output redirection operator >> to append some content to your file. Example:

$ echo "content to be appended" >> file

In case the operator fails to find the file, anew one is created. After the string has been appended to the file, a new line will follow it.

To avoid this, use the *–n* option as shown below:

$ echo -n “content to be appended” >> file

3. Assign the first line of a file to a variable- in this case, you need to read the first line of the file and then assign this line to a variable. The input redirection operator < is used.

Consider the line of code shown below:

$ read -r myline < file

myline is a variable. The command read will read only the first line of the file and place it in the variable *myline*. The *–r* option will ensure that this line is read raw.

4. Reading a file- a *read* command can be combined with the *while* loop so as to read a file line-by-line. This is illustrated below:

while read -r myline; do

```
        # perform something on the variable myline
done < file
```

A code of failure will be returned when the *read* command meets end-of-failure. The *while loop* will then halt.

Notice that we have placed < *file* at the end. To avoid this, pipe the file's content to a while loop as shown below:

```
cat file | while IFS= read -r myline; do
        # perform something on the variable myline
done
```

5. Read any line of a file and assign it to a variable - external programs assist the bash to read the lines of a program. Consider the code shown below:

```
$ read -r rline < <(shuf file)
```

Shuf is available on modern Linux systems and helps in this. The above code will read a random line of a file and assign it to the variable *rline*.

6. To extract a tar archive, use the following command:

tar xvf b.tar

7. To find the number of lines common between two files:

sort doc1 doc2 | uniq –d

8. To randomize the lines of a file:

shuf filename.txt

9. To sum a certain column:

awk ‘{ sum+=$1} END {print sum}’ document.txt

Conclusion

It can be concluded that Linux is an open source operating system and it has various distributions. The idea behind development of Linux was to develop a free Unix-like operating system for computers. Each of the Linux distributions comes in two versions, that is, the server and the desktop versions. The desktop versions support both the GUI and the command line.

By default, one accesses the GUI after login into the system. The command line can then be launched if needed. The server versions of these distributions support only the command line. No graphics in Linux servers. The purpose of this is to ensure security of these systems since hackers usually take advantage of any feature added to this system. This explains why you should be familiar with the Linux commands.

The Linux command Line can be used for nearly everything needed in the system. One earns much respect in the society when they become experts in using the Linux command line. With shell scripting, programmers can put together multiple commands and run them simultaneously so as to achieve the desired output. This also saves on time and space available on the system. Shell scripts are preceded by the shebang.

This alerts the system a shell scripts is just to be executed. These must be made executable and then execution will take place after this. Bash stands for Bourne Again Shell. It is the default shell in most Linux distribution. Bash scripts are run on the command line, and one can achieve much with them, and know much about the system. Bash on-liners are also important in Linux. One can use them to play around with their files.

Thank you!

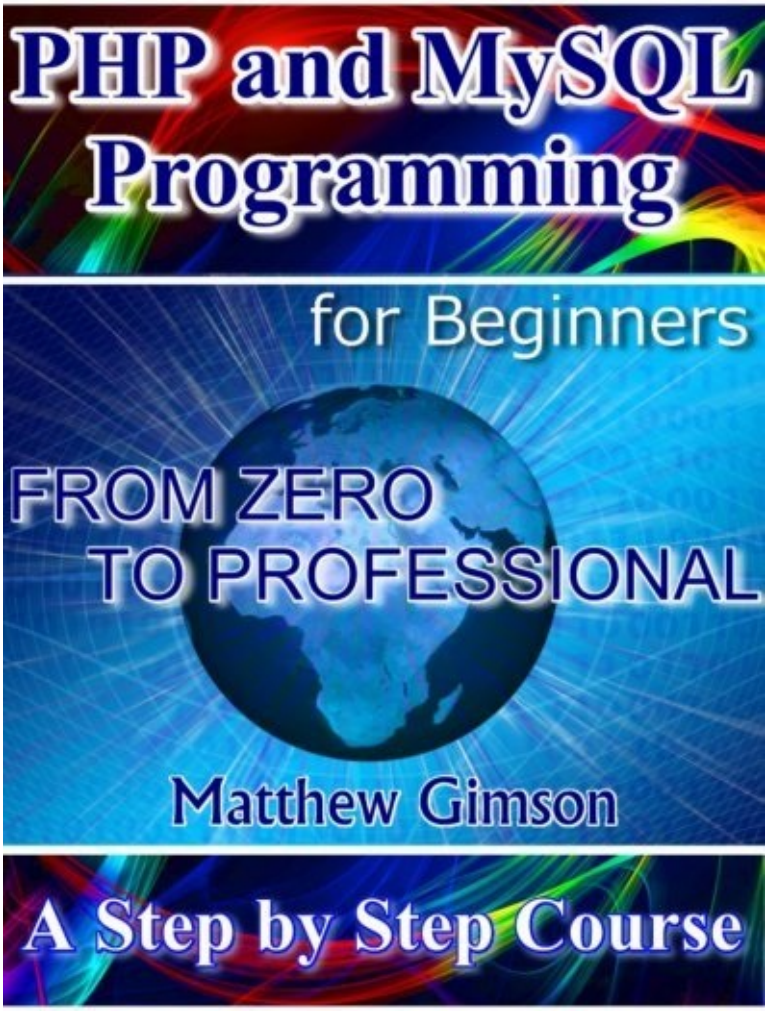
We would like to thank you for buying this book. Hope you found it helpful in your now EASY and FAST programming life development. And we are happy to recommend you some other books from this author:

1. **Linux Command Line: Become a Linux Expert! (Input/Output Redirection, Wildcards, File Security, Processes Managing, Shell Programming Advanced Features, GUI elements, Useful Linux Commands)**

   http://www.amazon.com/gp/product/B00XGUO4E4?*Version*=1&*entries*=0

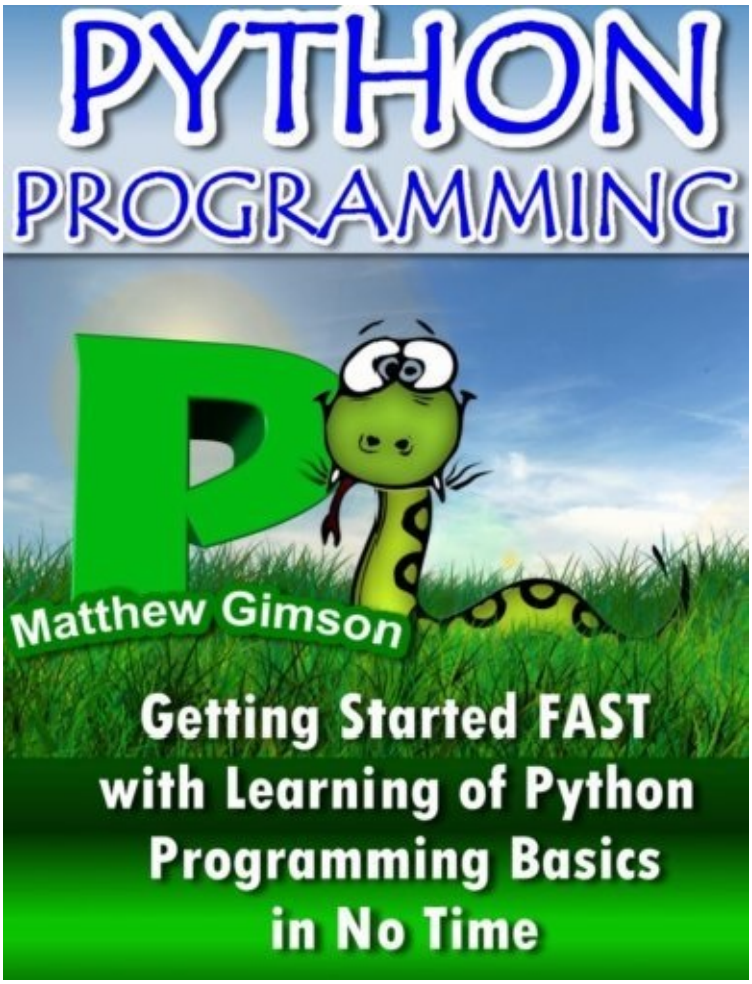
2. **PHP and MySQL Programming for Beginners: A Step by Step Course From Zero to Professional**

http://www.amazon.com/gp/product/B00XQBYXVW?*Version*=1&*entries*=0



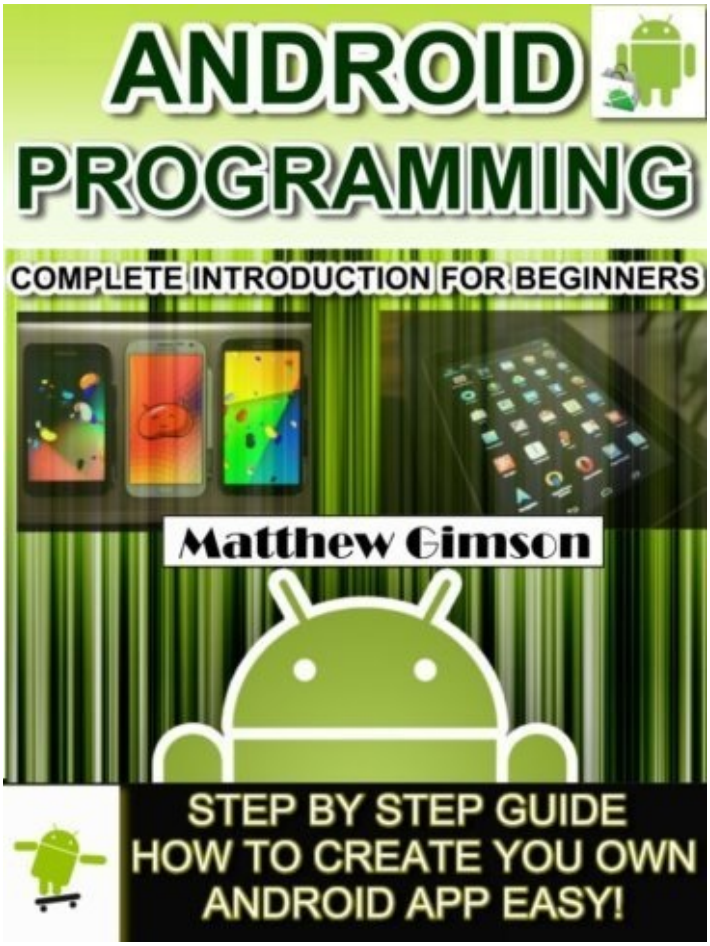
3. **Python Programming: Getting started FAST With Learning of Python Programming Basics in No Time.**

http://www.amazon.com/gp/product/B00WUNSH6Y?*Version*=1&*entries*=0



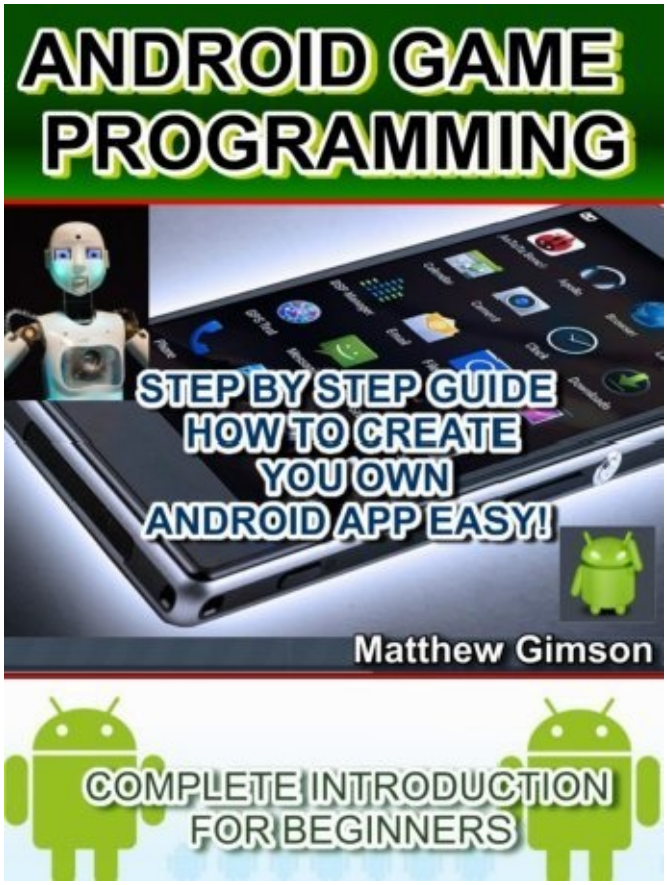
4. **ANDROID PROGRAMMING: Complete Introduction for Beginners -Step By Step Guide How to Create Your Own Android App Easy!**

   http://www.amazon.com/gp/product/B00WPK68IQ?*Version*=1&*entries*=0



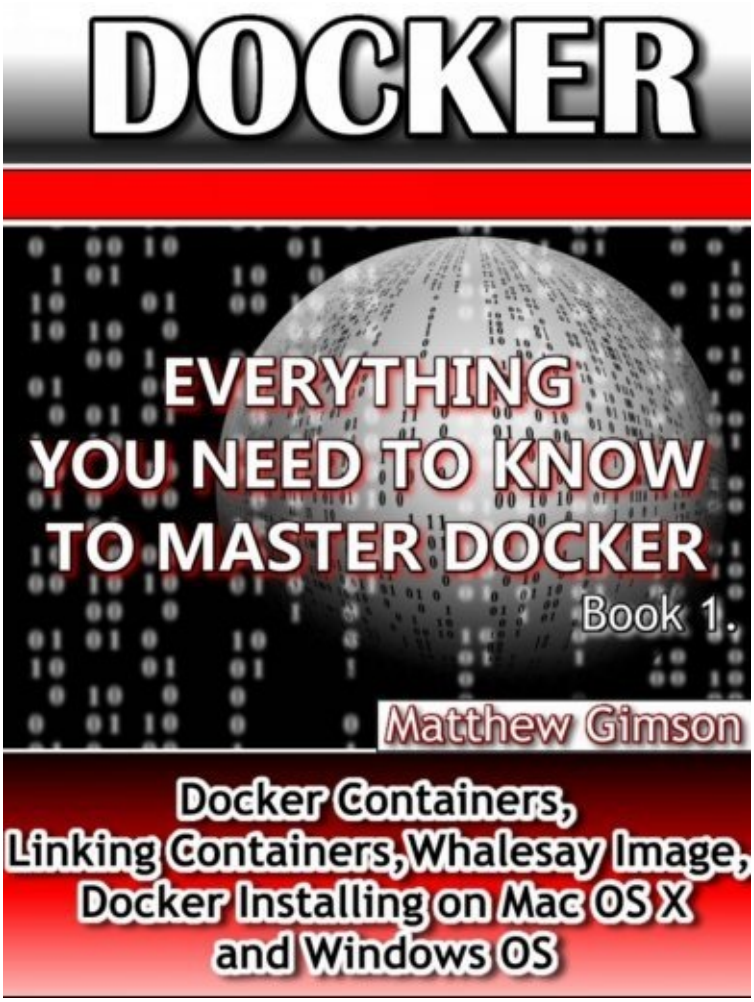
5. ANDROID GAME PROGRAMMING: COMPLETE INTRODUCTION FOR BEGINNERS: STEP BY STEP GUIDE HOW TO CREATE YOUR OWN ANDROID APP EASY!

http://www.amazon.com/gp/product/B011R2H2JQ?*Version*=1&*entries*=0



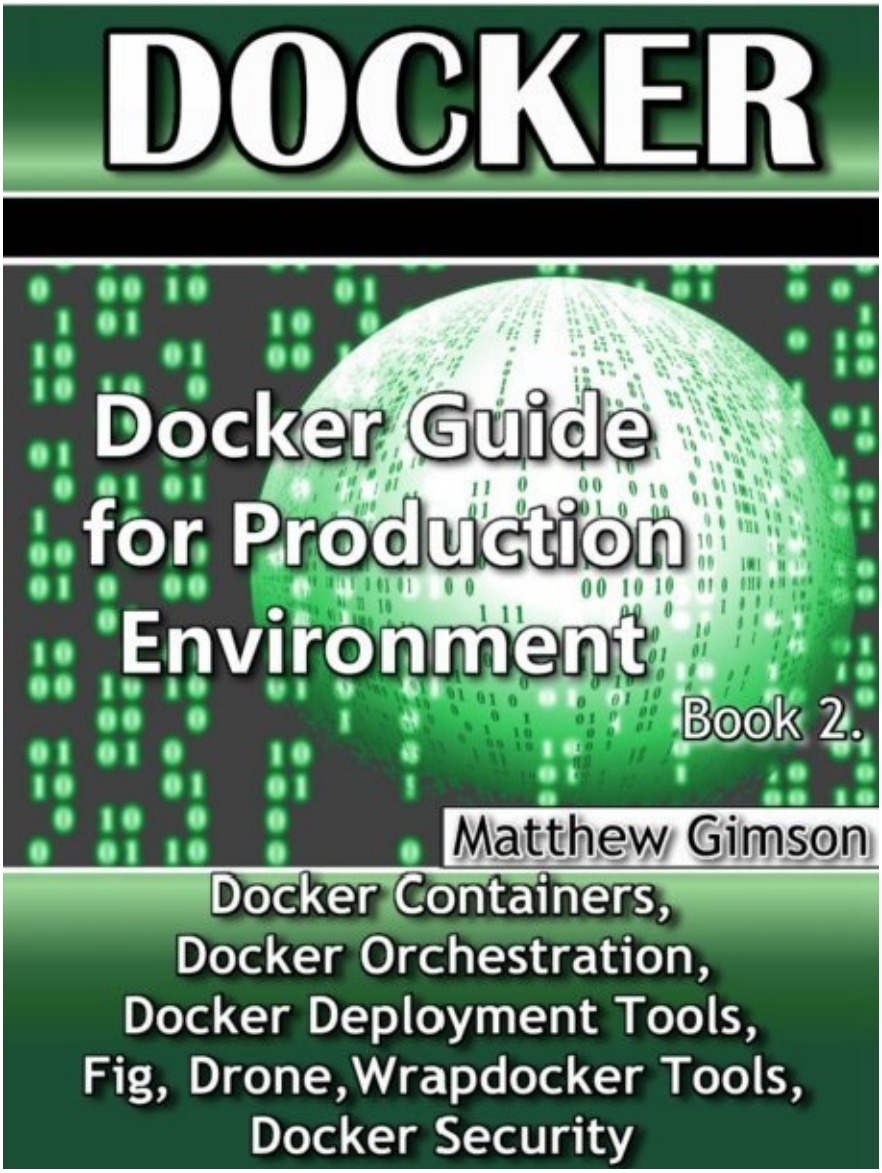
6. **DOCKER: Everything You Need to Know to Master Docker (Docker Containers, Linking Containers, Whalesay Image, Docker Installing on Mac OS X and Windows OS)**

http://www.amazon.com/gp/product/B013X2RPT0?*Version*=1&*entries*=0



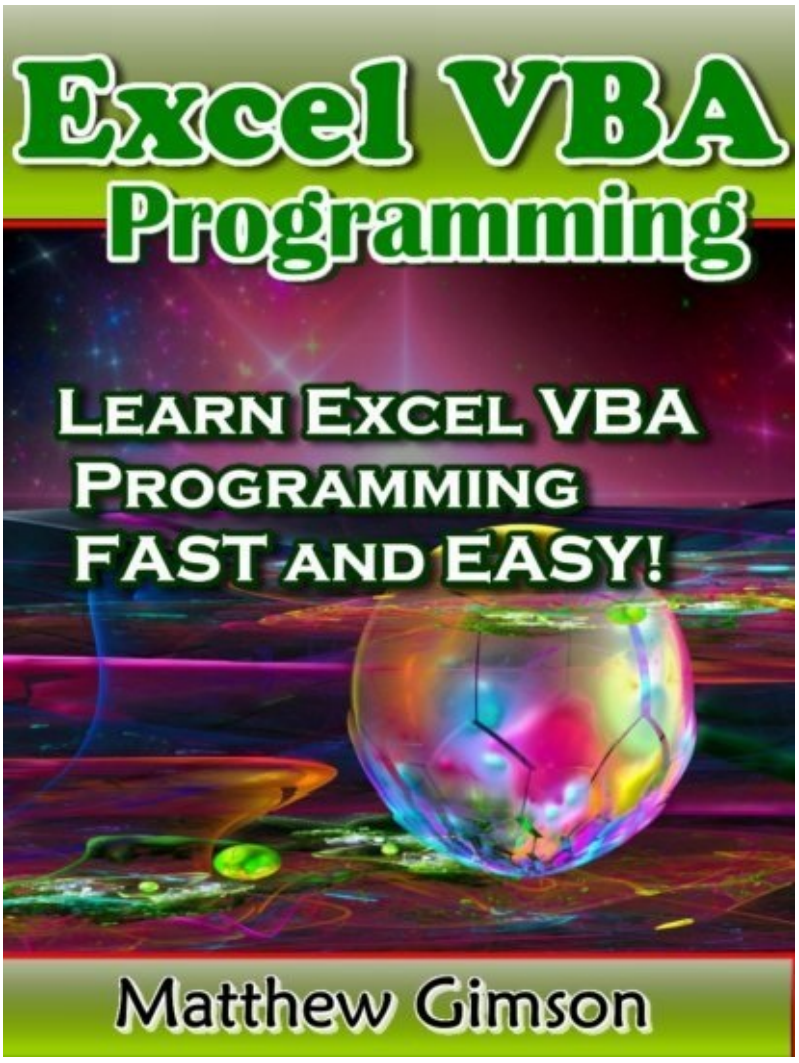
7. **Docker: Docker Guide for Production Environment (Programming is Easy Book 8)**

http://www.amazon.com/gp/product/B01452V9IA?*Version*=1&*entries*=0



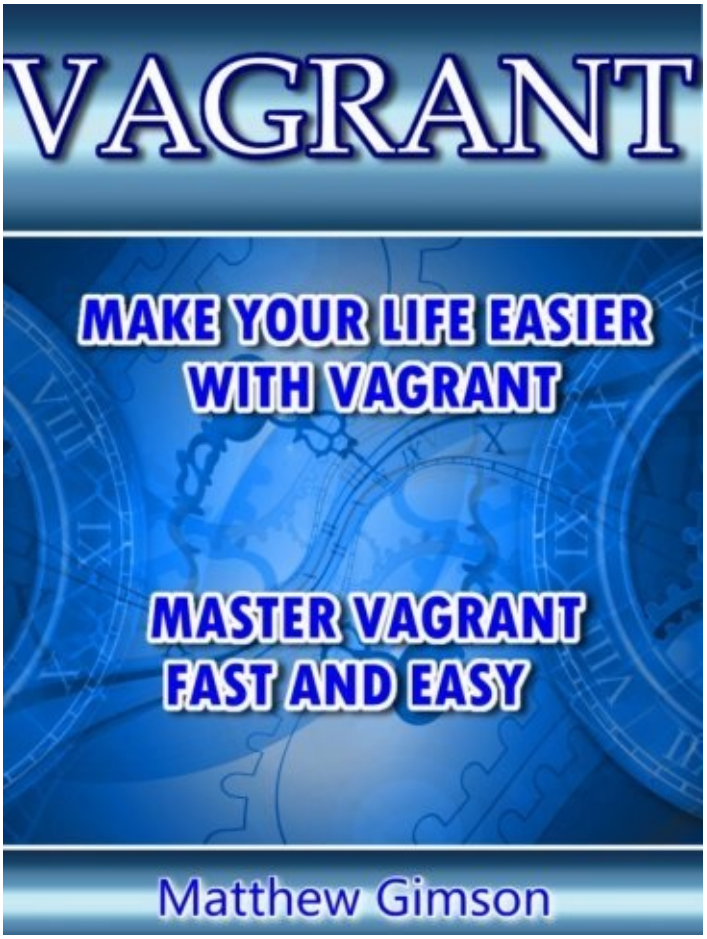
8. **Excel VBA Programming: Learn Excel VBA Programming FAST and EASY! (Programming is Easy Book 9)**

http://www.amazon.com/gp/product/B014DIPGVW?*Version*=1&*entries*=0



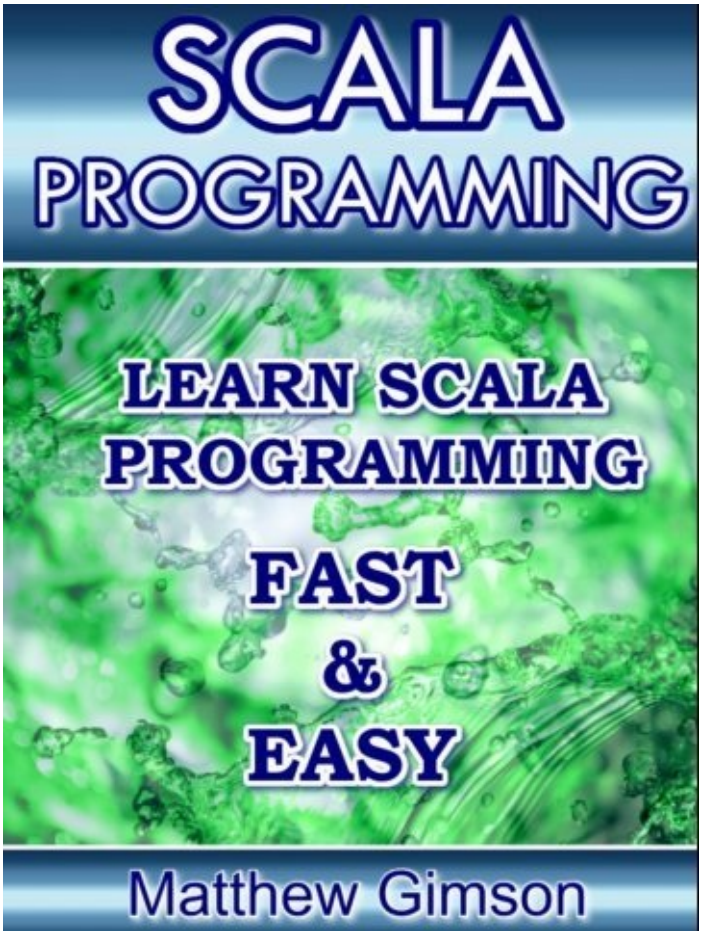
9. VAGRANT: Make Your Life Easier With VAGRANT. Master VAGRANT FAST and EASY! (Programming is Easy Book 10)

http://www.amazon.com/gp/product/B0151GIRCA?*Version*=1&*entries*=0



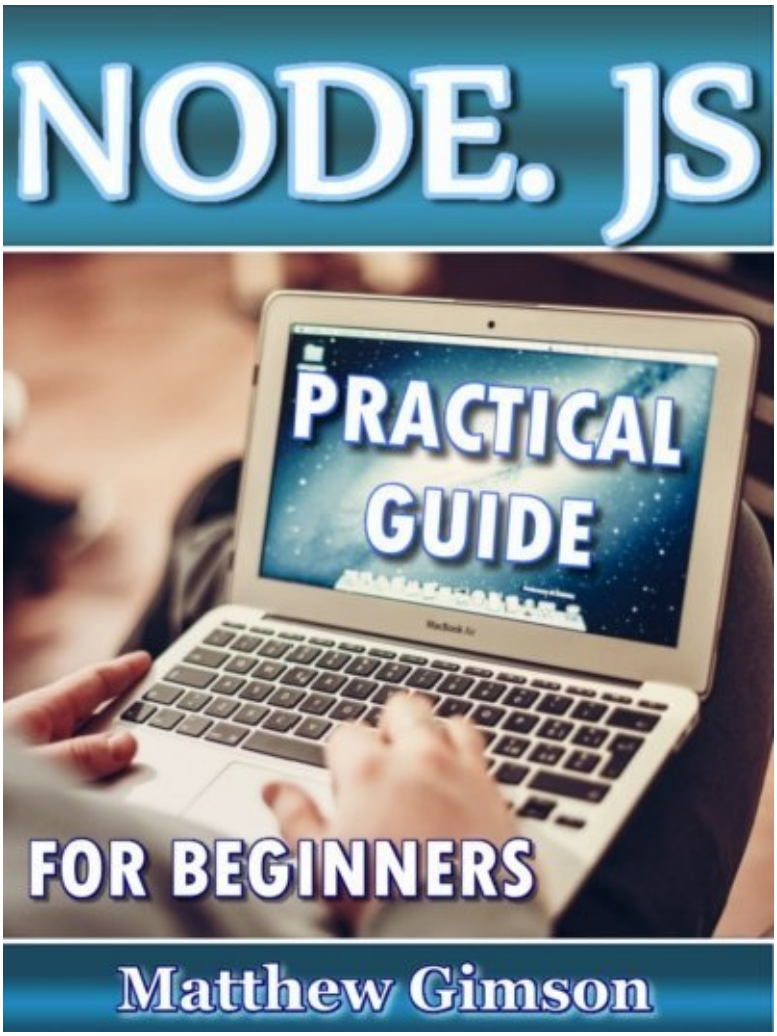
10. SCALA PROGRAMMING: Learn Scala Programming FAST and EASY! (Programming is Easy Book 11)

http://www.amazon.com/gp/product/B0151TBXEQ?*Version*=1&*entries*=0



11. **NODE. JS: Practical Guide for Beginners (Programming is Easy Book 12)**

http://www.amazon.com/gp/product/B01588CXAS?*Version*=1&*entries*=0



12. **IOS 8 APP DEVELOPMENT. Develop Your Own App FAST and EASY!**

http://www.amazon.com/gp/product/B015CMEJVQ?*Version*=1&*entries*=0