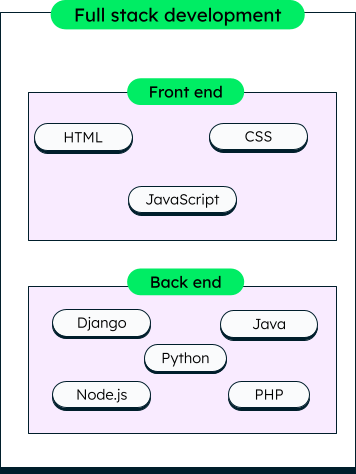
|  |
| --- |
|  |
| Full Stack Development |
| **(20A05703A)** |
| **Unit - I** |
| **Regulation – R20** |

**Full Stack Development**

1. What is Full Stack Development?

* Full stack development refers to the end-to-end application software development
* It refers to the development of both front end(client side) and back end(server side) portions of web application.
* Full stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database and debugging of web applications or websites.



# Technology Related to Full Stack Development

* 1. **Front-end Development**
* It is the visible part of website or web application which is responsible for user experience. The user directly interacts with the front end portion of the web application or website.

# Front-end Technologies:

The front end portion is built by using some languages which are discussed below:

* HTML
* CSS
* JavaScript
  + 1. Front End Libraries and Frameworks
       - [AngularJS](https://www.geeksforgeeks.org/category/web-technologies/angular-js/)
       - [React.js](https://www.geeksforgeeks.org/react-js-introduction-working/)
       - [Bootstrap](https://www.geeksforgeeks.org/bootstrap-tutorials/)
       - [jQuery](https://www.geeksforgeeks.org/jquery-tutorials/)
       - [SASS](https://www.geeksforgeeks.org/sass/)
       - Express.js, etc…
    2. Back-end Technologies
       - It refers to the server-side development of web application or website with a primary focus on how the website works.
       - It is responsible for managing the database through queries and APIs by client-side commands. This type of website mainly consists of three parts front end, back end, and database.
       - The back end portion is built by using some libraries, frameworks, and languages which are discussed below:
         * [PHP:](https://www.geeksforgeeks.org/php/)
         * [C++](https://www.geeksforgeeks.org/c-plus-plus/)
         * [Java](https://www.geeksforgeeks.org/java/)
         * [Python](https://www.geeksforgeeks.org/python-programming-language/)
         * [Node.js](https://www.geeksforgeeks.org/introduction-to-nodejs/)
         * [Database](https://www.geeksforgeeks.org/dbms/)
         * [MongoDB](https://www.geeksforgeeks.org/mongodb-an-introduction/)
         * [Sql](https://www.geeksforgeeks.org/sql-tutorial/)

# Popular Stacks

* MEAN Stack: MongoDB, Express, AngularJS and Node.js.
* MERN Stack: MongoDB, Express, ReactJS and Node.js
* Django Stack: Django, python and MySQL as Database.
* Rails or Ruby on Rails: Uses Ruby, PHP and MySQL.
* LAMP Stack: Linux, Apache, MySQL and PHP.

**Case Study:**

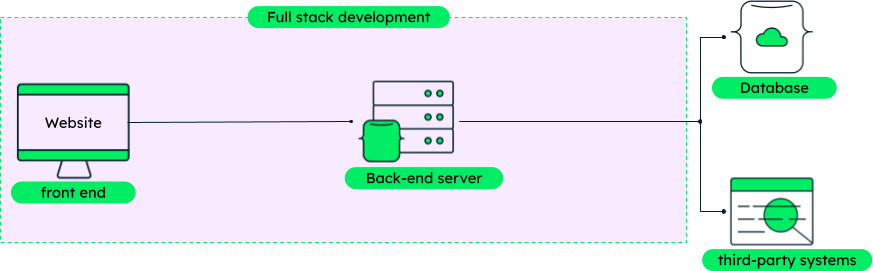
# Industry definition:

* Full Stack Developer is an engineer who can work on different levels of an application stack. The term stack refers to the combination of components and tools that make up the application. The components could be in the front-end or the back-end of the system.

Consider a retail website. Users can browse or purchase specific items, delete or add items in cart, change their profile, and do many other things. All these actions require a front-end user interface (UI), as well as some business logic, written in the back-end.

* The website UI can be built using various, *front-end technologies* like HTML, CSS, Javascript.
* The back end is written in *programming languages* like Java or Python. Further, a good web application would need scalability, event handling, and routing, which are usually handled by *libraries and frameworks* like SpringBoot or Django.
* The back end also consists of logic that can connect the application to other services and databases. For example, all the user and transaction data is stored in a database through specific *drivers* handled on the back end.

A full stack developer is one who can single-handedly implement both the front-end and back-end workflows, like placing the order or changing the user profile.



# What is a full stack developer and what do they do?

* Full stack developers must have knowledge of an entire [technology stack](https://www.mongodb.com/basics/technology-stack), i.e., the set of technologies that are used to build an end-to-end application quickly and efficiently.
* For example, if they want to build an application using the [MEAN stack](https://www.mongodb.com/mean-stack), they should know how to work with [MongoDB](https://www.mongodb.com/atlas/database), Express, Angular and Node.
* Full stack developers should be able to judge whether the selected technologies are the right choice for their project during the early phases. Some responsibilities of a full stack developer are to:
* Help in choosing the right technologies for the project development and testing both on the front end and the back end.
* Write clean code across the stack by following the best practices of the tools used.
* Be up to date with the latest technologies and tools to make the best technology usage decisions.

# Full stack development advantages

There are many advantages of hiring full stack developers for web application development:

* Complete ownership and understanding of the project
* Saves both project time and cost, and enhances productivity
* Faster bug fixing due to knowledge of complete system
* Easy knowledge transfer to other team members
* Better division of work amongst team members

**Unit – I**

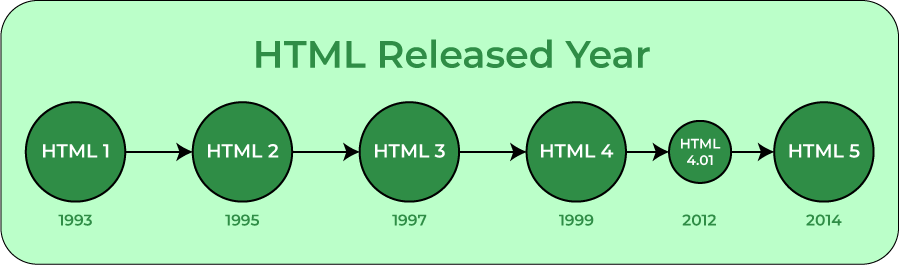
**Web Development Basics**

Web Development Basics - HTML & Web servers Shell - UNIX CLI Version control - Git & Github HTML, CSS

# HTML:-

* HTML is markup language commonly used to create web pages.
* **Hyper Text:** Hyper Text simply means "Text within Text." A text has a link within it, is a hypertext.
* **Markup Language**: It provides a way to describe the structure of text and graphics on a web page.
* HTML is developed and maintained by W3C.
* It is derived from Standard Generalized Markup Language (SGML).
* In HTML, the term signifies the navigation from one to another.

# HTML Versions:



* HTML 1.0:
  + Only 22 elements
* HTML 2.0:
  + New features, such as tables, image maps, file upload.
* HTML 3:
  + New elements FIG, LIST, LINK, and NOTE added.
* HTML 4:
  + New elements for style sheets, scripts, frames, embedded objects, complex tables and forms.
* HTML 5
  + It introduces the new elements such as AUDIO, VIDEO

# HTML DOCUMENTS STRUCTURE

* Html used predefined tags and attributes to tell the browser how to display content, means in which format, style, font size, and images to display. Html is a case insensitive language.
* Case insensitive means there is no difference in upper case and lower case (capital and small letters) both treated as the same, for r example ‘B’ and ‘b’ both are the same here.

# There are generally two types of tags in HTML:

1. Paired Tags: These tags come in pairs. That is they have both opening (< >) and closing (</ >) tags.
2. Empty Tags: These tags do not require to be closed.

Below is an example of a (<b>) tag in HTML, which tells the browser to bold the text inside it.

***<b> IV CSE </b>***

**Tags and attributes:** Tags are individuals of html structure, we have to open and close any tag with a forward slash like this <h1>

</h1>.

There are some variations with the tag some of them are self-closing tag which isn’t required to close and some are empty tag where we can add any attributes in it.

Attributes are additional properties of html tags that define the property of any html tags. i.e. width, height, controls, loops, input, and autoplay.

These attributes also help us to store information in meta tags by using name, content, and type attributes. Html documents structured mentioned below:

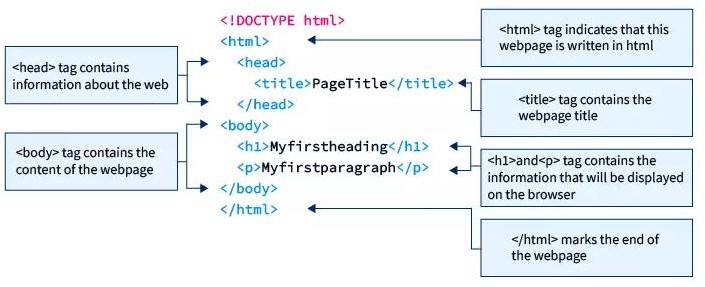
# An HTML document is divided into two parts:

1. **Head part**:-

* The title and metadata of a web document are contained in the head element.

1. **Body part**:-

* The body element includes the information that you wish to display on a web page.



# Basic Structure of HTML

An HTML document's basic structure consists of 5 elements:

* <!DOCTYPE>
* <html>
* <head>
* <title>
* <body>

# <!DOCTYPE>

* The tag in HTML is used to inform the browser about the HTML version used in the web page.
* It is referred as the **document type declaration (DTD)**. It is not really a tag/element but rather an instruction to the browser regarding the document type.
* It is a null element that does not have a closing tag and must not contain any content.

**Syntax**

<!DOCTYPE html>

# <html>

* The <html> tag in HTML is used to specify the root of HTML and XHTML pages. The <html> tag informs the browser that this is an HTML document. It is the second outer container for everything in an HTML document, followed by the tag. The <html> tag requires a beginning and ending tag.

**Syntax**

<!DOCTYPE html>

<html>

...

</html>

**1.3.3 <head>**

* The <head> tag in HTML is used to contain metadata (**data about data**). It is used between the<html> and <body> tags.
* The head of an HTML document is a section of the document whose content is not displayed in the browser when the page loads. It only contains HTML document metadata, which specifies information about the HTML document.
* Depending on our needs, an HTML head might contain a lot of metadata information or can have very little or no metadata information.
* The document title, character set, styles, links, scripts, and other meta information are defined by metadata.

The following is a list of metadata tags:

* <title>
* <style>
* <meta>
* <link>
* <script>
* <base>

# Syntax

<!DOCTYPE html>

<html>

<head>

...

</head>

</html>

# 1.3.4 <title>

* This <title> tag in HTML displays the title of a web page and can help in higher rankings in search results if appropriate keywords are included.
* ​





The most significant meta element to add to our webpage is the <title> element.

It gives a relevant title to the full HTML content. It appears at the top of the browser window and gives the webpage a fitting name when saved as a favorite or bookmark.

A solid web page title will guarantee a higher rank in search results. Thus, we must constantly utilize relevant keywords.

* The <title> element must be positioned between the <head> element, and there can only be one title element per document.

**Syntax**

<!DOCTYPE html>

<html>

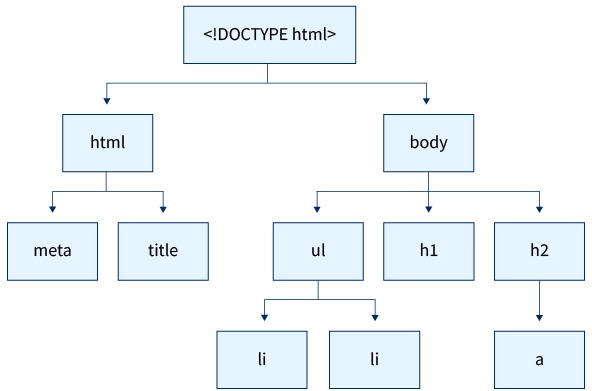
<head>

<title> ... </title>

</head>

</html>

# 1.3.5 <body>



* The <body> tag in HTML specifies the main content of an HTML document that appears on the browser. It can contain headings, text, paragraphs, photos, tables, links, videos, etc.
* The <body> tag must come after the <head> tag, or it must be inserted between the </head> and </html> tags. This tag is essential for all HTML documents and should only be used once throughout the document.

**Syntax**

<!DOCTYPE html>

<html>

<head>

<title>Body Tag</title>

</head>

<body>

<h1>...</h1>

<p>...</p>

</body>

</html>

# Characteristics of HTML

* Easy to understand
* Flexibility
* Linkable
* Limitless features
* Support
* Not a Programming Language
* Language Support

# Advantages of HTML

* HTML is easy to learn, easy to apply and it’s totally free you will just need a text editor and a browser.
* HTML is supported by all the browsers and it is the most friendly search engine.
* HTML can easily integrate with other languages and is easy to develop.
* It is the basic of all programming languages and the lightest language ever.
* In HTML, the display changes frequently depending on the window size or the device size making it comfortable to read by the user.

# Disadvantages of HTML

* HTML can be used to create only static Web-page, it cannot create dynamic web-page.
* There is a lack of security in HTML.
* Creating a simple Web-page required so many tags.
* HTML language is not centralized i.e. all the web pages that are connected, you have to design them separately else need to use CSS.
* HTML becomes complex when you try to create a huge website.

# Web Servers Shell:-

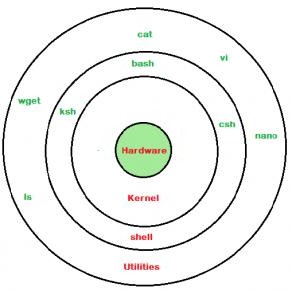
* If we are using any major operating system, we are indirectly interacting with the **shell**. While running Ubuntu, Linux Mint, or any other Linux distribution, we are interacting with the shell by using the terminal.
* we have to get familiar with the following terminologies:
  + Kernel
  + Shell
  + Terminal

# What is Kernel?

* + - * The kernel is a computer program that is the core of a computer’s operating system, with complete control over everything in the system. It manages the following resources of the Linux system –
        + File management
        + Process management
        + I/O management
        + Memory management
        + Device management etc.

# What is Shell?

* + - * A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human- readable commands from users and converts them into something which the kernel can understand.
      * It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

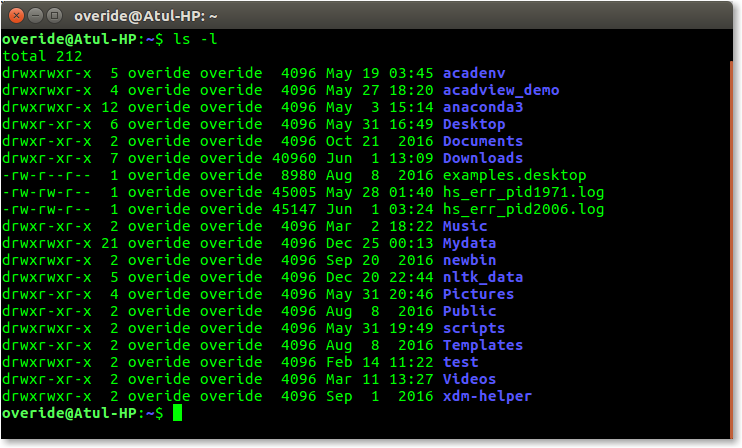


Shell is broadly classified into two categories –

* + - * + Command Line Shell
        + Graphical shell

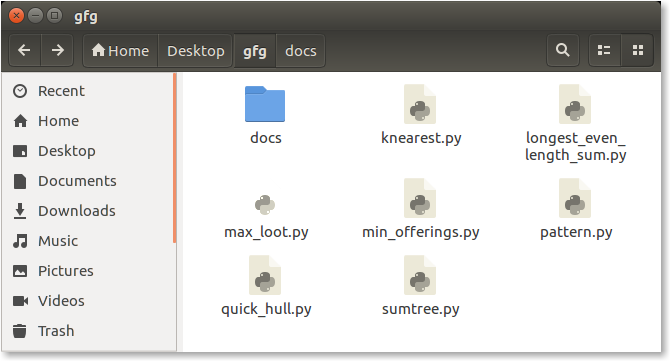
# Command Line Shell

* + - * Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as “cat”, “ls” etc. and then it is being executed. The result is then displayed on the terminal to the user.



# Graphical Shells

* + - * Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as a good example which provides GUI to the user for interacting with the program. Users do not need to type in commands for every action.



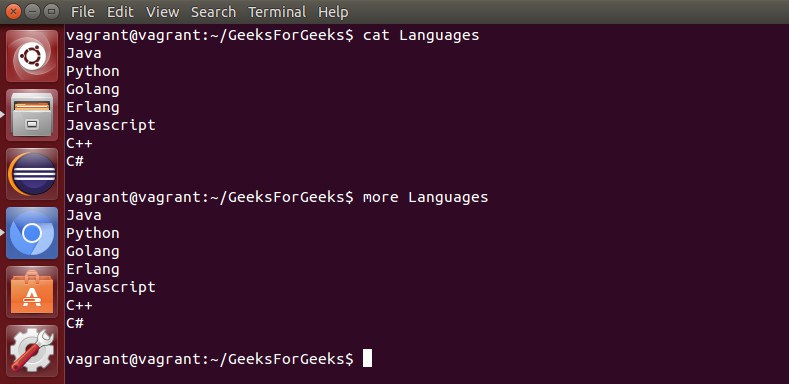
# What is a terminal?

A program which is responsible for providing an interface to a user so that we can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface.

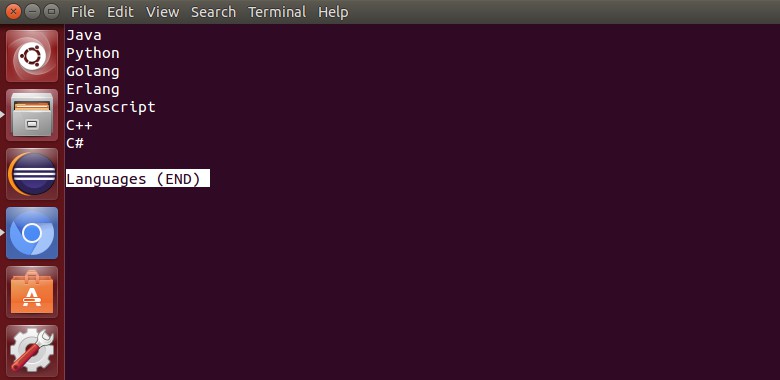
# Basic Shell Commands in Linux

Shell accepts human-readable commands from the user and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

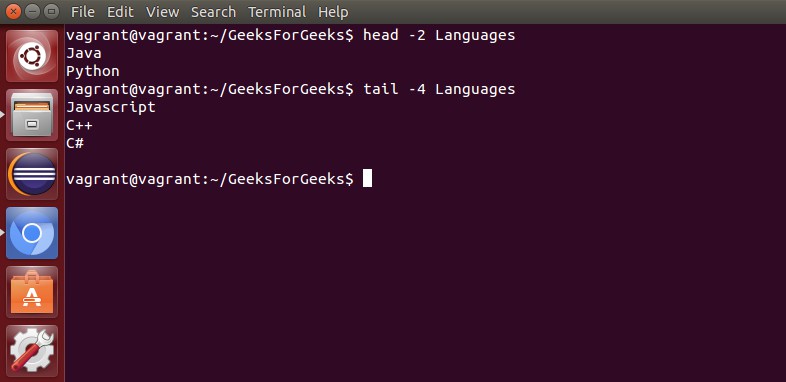
1. **Displaying the file contents on the terminal:**
   * [cat](https://www.geeksforgeeks.org/cat-command-linux-examples/): It is generally used to concatenate the files. It gives the output on the standard output.
   * [more](https://www.geeksforgeeks.org/more-command-in-linux-with-examples/): It is a filter for paging through text one screenful at a time.



* + [less](https://www.geeksforgeeks.org/less-command-linux-examples/): It is used to viewing the files instead of opening the file. Similar to more command but it allows backward as well as forward movement.



* + [head](https://www.geeksforgeeks.org/head-command-linux-examples/) : Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.
  + [tail](https://www.geeksforgeeks.org/tail-command-linux-examples/) : Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

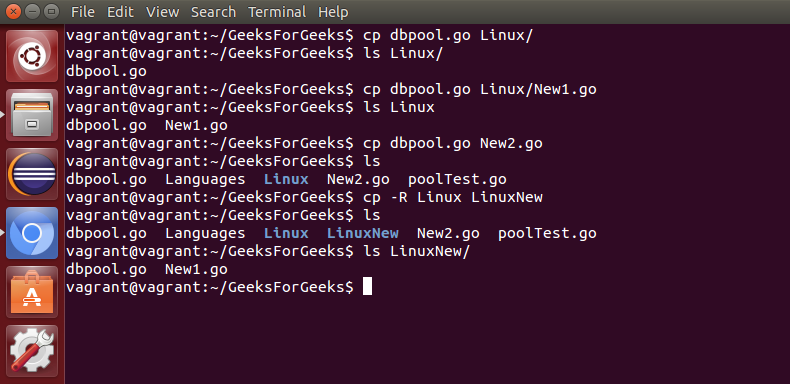


# File and Directory Manipulation Commands:

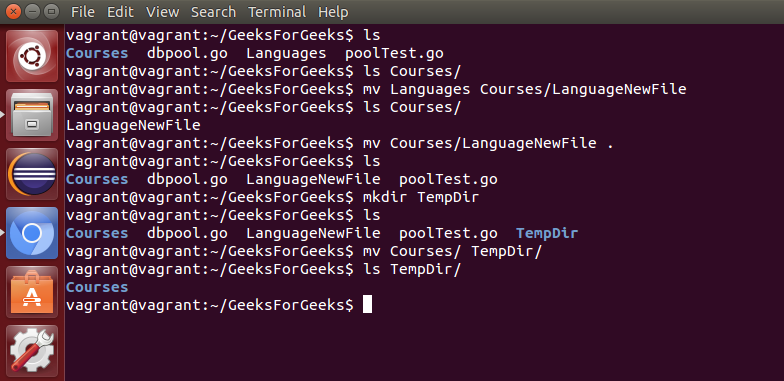
* + [mkdir](https://www.geeksforgeeks.org/mkdir-command-in-linux-with-examples/) : Used to create a directory if not already exist. It accepts the directory name as an input parameter.



* + [cp](https://www.geeksforgeeks.org/cp-command-linux-examples/) : This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.



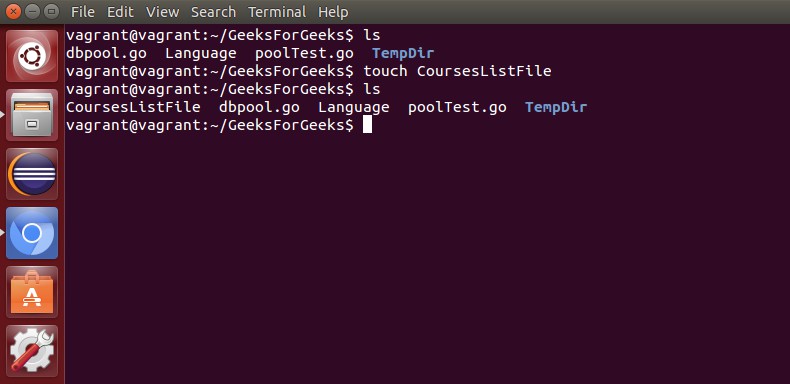
* + [mv](https://www.geeksforgeeks.org/mv-command-linux-examples/) : Used to move the files or directories. This command’s working is almost similar to cp command but it deletes a copy of the file or directory from the source path.



* + [rm](https://www.geeksforgeeks.org/rm-command-linux-examples/) : Used to remove files or directories.

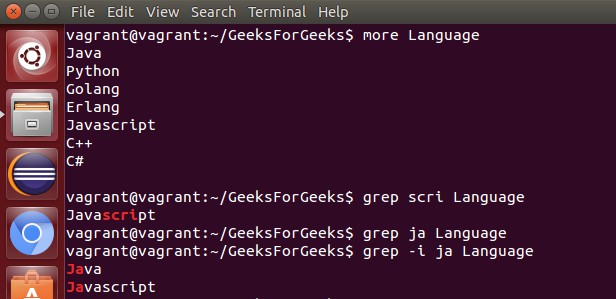


* + [touch](https://www.geeksforgeeks.org/touch-command-in-linux-with-examples/) : Used to create or update a file.

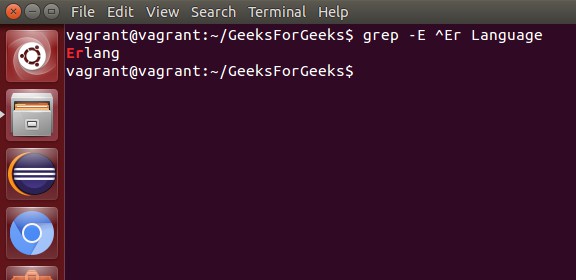


3). Extract, sort, and filter data Commands:

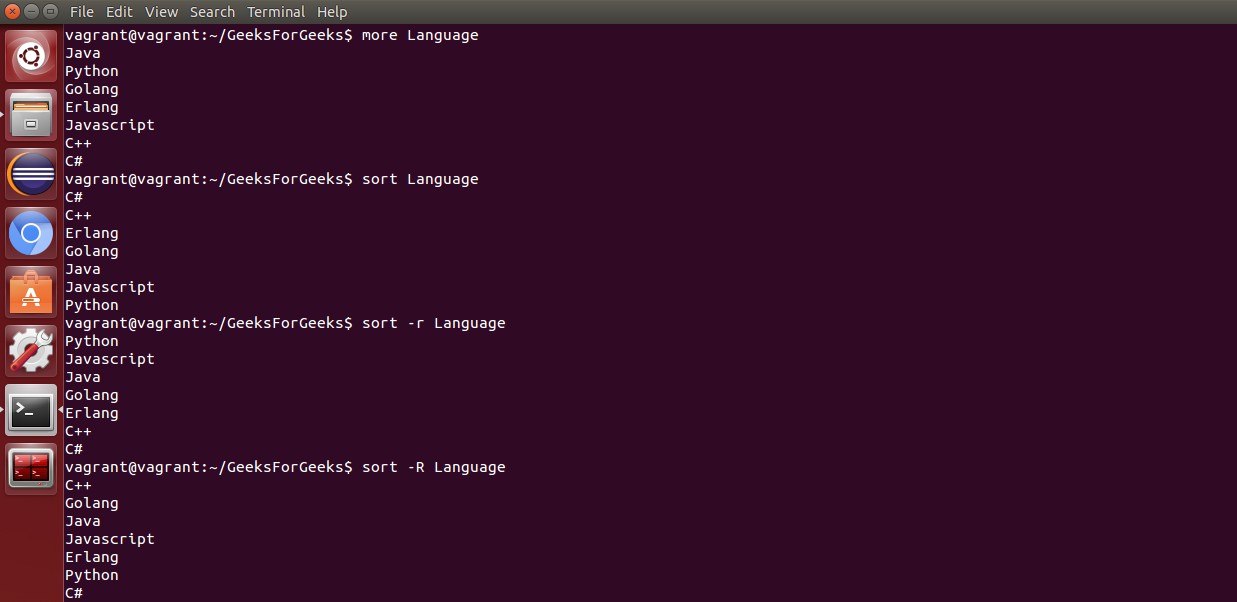
* [grep](https://www.geeksforgeeks.org/grep-command-in-unixlinux/) : This command is used to search for the specified text in a file.



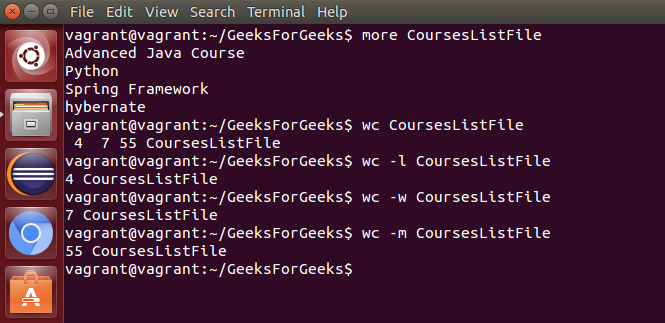
* grep with Regular Expressions: Used to search for text using specific regular expressions in file.



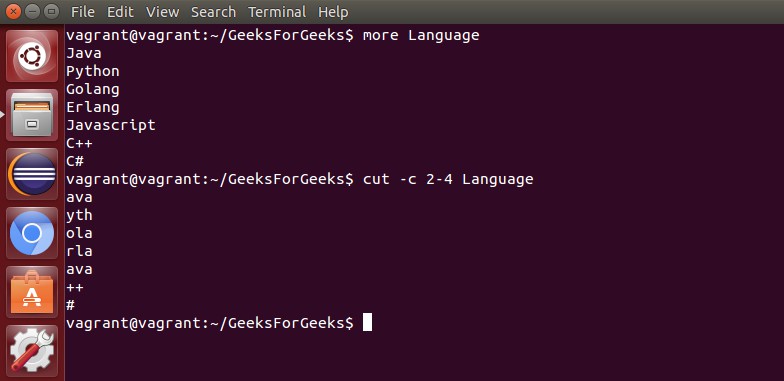
* [sort](https://www.geeksforgeeks.org/sort-command-linuxunix-examples/) : This command is used to sort the contents of files.



* [wc](https://www.geeksforgeeks.org/wc-command-linux-examples/) : Used to count the number of characters, words in a file.



* [cut](https://www.geeksforgeeks.org/cut-command-linux-examples/) : Used to cut a specified part of a file.



# Basic Terminal Navigation Commands:

* 1. [ls](https://www.geeksforgeeks.org/practical-applications-ls-command-linux/) : To get the list of all the files or folders

**Syntax: ls**

* 1. [cd](https://www.geeksforgeeks.org/cd-command-in-linux-with-examples/): Used to change the directory.

cd **<directory** name**>**

cd IVCSE

* 1. [du](https://www.geeksforgeeks.org/du-command-linux-examples/): Show disk usage.

du [options] [directory/file] du /home/mandeep/test

* 1. [pwd](https://www.geeksforgeeks.org/pwd-command-in-linux-with-examples/): Show the present working directory. pwd
  2. [man](https://www.geeksforgeeks.org/man-command-in-linux-with-examples/): Used to show the manual of any command present in Linux.

$ man printf

* 1. [rmdir](https://www.geeksforgeeks.org/rmdir-command-in-linux-with-examples/): It is used to delete a directory if it is empty. rmdir <options> <directory>
  2. [locate:](https://www.geeksforgeeks.org/locate-command-in-linux-with-examples/) It is used to locate a file in Linux System locate filename
  3. [echo:](https://www.geeksforgeeks.org/echo-command-in-linux-with-examples/) This command helps us move some data, usually text into a file.

echo string

* 1. [df:](https://www.geeksforgeeks.org/df-command-linux-examples/) It is used to see the available disk space in each of the partitions in your system.

df

* 1. [tar:](https://www.geeksforgeeks.org/tar-command-linux-examples/) Used to work with tarballs (or files compressed in a tarball archive)

# File Permissions Commands: Different Type of Permission:

* Read
* Write
* Execute

# File Permission Types:

* User
* Group
* Other
  1. [chown](https://www.geeksforgeeks.org/chown-command-in-linux-with-examples/) : Used to change the owner of the file. Syntax:

chown [**OPTION**]... [OWNER][:[**GROUP**]] FILE...

# Eg: chown owner\_name file\_name

* 1. [chgrp](https://www.geeksforgeeks.org/chgrp-command-in-linux-with-examples/) : Used to change the group owner of the file. Syntax:

chgrp [OPTION]... GROUP FILE...

Eg: chgrp javatpoint Demo1.txt chgrp javatpoint Newdirectory

* 1. [chmod](https://www.geeksforgeeks.org/chmod-command-linux/) : Used to modify the access/permission of a user. Syntax:

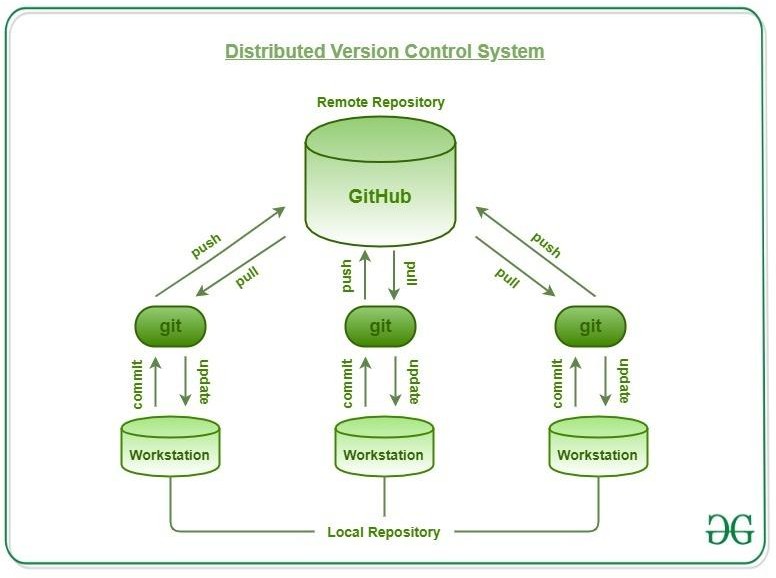
chmod <options> <permissions> <file **name**>

**chmod u+rwx [file\_name] chmod go-w [file\_name] chmod u+rw,go+r [file\_name]**

# Unix CLIVersion Control

**Git:-**

* Git is one of the way of implementing the idea of version control. It is Distributed Version Control System(DVCS)
* Git allows a team of people to work together, all using the same files. And it helps the team cope with the confusion that tends to happen when multiple people are editing the same files.
* Version control, also known as source control, is the practice of tracking and managing changes to software code.
* Version control systems are software tools that help software teams manage changes to source code over time.
* As development environments have accelerated, version control systems help software teams work faster and smarter.
* They are especially useful for [DevOps](https://www.atlassian.com/devops/what-is-devops) teams since they help them to reduce development time and increase successful deployments.



* Having a distributed system, Git allows the users to work simultaneously on the same project, without interfering with others’ work.
* When a particular user gets done with their part of the code, they push the changes to the repository and these changes get updated in the local copy of every other remote user which pulls the latest copy of the project.

## History

* Until April 2005 Linus Torvalds was using BitKeeper for version control of the Linux Kernel development.
* He had a large number of volunteer developers working on the Linux Kernel and their contributions had to be managed.
* BitKeeper was a nice tool for managing the enormous contribution by the developers.
* Linux knew he had to act fast to replace the version control system that he knew and loved so he took a working vacation to decide on what to do as the current free-to-use version control systems could not solve his problems at the time.
* The result of his vacation was the birth of a new version control system named **Git**.

## Advantages of Git

1. **Free and Open Source:** Git is a free and Open source software system with which the users and programmers can edit, modify or reuse the software’s source code. It gives developers the opportunity to improve
2. **Instant Backup:** Data can be instantly retrieved as there are several copies available.

## Efficient and Low requirements

1. **Staging Area:** This is an intermediate area where commits can be formatted and reviewed before completing the commit. We can manage which change is needed for which version of the file and stage them for different commit commands.

## What is a GIT Repository?

* + Repositories in [GIT](https://www.geeksforgeeks.org/git-lets-get-into-it/) contain a collection of files of various different versions of a Project.
  + It contains the collection of the files as well as the history of changes made to those files. Repository in Git is considered as your project

folder. A repository has all the project-related data. Distinct projects have distinct repositories.

## Getting a Git Repository

There are two ways to obtain a repository. They are as follows:

* + - Create a local repository and make it as Git repository.
    - Clone a remote repository (already exists on a server). In either case, you can start working on a Git repository.

## Initializing a Repository

* + If you want to share your project on a version control system and control it with Git. Then, browse your project's directory and start the git command line (Git Bash for Windows) here. To initialize a new repository, run the below command:

**Syntax:**

**$ git init**

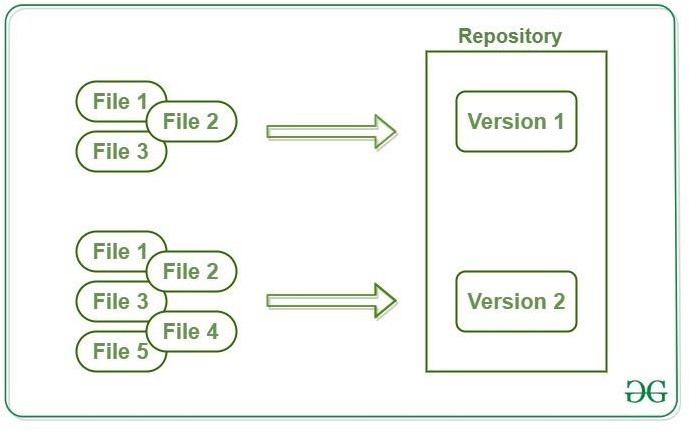
**Cloning an Existing Repository**

* + We can clone an existing repository. Suppose we have a repository on a version control system like subversion, GitHub, or any other remote server, and we want to share it with someone to contribute. The git clone command will make a copy for any user to contribute.
  + We can get nearly all data from server with git clone command. It can be done as:

## Syntax:

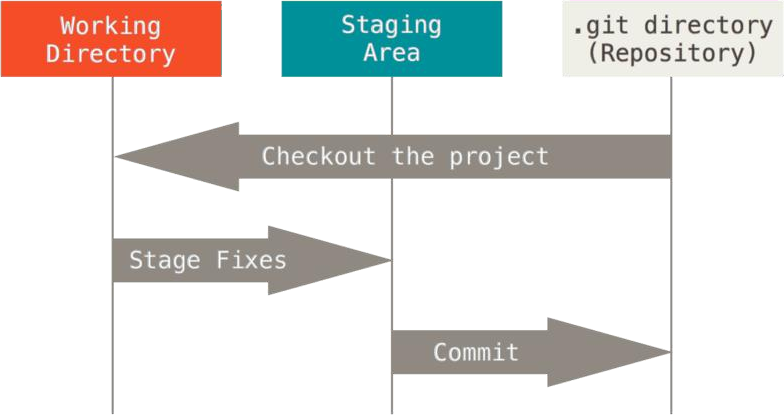
**$ git clone <Repository URL>**

* + These files are imported from the repository into the local server of the user for further updations and modifications in the content of the file.
  + A VCS or the [Version Control System](https://www.geeksforgeeks.org/version-control-systems/) is used to create these versions and store them in a specific place termed a repository. The process of copying the content from an existing Git Repository with the help of various Git Tools is termed **cloning**. Once the cloning process is done, the user gets the complete repository on his local machine.
  + Git by default assumes the work to be done on the repository is as a user, once the cloning is done. Users can also [create a new repository](https://www.geeksforgeeks.org/creating-repository-in-github/) or delete an existing repository.
  + To delete a repository, the simpler way is to just delete the folder containing the repository. Repositories can be divided into two types based on the usage on a server. These are:
    - **Bare Repositories:** These repositories are used to share the changes that are done by different developers. A user is not allowed to modify this repository or create a new version for this repository based on the modifications done.
    - **Non-bare Repositories:** Non-bare repositories are user-friendly and hence allow the user to create new modifications of files and also create new versions for the repositories. The cloning process by default creates a non-bare repository if any parameter is not specified during the clone operation.



* + Git has three main states that your files can reside in: **modified**, **staged**, and **committed**:

1. **Staged:** In this stage, the file is ready to be committed and is placed in the staging area waiting for the next commit.
2. **Modified/Dirty:** When the changes are made to the file i.e. the file is modified but the change is not yet staged.
3. **Committed** means that the data is safely stored in your local database.



* + The working tree is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
  + The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git parlance is the “index”, but the phrase “staging area” works just as well.
* The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

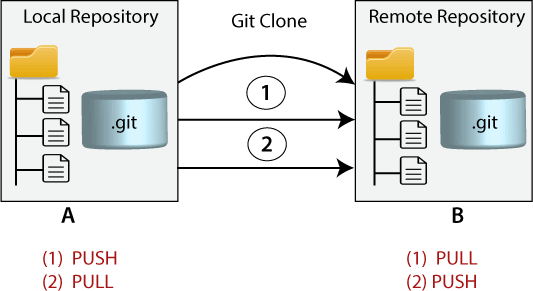
**Working with a Repository**

* + A [GIT](https://www.geeksforgeeks.org/git-lets-get-into-it/) repository allows performing various operations on it to create different versions of a project.
  + These operations include the addition of files, creating new repositories, committing an action, deleting a repository, etc.
  + These modifications will result in the creation of different versions of a project.
  + The operations are included in git:

1. Clone

Git Clone

1. Add
2. Commit
   * In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.



## Syntax:

**$ git clone <repository URL>**

**Adding to a Repository**

* + The git add command is used to add file contents to the [Index (Staging](https://www.javatpoint.com/git-index) [Area)](https://www.javatpoint.com/git-index).This command updates the current content of the working tree to the staging area. It also prepares the staged content for the next commit. Every time we add or update any file in our project, it is required to forward updates to the staging area.
  + The git add command is a core part of Git technology. It typically adds one file at a time, but there some options are available that can add more than one file at once.
  + The "index" contains a snapshot of the working tree data. This snapshot will be forwarded for the next commit.
  + The git add command can be run many times before making a commit. These all add operations can be put under one commit. The add command adds the files that are specified on command line.
  + We can add single or multiple files at once in the staging area. It will be run as:

## $ git add <File name> Different ways to use add command:

* + To add a specific list of files to the staging area.

## $ git add .

* + To add all files of the current directory to a staging area. ---

## $ git add --all

* + To add all text files of the current directory to staging area.

## $ git add \*.txt

* + To add all text files of a particular directory(docs) to staging area.

## $ git add docs/\*.txt

* + To add all files in a particular directory(docs) to staging area.

## $ git add docs/

* + To add text files of entire project to staging area.

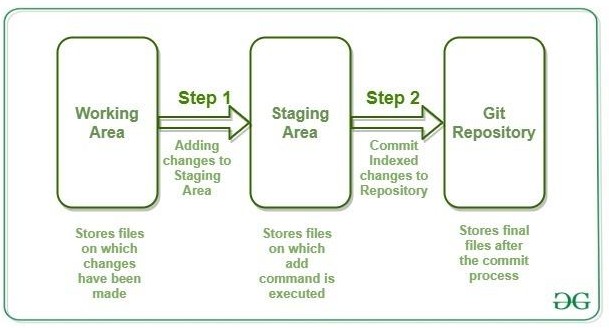
## $ git add “\*.txt”

* This committing process is done by the use of **git commit** command. This command commits the staged changes to the local repository.

## Syntax:

**$ git commit -m "Add existing file"**

* This commit command is used to add any of the tracked files to staging area and commit them by providing a message to remember.

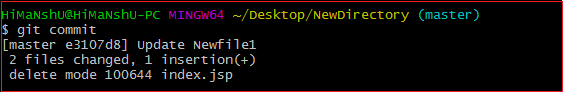


Git Commit

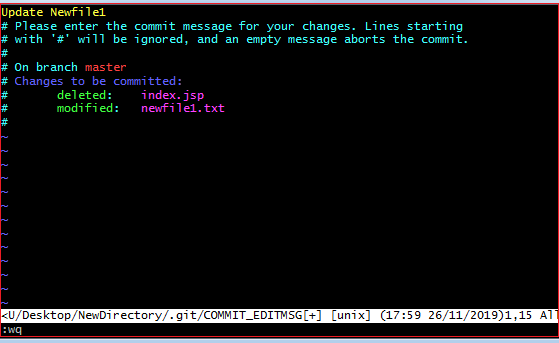
* + It is used to record the changes in the repository. It is the next command after the [git add.](https://www.javatpoint.com/git-add) Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area. A commit command is used to fetch updates from the staging area to the repository.
  + The staging and committing are co-related to each other. Staging allows us to continue in making changes to the repository, and when we want to share these changes to the version control system, committing allows us to record these changes.
  + Commits are the snapshots of the project. Every commit is recorded in the master branch of the repository. We can recall the commits or revert it to the older version. Two different commits will never overwrite because each commit has its own commit-id. This commit-id is a cryptographic number created by **SHA (Secure Hash Algorithm)** algorithm.
  + It will run as follows:

$ git commit

* + The above command will prompt a default editor and ask for a commit message. We have made a change to newfile1.txt and want it to commit it. It can be done as follows:



* + As we run the command, it will prompt a default text editor and ask for a commit message. The text editor will look like as follows:



**Working with Git:**

## Installation of Git

*Installation :* [*https://git-scm.com/download/win*](https://git-scm.com/download/win)

## Using Git with Command Line

* + To start using Git, we are first going to open up our Command shell.
  + For Windows, you can use Git bash, which comes included in Git for Windows. For Mac and Linux you can use the built-in terminal.

The first thing we need to do, is to check if Git is properly installed:

git --version

git version 2.30.2.windows.1

**Example**

If Git is installed, it should show something like git version X.Y

## Configure Git

* + Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

Example

git config --global user.name "jegan"

git config --global user.email "[jegan@acem.com](mailto:jegan@acem.com)"

## Creating Git Folder

Now, let's create a new folder for our project: Example

mkdir myproject cd myproject

## Initialize Git

* + Once you have navigated to the correct folder, you can initialize Git on that folder:

Example

git init

Initialized empty Git repository in /Users/user/myproject/.git/

## Git New Files

* + Git Adding New Files
  + You just created your first local Git repo. But it is empty.
  + So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created.
  + For this example, I am going to use a simple HTML file like this:

**Example**

<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

</head>

<body>

<h1>Hello world!</h1>

<p>This is the first file in my new Git Repo.</p>

</body>

</html>

And save it to our new folder as index.html.

## Example

ls

index.html ls will **list** the files in the directory. We can see that index.html is there.

Then we check the Git status and see if it is a part of our repo:

## Example

index.html

git status

On branch master

No commits yet Untracked files:

(use "git add ..." to include in what will be committed)

Now Git is **aware** of the file, but has not **added** it to our repository! Files in your Git repository folder can be in one of 2 states:

* + Tracked - files that Git knows about and are added to the repository
  + Untracked - files that are in your working directory, but not added to the repository

When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

## Git Staging Environment

* + One of the core functions of Git is the concepts of the Staging Environment, and the Commit.
  + As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.
  + **Staged** files are files that are ready to be **committed** to the repository you are working on.
  + For now, we are done working with index.html. So we can add it to the Staging Environment:

Example

git add index.html

* + The file should be **Staged**. Let's check the status:: Example

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage) new file: index.html

## Git Add More than One File

* + You can also stage more than one file at a time. Let's add 2 more files to our working folder. Use the text editor again.
    1. A README.md file that describes the repository (recommended for all repositories):

**Example**

# hello-world

Hello World repository for Git tutorial

This is an example repository for the Git tutoial on https://[www.w3schools.com](http://www.w3schools.com/)

This repository is built step by step in the tutorial.

## A basic external style sheet (bluestyle.css):

**Example**

body {

background-color: lightblue;

}

h1 {

color: navy; margin-left: 20px;

}

## update index.html to include the stylesheet:

**Example**

<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet" href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<p>This is the first file in my new Git Repo.</p>

</body>

</html>

* + Now add all files in the current directory to the Staging Environment:

git add --all

**Example**

* + Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

git status

On branch master No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage) new file: README.md

new file: bluestyle.css new file: index.html

**Example**

* + Now all 3 files are added to the Staging Environment, and we are ready to do our first commit.

**Note:** The shorthand command for git add --all is git add -A

## Git Commit

* + Since we have finished our work, we are ready move from stage to commit for our repository.
  + Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.
  + When we commit, we should **always** include a **message**.
  + By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

## Example

git commit -m "First release of Hello World!"

[master (root-commit) 221ec6e] First release of Hello World! 3 files changed, 26 insertions(+)

create mode 100644 README.md create mode 100644 bluestyle.css create mode 100644 index.html

* + The commit command performs a commit, and the -m "*message*" adds a message.
  + The Staging Environment has been committed to our repo, with the message:

"First release of Hello World!"

# GitHub:-

* + GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
  + It is a web-based service that hosts Git projects.
  + GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.
  + GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.
  + GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.
  + It offers both distributed version control and source code management (SCM) functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

# Features of GitHub

* Collaboration
* Integrated issue and bug tracking
* Graphical representation of branches
* Git repositories hosting
* Project management
* Team management
* Code hosting
* Track and assign tasks
* Conversations
* Wikisc

## Benefits of GitHub

* + GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

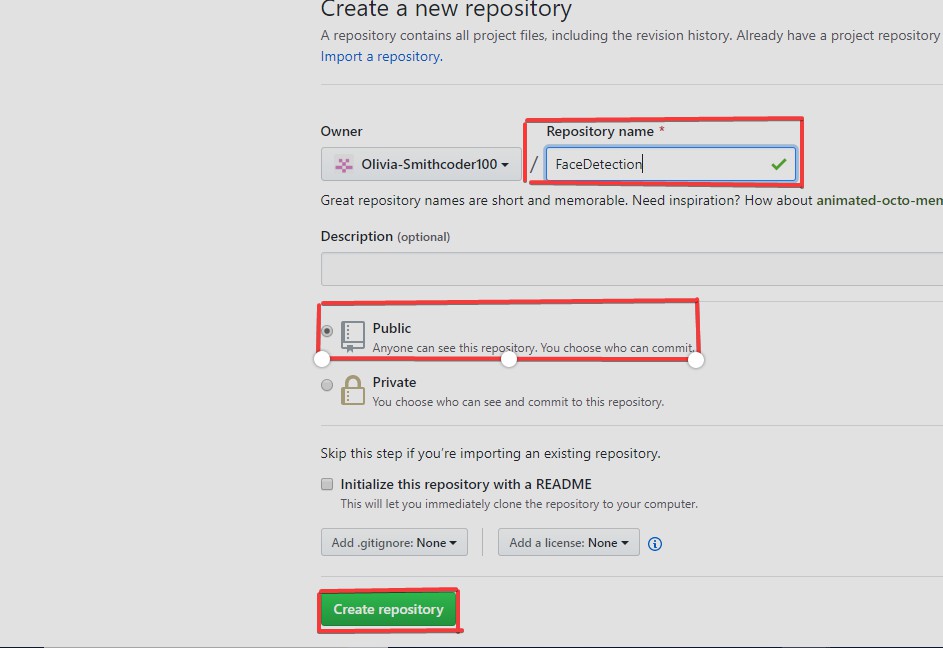
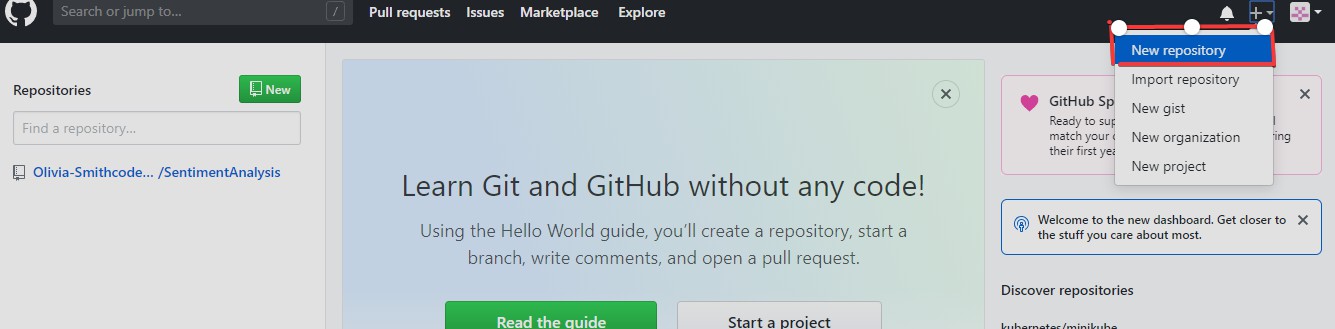
The key benefits of GitHub are as follows.

* + It is easy to contribute to open source projects via GitHub.
  + It helps to create an excellent document.
  + You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
  + It allows your work to get out there in front of the public.
  + You can track changes in your code across versions.

Steps:

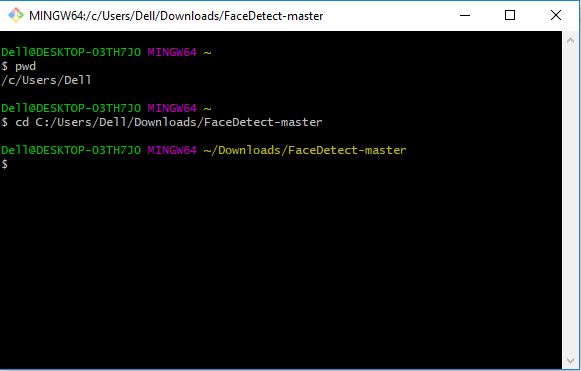
## Using Command line to PUSH to GitHub

1. **Creating a new repository**
   * You need to create a new repository and click on the plus sign.
   * Fill up all the required details, i.e., repository name, description and also make the repository public this time as it is free.



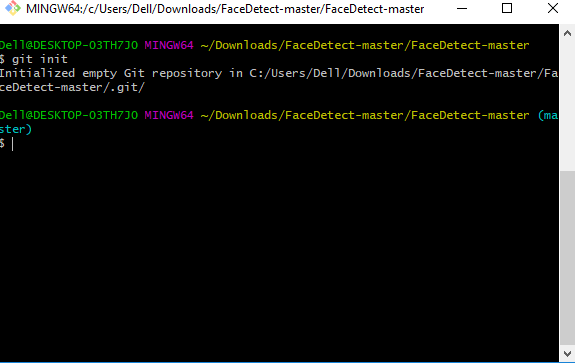
## Open your Git Bash

1. **Create your local project in your desktop directed towards a current working directory**
   * pwd stands for 'print working directory', which is used to print the current directory.
   * Move to the specific path in your local computer by cd 'path\_name'. The cd commands stand for 'change directory' and it is used to change to the working directory in your operating system, and to locate your file, 'path\_name', i.e., C:/Users/Dell/Downloads/FaceDetect- master needs to be given. This command can identify the required file that you are looking to work with.



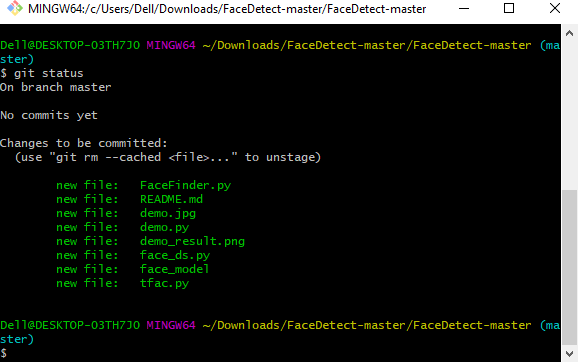
## Initialize the git repository

* + Use git init to initialize the repository. It is used to create a new empty repository or directory consisting of files' with the hidden directory. '.git' is created at the top level of your project, which places all of the revision information in one place.



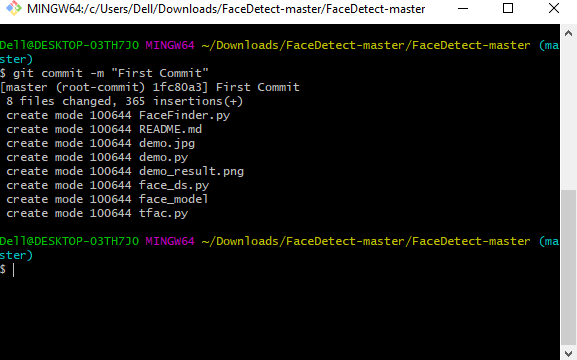
## Add the file to the new local repository

* + Use git add . in your bash to add all the files to the given folder.
  + Use git status in your bash to view all the files which are going to be staged to the first commit.



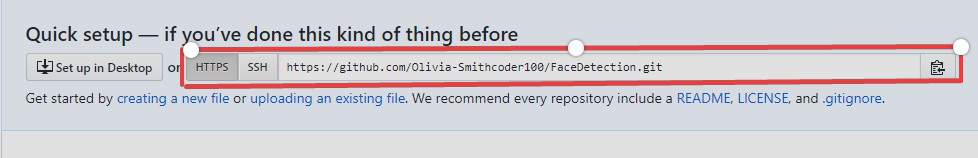
## Commit the files staged in your local repository by writing a commit message

* You can create a commit message by git commit -m 'your message', which adds the change to the local repository.
* git commit uses '-m' as a flag for a message to set the commits with the content where the full description is included, and a message is written in an imperative sentence up to 50 characters long and defining "what was changed", and "why was the change made".



## Copy your remote repository's URL from GitHub

* + The HTTPS or URL is copied from the given GitHub account, which is the place of the remote repository.



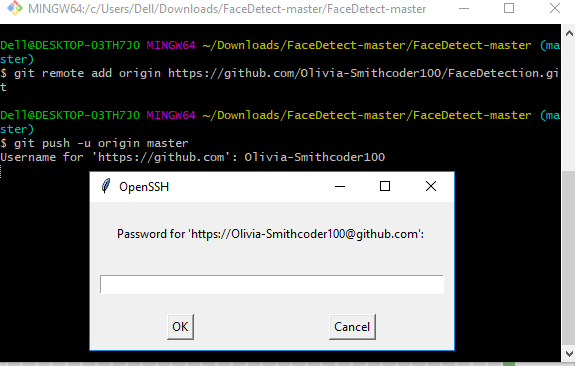
## Add the URL copied, which is your remote repository to where your local content from your repository is pushed

git remote add origin 'your\_url\_name'

* + In the above code, The 'origin' is the remote name, and the remote URL is "[**https://github.com/Olivia-Smithcoder100/FaceDetection.git**](https://github.com/Olivia-Smithcoder100/FaceDetection.git)". You can see the remote as GitHub in this case, and GitHub provides the URL for adding to the remote repository.

## Push the code in your local repository to GitHub

* + git push -u origin master is used for pushing local content to GitHub.
  + In the code, the origin is your default remote repository name and '-u' flag is upstream, which is equivalent to '-set-upstream.' and the master is the branch, name.upstream is the repository that we have cloned the project.
  + Fill in your GitHub username and password.



## View your files in your repository hosted on GitHub

* + You can finally see the file hosted on GitHub.

