From “Mumu”

To

Master:

Command Line

And

Version Control

In Minutes

BY OGUNDIRAN AYOBAMI

About the author

Ogundiran Ayobami is Nigerian born software developer. He is a lover of software development in its entirety despite his background in Art. He loves learning, unlearning and relearning to get great things done. He can be reached on twitter @ayovision, facebook as Ayobami Ogundiran and as codingnninja on github.

This material is dedicated to Nigerian community of software developers for their resilience against all odds.

This material is not properly proofread, so you may come across some errors. Thanks for understanding.

Command Line and Version Control make life easier for developers for so many reasons which you will understand better by the time you start using them in your projects. It goes without saying that it is more than necessary to learn using Command Line and Version Control as a developer to make life easier for yourself

- It can easily be automated.
- It provides a simple access to options.
- *It is expert*-friendly.
- It makes it possible to log commands to review or repeat actions.
- Adding special sub-options can be done with ease.

For you to know the practical reason for using these things, answer the question below:

- You are currently in your office and it is 1:50pm; just at the moment a company calls you to come and close a N20 billion contract which you have to start its processing by exactly 2:00pm and ends it by 3:00pm without stopping. That same day, your partner is also coming from Lagos to your office to collect a file from you to close an important contract but he will get to your office by 2:20pm. Coping or sending (on internet) that file will take 15 minutes but you can only spend 2 more minute in your office; yet you can't ask anybody to help you copy it because you have confidential documents on your laptop. You can only ask your secretary to give the flash that contains the copied files to your partner but your computer must be shutdown by the time she gets there to take the flash.

  How will you do it?

It is impossible: If you think with your normal way of coping files and shutting down your computer, you won't be able to solve this question but using command line will solve it in seconds.

Did you say: show me the answer if it is possible?

Don’t worry! I will show you how to do it along the line. Just keep reading.

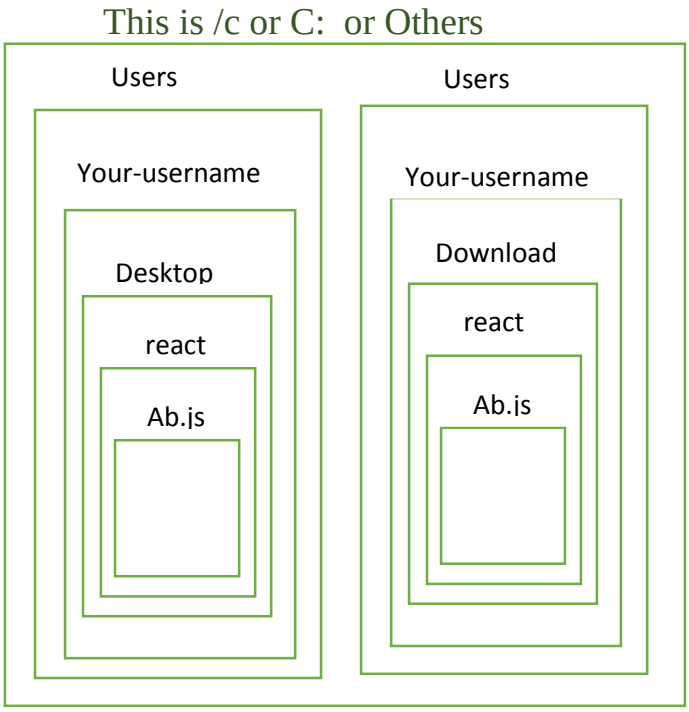
Download Git Or Use CMD

The first thing you have to do is downloading git here ; then install it and get a bottle of ZOBO to calm your nerves. While installing git, make sure you select features that are suitable for your project. If you have any issue installing it to support your project, add me on Facebook @ www.facebook.com/ayobami.ogundiran or ask other people you know.

Understanding $PATH

After you have installed git, it is time to do something meaningful and magical. Let’s starts with understanding $PATH. When you have a $PATH like /c/users/your-username/ Desktop/ react/ Ab.js or /c/users/your-username/Download/react/Ab.js , do you know what it means? Don’t rack you brain, check below.

The $PATH above can be represented with files and folders as in below:



The visual representation of files and folders above shows that your Ab.js file is inside REACT folder which is on the desktop page or in the download folder and so on. You should have got a glimpse of how that works through the above visual. The next thing is writing commands in real time. First thing first: launch your git bash/cmd or git.

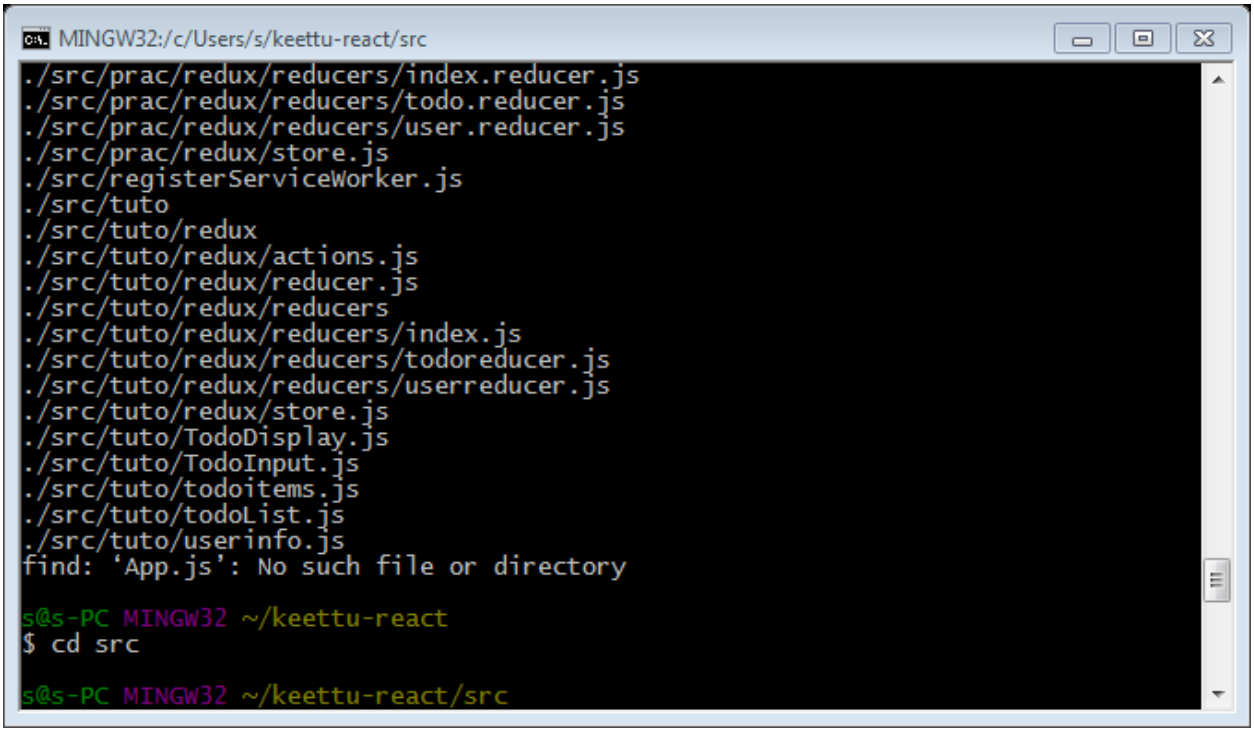
Wait a second! Let’s talk about command, flag and argument for you to understand how they are used better.

Understanding Command, Flag and Argument

Let's look at this: rm -rf filename

rm is a command or utility ; -rf is a flag and filename.txt is an argument to the command. Flag can sometimes be indicated with two dashes (--) or a dash (-). Command is the instruction to be carried out; flag is used to give options or preferences to command's execution. And argument is the object the command or instruction operates on.

What does the command syntax above do? Don't worry, we will get to that soon.



Let's start with **START**

START

Start is a command that is used to launch applications or execute computer files. For example, your Mozilla Firefox browser will be launched if you run start firefox in you git console. All you have to do now is to open up you git console play with start command to come up with several ways to use it.

cd

Cd simply means CHANGE DIRECTORY. It is used to traverse directories, this is, it handles opening of folders. If you are currently at the root directory but you want to do something on the desktop page; you can run cd $PATH (e.g cd /c/users/User/Desktop). By running that command, you should be in your desktop folder. Let's say you want to create a folder in your desktop page after navigating to the page. Then you can use **mkdir**.

mkdir

To create a directory, just run this command: mkdir DIRNAME in your command line console. It will create a folder with the name you specified. E.g. mkdir Class. Oh! The file you created is not needed, you have to delete. It is time to run: rmdir DIRNAME.

rmdir

rmdir is simply used to remove directory(s). In case you feel a directory is unnecessary and you want to remove it; just use rmdir in your git to delete it. E.g. rmdir Class.

cp

cp simply means copy and it is used to copy a file or folder to another folder. Be reminded that copy is the equivalent of cp while using windows cmd. To use this, you need to run cp $FILL-TO-COPY $FOLDER-TO-COPY-TO. E.g. cp myfile.txt Class. The foregoing will copy the file named myfile.txt to the folder named Class provided that they are in the same folder. Make sure you specify correct paths

touch

touch is used to create new file. To use 'touch', just run touch $NEW-FILE-NAME. E.g, touch mumu.js. What if you want to create multiple files will just a code? Don't worry! Keep reading.

rm

Sometimes, there may be a file you want to delete for a genuine reason; in such case, you can use rm to remove the file with ease. This is how to use it: rm $FILENAME.ANYTHING – e.g. rm new-file.txt.

pwd

In some occasions, you may not know the directory you are currently working inside of; then you may want to find out and all you need to use in git or cmd is pwd which means present working directory. To use this, just run: pwd in terminal or cmd and it will show you the path to the folder you are currently working inside of.

..

These two dots mean parent directory. If I run cd /c/users/Ayobami/desktop, my directory will be changed to my desktop page or folder, but what I actually want to work on is in Ayobami folder; then I have to go back to Ayobami. Considering the diagrammatic explanation provided previously in this book, Ayobami folder is the parent of desktop. In this case, I want to go back to the parent directory which is Ayobami folder, so I need to run cd .. which means go back to the parent directory. Thus, .. means parent directory.

.

This "." simply means current directory. May be you have many folders inside Ayobami folder but you want to run a command to work in the same folder; then, you can use **.** to show that the command should start from the current folder. So, "**.** " means current directory.

-

The symbol above is an hyphen and it is used in command line to mean previous directory. When you have changed one directory to another, sometimes, you may need to go back to the previous folder; just run: cd – .

Ls

In some instances, you will need to check files that are present in a directory to know what to choose or open for use. Then, ls can be of importance. Ls means “list contents”, so when you use it; the contents of the present working directory will be listed. E.g. ls .

Less

Any time there is a need to open are file git or cmd, less command becomes useful. If a file has been created and it is time to open it; just run: less $filename.txt. Be informed that the file will be opened in the same command line console, which may not be want you want, then what will you do to open it in an application of your choice like notepad, Microsoft word or excel, etc.? In this case, we need to make use of start as used below:

start file.txt.

* (Wildcat)

You do call the above symbol asterisk in general use but it is called wildcat in command line term. What does it do? It means “**all or anything”** whenever it is used. For instance, the command below means that all files that end with .txt be deleted:

rm *.txt

Note: Check below for more applications.

Find

Find is used, as its name suggests, to locate files. You can use it to search any files on your system. E.g. find image.jpg

Rename / ren

Any time you want to rename a file or multiple files, rename command becomes useful. It is run like: rename OLD-NAME NEWNAME

shutdown

This is a command that can be used to shutdown your computer from the command line console. It very useful since it makes it easier for you to time your computer and even shutdown itself when you are not around. It can be run as in below:

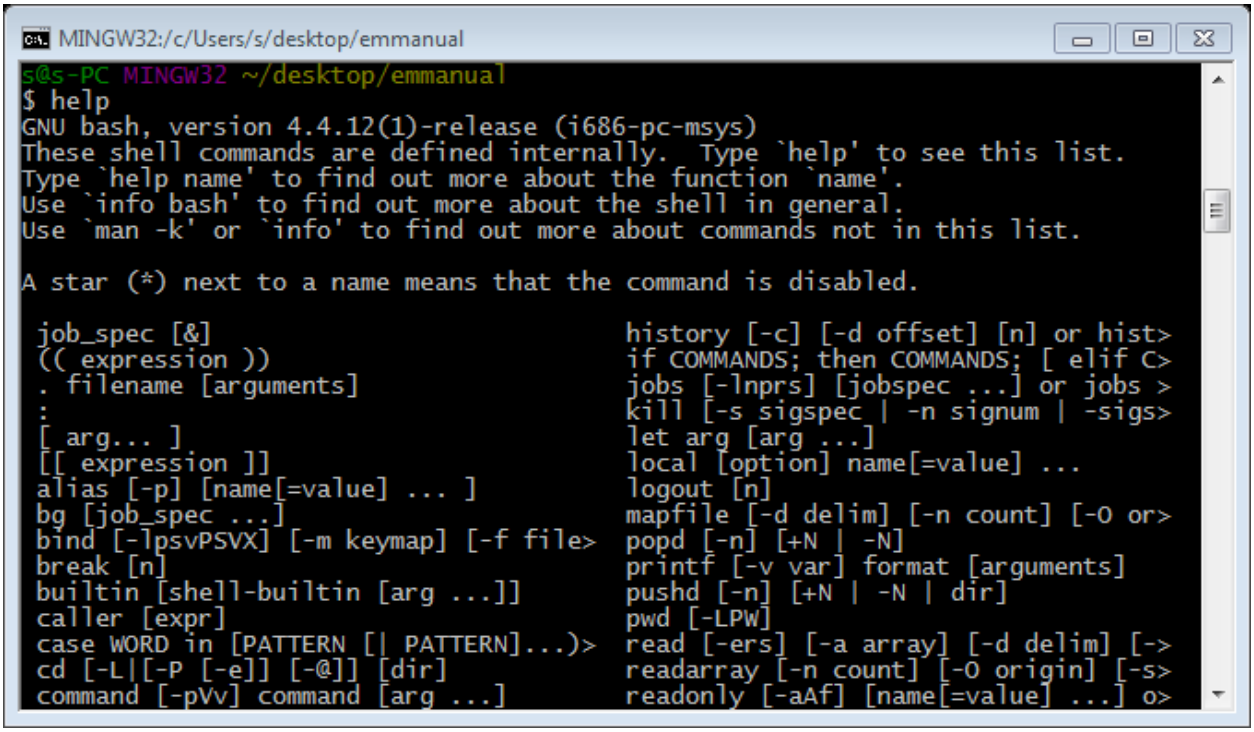
Shutdown –s (shutdown only)

Shutdown –r (shutdown but restart)

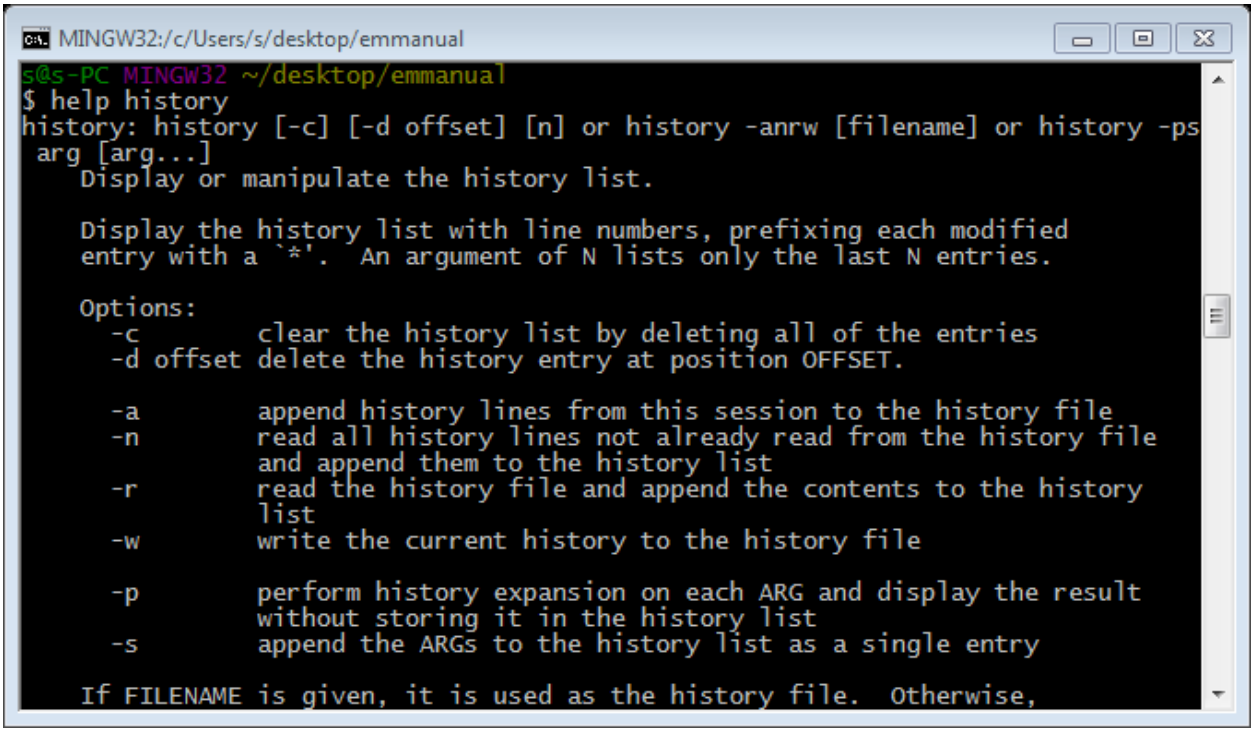
Shutdown –l (logoff) etc.

Help

Help is an important command that enable us to get more information about how other commands are used. To get information about many commands in command console, just run help



We may choose to find out further information about one of the commands shown in the console, so we can just run help CommandName. E.g. help shutdown



Using Bash Shortcut Keys

There are a number of very useful shortcut keys you can use in the bash shell, and it pays to master them all. Here's a couple to get you started:

- **Ctrl + U**: To clear the line from the cursor point back to the beginning, press ctrl + u.
- **Ctrl + A**: To move the cursor to the beginning of the line, press ctrl + A.
- **Ctrl + E**: To Move the cursor to the end of the line, press ctrl + E.
- **Ctrl + R**: To allow you to search through the previous commands, press ctrl + R.

Note: This may not work on windows if you don't have git-bash installed.

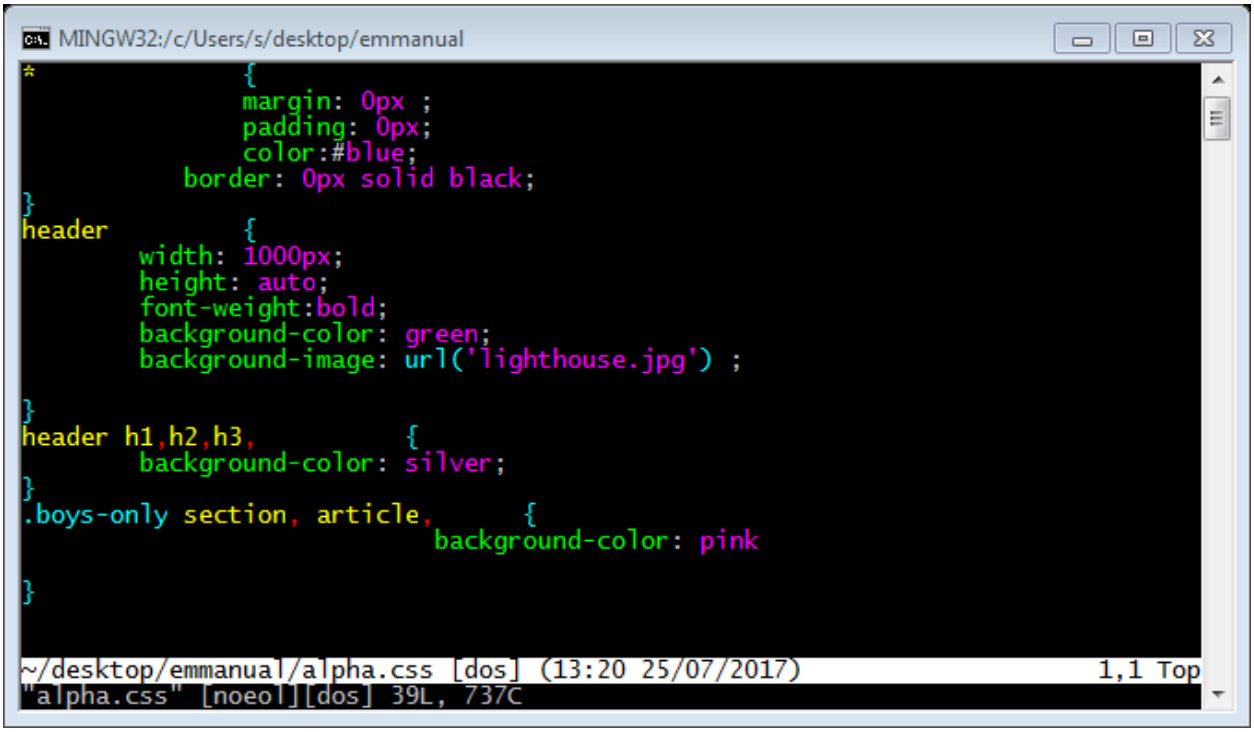
History

History will show all you have done your computer previously.

Using Vim Editor

If you have vim installed on your computer, you can easily edit your files in vim editor as shown below:

vi file.txt – this will open you file.txt in vim editor



Press i to insert or edit your file. After editing the file, but there is no need to save it just press esc key and type :q! (to quit it without warning) or :q (with warning); in case there is a need to save it, just run :wq (write to disk and quit).

More Applications:

This section shows how to think and use command line to make life easy for you.

1. Answer to the question asked at the beginning of this book:

```
cp /c/users/ayobami/desktop/file.txt  D:/  && shutdown -s -t72000
```

This will copy the file named file.txt to D: / (flash) and the computer will shutdown in 20minutes. You may remove –t7200 if you don't want to set time.

Note: If you drag the file and directory to the cmd, it may not work in git-bash. To correct the cause, just change \(backward slash) to /(forward slash) wherever you find it in the path in command line console.

2. Sometimes, you may need to run multiple commands once. For example, you want to create a file and open it in notepad.

Ans: `touch file.txt | start file.txt`

This "|" is called pipe; we will talk more about it in this material later.

3. You may need to open many application at the same time or once, then command line becomes useful.

Ans:

```
start C:\Users\s\Desktop\IMG_20170705_135937.jpg | start C:\Users\s\Desktop\php_objects_patterns_and_practice_3rd_edition.pdf | start C:\Users\s\Desktop\Timing.txt | start C:\Users\s\Desktop\web
```

or

```
start C:/Users/s/Desktop/img.jpg  && C:/Users/s/Desktop/img2.jpg && start C:/Users/s/Desktop/img2.jpg
```

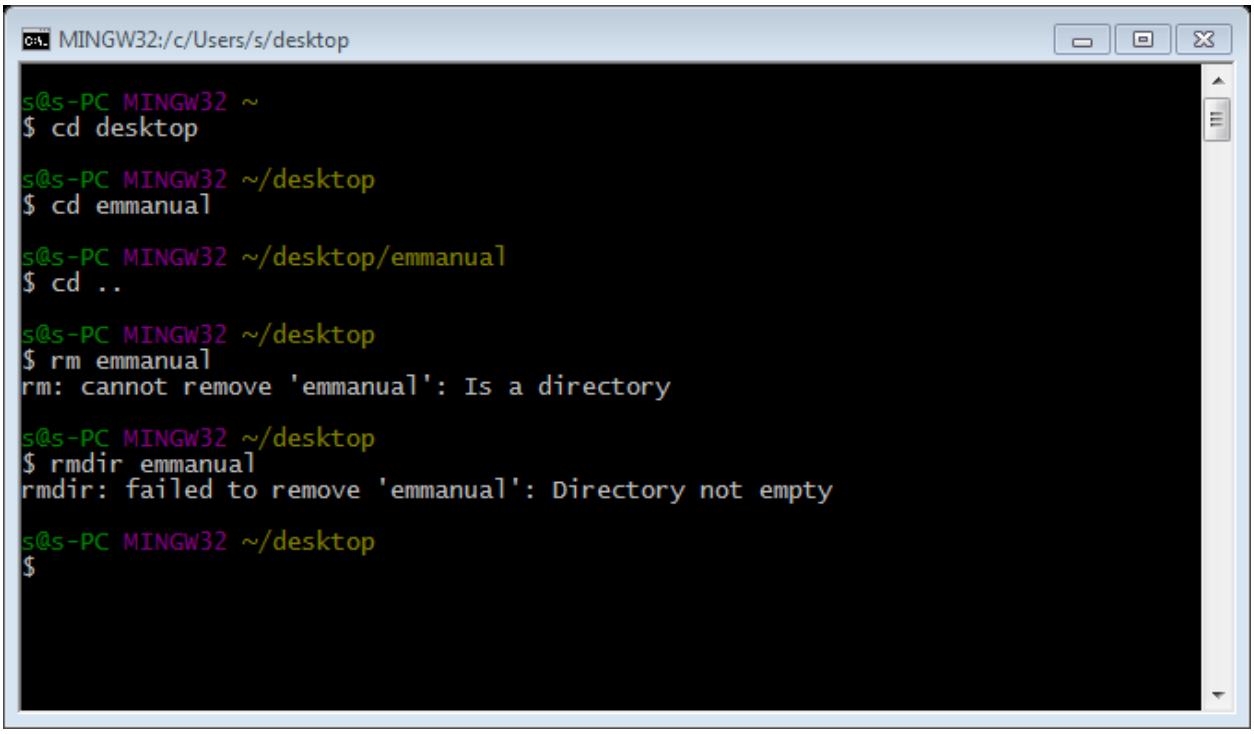
Note: I didn't write above paths, I dragged each of them to the console. You may need to change \ to / if it doesn't work in your console.

With the above commands, I start three things once – an image , a book and a folder.

But, don't you think it is not necessary to repeat start and "&&". Then, how can we do it? Watch out for that in volume ii.

4. It may be needed to delete a folder. If that is necessary, check below command.

rm foldername ; this will give below error message



In order to force the command to delete the file, just run rm –rf foldername.

Hey! What is the meaning of –rf? Expect what it does later in this material.

Multiple actions

Hey! Welcome back to this part of the tutorial. I promised to show you how to use pipe |, start, touch and and (&&) for multiple actions.

Let start with **touch,** if you want to create multiple files at once, touch will be handy. I always use touch to create multiple files after I have planned the structure of my application on paper.

1. To create multiple files, just run touch first_file.txt second_file.js third_file.py and so on.

2. To remove multiple files, just run rm first_file.txt second_file.js third_file.py and so on.

3. To start multiple files, folders or applications with **start** we need to use pipe **|**. What does pipe do? It passes files, folders or applications as arguments to a command from right to left. E.g start firefox | opera | notepad ; this command will start firefox first, opera second and notepad last. That means pipe is used to pass several arguments to a command.

4. To create multiple folders, just do as usual. E.g mkdir first_dir second_dir third_dir.

5. To delete multiple folders, run rmdir first_dir second_dir third_dir. I also promised to explain the use of **rm –rf foldername**. Just keep reading. I will discuss it later in this material. Yay!

Pipe |

You should have known the usage of pipe | from my explanation while discussing **start.** Anyway, let me repeat my explanation. Pipe | is used to pass many arguments to a function. E.g start firefox | opera | notepad. You should have noticed that **touch, rm , mkdir, and rmdir** don't use pipe to act on multiple files, folders or applications; that means such commands take multiple arguments natively, but you will come across some command that only take an argument; that is when pipe is handy. That is, you will use pipe to pass an argument to such command one after the other. **Start** is an example of commands that take an argument. That is why we always use pipe to pass many arguments to it consecutively.

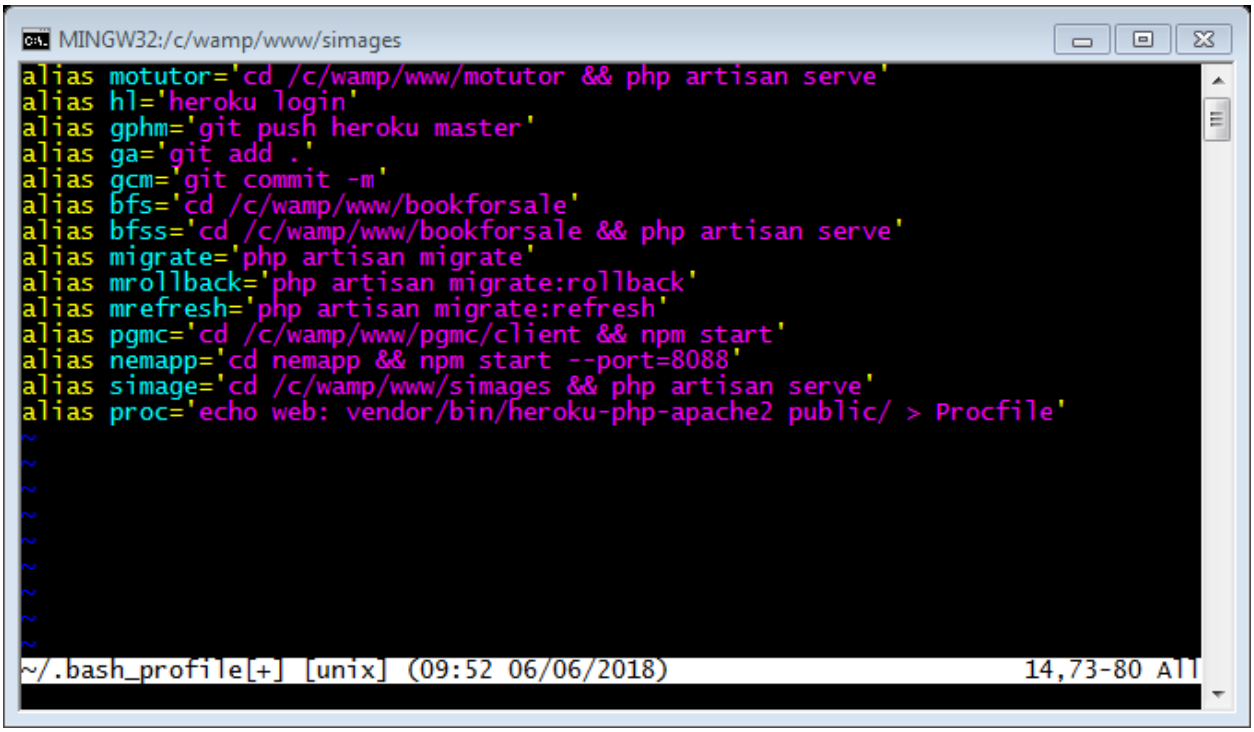
&&

&& is used to join multiple actions to be carried out together. For example, you can create a new folder and delete its old version at the same time. E.g, run mkdir School && rmdir Class && cd School && touch students.js. Did you know the meaning of the above string of commands? It creates a folder called School in the present working directory; it deletes Class folder, opens School folder and inserts a file named students.js into it. It makes running multiple actions with a string of commands at once possible. That is very handy if you have already planned you application structure.

Alias

My full name is Ogundiran Ayobami Akeem; Oh! That is f**king long. You won't always want to call that f**king long full name all the time. Yeah! That is why most people call me 'Ay'. That is short and easy to write. While using command

line, you would need to write long commands repetitively. In this case, alias would save you a lot of typing. Below image shows some of the aliases I used in some of my related applications.



How do we create and use alias?

It is not rocket science! Just run vim ~/.bash_profile or any name you prefer. Or navigate to localdisc(C:)/Users/your-username. In your username folder, create a file and save it with .bash_profile. You can then open it in any text editor to add or remove aliases. But if you go with Vim, it will open a new file, in your console, wherein you will put your aliases. In the file, just write your aliases. It is always in this format: alias Name= 'A string of command(s)'. For instance, alias myapp="cd /c/wamp/www/myapp".

To use the alias, run **source ~/.bash_profile** in your console; then use your alias as in: myapp. Yeah! That is cool. That is it for alias.

> and >> (Redirection)

> is a redirection flag that is used to pass a result of such action into another file.

If you have used heroku to host laravel, you may get what > does. E.g, **echo web: vendor/bin/heroku-php-apache2 public/ > Procfile.** This string of command create a file calls Procfile and insert **web: vendor/bin/heroku-php-apache2 public /** into into it. That is how > is used. You should be aware that if Procfile exists before running the above command, Procfile with will be overridden. What if you don't want to override the Procfile but you want to append another result or information to it; how will you do it?

This question takes us to the use of **>>.** It works just like **>** but it is different in the sense that it will not override the file if it exists; it will only add the result to the end of the file.

To bring this part to an end, be informed that Google is your friend and it can help you get answers to things you may not know. It is not a crime if you don't remember or know something, but it is bad not use all you know or remember practically. Use all the cool stuff you learn in this section before you continue.

Lest I forget, **rm –rf foldername** is used to forcefully remove a folder when it is not empty. Normally, rmdir foldername can't delete foldername if it is not empty to save you from losing important files. To confirm you know what you are doing, git makes it possible to forcefully remove such file recursively…. rm: remove –: flag identifier r: recursive f: force.

Note: don't use rm –rf foldername except you know what you are doing. And don't ever try it with * if you don't understand what it does because you may end up destroying all the soft wares on your computer. Be warned! Be careful.

Version Control (Git & GitHub)

As a developer, you would need to collaborate, contribute or work remotely; then you will need to use Version Control. Stop! Version control is more than that, but such simple description gives a proper insight about it. It enhances tracking, storage and maintenance of software source code. Who cares?

Let's get into its uses. If you care about knowing it theoretically, you can google search it.

I have explained how to use some commands in git; now, we will focus on how to create, store, contribute and maintain our code online on github.com.

Let's assume you have written a software on your computer and you want your friend in another country to review your code, so you need to store it on github. Mind you, there are other similar platforms to github but once you can use github, using others won't be a problem.

You have written the software; you just have to initiate git in it so that you can make it available to your friend.

Makes sure you download git HERE. Then you need to set up git for use. To set up git, you need to navigate to the Terminal app (Hard Drive -> Applications -> Utilities -> Terminal) on OS X and on Windows, you need to launch the Git Bash app you just installed. Caveat! Don't use windows command prompt; that is different from git.

Let's create a root directory for your hello world application.

mkdir hello-world-app .

Then change your directory to the hello-world-app.

cd hello-world-app

Yeah! Let's get started by running: git init. Then, you will follow the messages prompted by the command you run. It will ask you some questions (JAMB QUESTIONS – Nigerian slang, meaning too basic questions), answer them as you like. Wait! What if I make some mistakes while answering the questions; can I still correct them? Yes, the git init will create a package.json for you and you can edit the answers you supply in the package.json file.

It is good to explain one or two things to people that might see your hello-world-app. We will include that in our README file. It is time to create README inside the hello-world-app.

touch README.md

Now is the time to prepare our application to go github. But before then, you need to know some basic stuff.

Workflow

The local repository we work with is made up of three components or "trees" all maintained by git. The first one is the Working Directory which contains the actual files. The second one is the Index which acts as a staging area and finally the HEAD which points to the last commit you've made.

add

After you might have written some description in the README file, you would need to add(propose) it to the index area (staging area). So, we will use **git add**

git add <filename>: **git add README.md**

What if you have worked on many files and you have to add/stage all of those files? It would be more labourious to stage or add them one after the other. Then, you can use below command to add (propose) all files you have worked on once:

git add *

Or

git add .

Once you have staged or added your files to the Index directory as a proposal; you need to accept the proposal. That is what committing your files does.

To actually commit the changes in README file use

git commit -m "Commit message"

Now the file is committed to the HEAD, but not in your remote repository (on github or somewhere else) yet.

Sometimes, you will need to know the files you have committed for some reasons. If that is the case, you need to run

git status

Pushing changes

You have done some things, yet you still need to do some more. The README changes have been committed to the HEAD in your local copy repository. From there, you will need to push it to the remote repository. So, execute

git push origin master

Opps! If you are pushing for the first time without configuring some stuff, you will bump into an error. Let's configure our console to recognize you as a user .

git config --global user.name "Your Name Here"

Example: git config --global user.name "Ayobami Ogundiran"

You can put in any name you like. Thereafter, you'll need to input your email and that email must be the email you used when joining GitHub:

git config --global user.email "your_email@youremail.com"

Example: git config --global user.email "myemail@yahoo.com"

The local repository has not been connected to any remote repository on a server and the repository is not cloned from an existing one. Therefore, git push remote master will not do what you want. Doing that means you want to push it to nowhere. Is that possible?

Let's connect it to a remote repository:

git remote add origin <server>

Example: **git remote add origin https://www.github.com/codingnninja/hello-world-app.git**

Now, execute push command:

git push remote master

Barabim baraboom boom! You have pushed you first repository to github.

Note: You must have registered with github before you can push to github.

Wait! There is a problem!

This stupid git will always ask you to enter your github credentials/password any time you want to push to the server. To stop that, if you use window, download THIS and install it to cache you credentials.

Or you can command the console to cache it for you as in:

git config --global credentials.helper cache

But the console will only cache your credentials for just 15minutes by default. To set any timeout you prefer, execute

Git config --global credentials.helper 'cache --timeout=5000'

It might be necessary to delete your credentials from the helper; then, use

git credential-osxkeychain erase host=github.com protocol=https

Now you're all set to actually use git and GitHub!

The issue surfaces because we use https protocol to connect to our repository. Otherwise, we would have set up SSH to avoid such issue. Anyway, you can learn that on your own because it is not within the scope of this tutorial.

git clone

Any time you want to copy a remote repository to you computer, execute

git clone <url> : **git clone https:github.com/codingnninja/yeah.git**

Branching

“Branches are used to develop features isolated from each other. The master branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion”. I can’t say it better.

Create a new branch named "Dev" and switch to it using

git checkout -b Dev

To switch back to maste, execute **git checkout master**. If there is a need to delete the branch, you can run

git branch -d Dev

To make your branch available to others, you have to push it to the remote server as in:

g**it push origin <branch>**

If you like to sync you local branch to it version in the remote server so that you can just do git push origin, execute

git push -u origin <branch>

Replace local changes

In case you did something wrong, you can replace local changes using the command

git checkout -- <filename>

This command replaces the changes in your working tree with the last content in HEAD. Changes already added to the index, as well as new files, will be kept.

update & merge

If you want to fetch and replace your local copy with the newest from the server because you think what you have on your computer has been messed up, run

git fetch origin

git reset --hard origin/master

If you want to merge another branch into your active branch (e.g. master), use

git merge <branch>

Sometimes, merging will result in conflicts and you will need to correct the merge as you like and then run

git add <filename>

To indicate you have resolve the conflict to prepare it for merging. It is also good to preview branches you want to merge before merging them. You can use below command to preview them to check differences

git diff <source_branch> <target_branch>

Note: To carry out adding interactively run **git add -i**

To update your local repository to the newest commit, execute

git pull

This command will fetch and merge remote changes to your local copy. It is the combination of both fetch and merge, and it is always used in most cases for many reasons.

Tagging

You can create tag for releases of your software as in below:

git tag 1.0.0 1ce1b2e1d4

1.0.0 stands for the first release of you software and **1ce1b2e1d4** stands for commit id which can be got through git log.

Logging

Through log, you have access to the history of you commit. Just run

git log

If several people work on the repository, you can get logs by other users by specifying their names.

git log --author=ayobami

See only which files have changed:

git log --name-status.

For more, see git log --help

How to contribute to an open source repository?

Visit the repository you want to contribute to on github, click on fork button and a copy of the repository will be copied for you.

After that, get the url of the forked repository created for you to clone it on your computer as in.

git clone https:github.com/codingnninja/to-contribute-repo.git

The next thing to do is to connect the original project in order to get update from there when available. So, execute

git remote add upstream https:github.com/originalDeveloper/original-repo.git

If you have committed any changes and you want to push to you remote repository, just run

git pull uptsream master && git push origin master

Note: the command above will fetch and merge update from the original project to your local copy and push your changes to the forked version you have in your account.

If you create another branch, you can push as in **git push -u origin <branch>.** The –u means you want to link your local branch to the remote version of it. So, any time you want to push in the future you can just run git push origin and you are good to go. With this, you have started what is called PR. Okay, let's finish it then.

Go back to github and look for compare and pull button or link and complete the process it takes you through. Then, you are through with your PR. Just be waiting for whether you PR would be accepted or rejected. Nice one!

.gitignore

Gitignore is always added to the root directory to ignore some files or folder while pushing to the server. Sometimes, some folders or files are not needed to be pushed to github. So, we use .gitignore file to command git to ignore them.

.gitignore can be added to each folder and can be made global on ones computer. For local use within a directory, just touch .gitignore in the root director and add files you want to ignore as in:

If there is a root directory called root and it contains test folder and .gitignore, to command git to ignore test folder, just add /test (path to test) in .gitignore. To ignore all files that end with .js, just add *.js inside .gitignore or you can just copy the path to a file you want to ignore to it.

To make .gitignore global to all the projects on your computer, just run touch ~/.gitignore_global and then execute below to command .gitignore file you created to the global space of your console.

git config –global core.excludesfile ~/.gitignore_global

Mind you, git may not ignore some files after you have added them to .gitignore; that happens because an old version without the .gitignore command has been cached by the console, so run below command to remove it.

git rm –cached Filename

The next time you push, git will not touch the private part of your application in public. Wow! Thanks for reading all along. **Power to your brain!**

Meet me once again in another material. My name remains Ogundiran Ayobami; I love writing and slapping my keyboard until great things happen. Thanks for your time. Follow me on twitter @ayovision. If you need a software developer for your project, message me on twitter or Facebook. Bye for now.