

# VERSION CONTROL



git

By; Ahmed EL-Mahdy

LNKD: [bitlylink.com/ooLxn](https://bitlylink.com/ooLxn)

FB: [bit.ly/2Utbjkz](https://bit.ly/2Utbjkz)

# LECTURE 1

---

- What is Git?
- Why is Git used?
- Git Lifecycle
- Git Installation
- Environment setup
- Common Git Commands-Local and Remote

# WHAT IS GIT

---

**Git** helps to keep track of the history of codes files by storing them in versions on its own server repository

EXAMPLE: GitHub.

**Git** has:

- Functionality
- Performance
- security
- flexibility

# WHY GIT

---

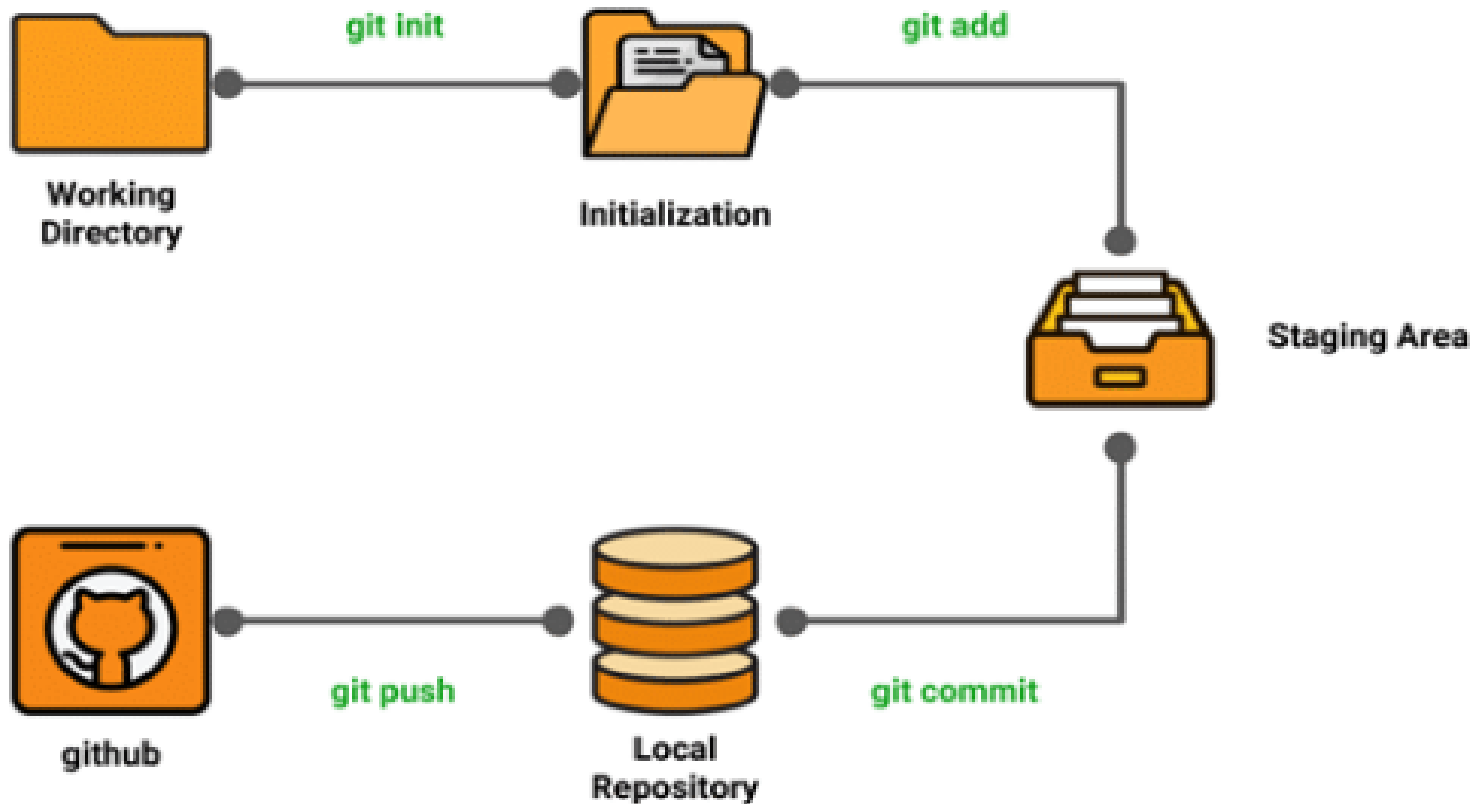
**Git** is so popular for those reasons:

- Work offline
- Undo your mistakes
- Restore deleted commits
- Secure
- Performance
- Flexibility

# GIT LIFE CYCLE

---

- Local working directory



# GIT LIFE CICLE

---

- Git lifecycle
  - Initialization
  - Staging area
  - Commit

# INSTALL GIT

---

## Linux :

- Debain and Ubuntu  
`sudo apt-get install git`
- CentOS  
`sudo yum install git`
- Fedora  
`sudo yum install git-core`
- Arch Linux  
`sudo pacman -Sy git`
- Gentoo  
`sudo emerge --ask --verbose dev-vcs/git`

## Windows:

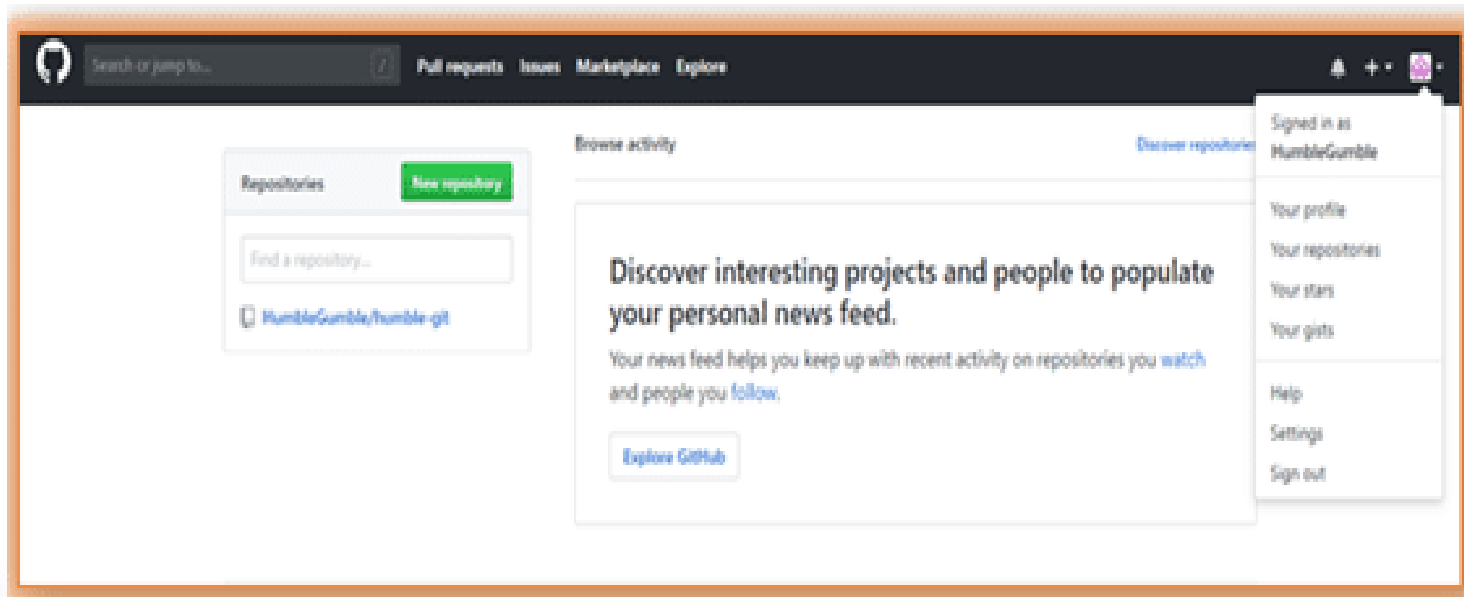
<https://bit.ly/2OSPzzb>

# ENVIRONMENT SETUP

---

## Create a GitHub account

- Go to <https://Github.com>
- Create one GitHub account
- Login





# ENVIRONMENT SETUP

---

## Configuring Git

To tell Git who you are, run the following two commands

```
INTELLIPAAT@DESKTOP-186LH03 MINGW64 /c/git/ProjectGit (master)
$ git config --global user.email "debashis.intellipaata@gmail.com"

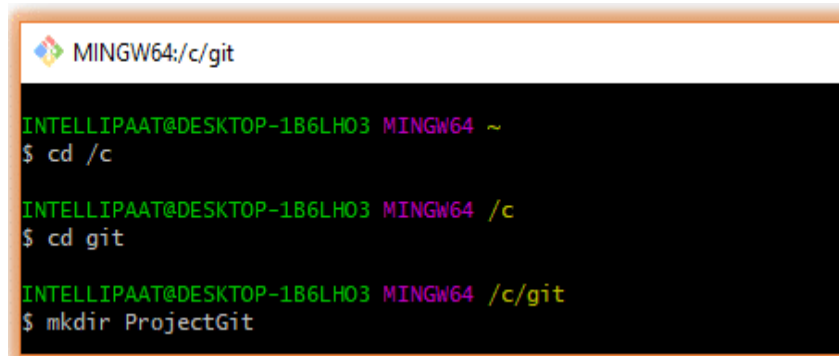
INTELLIPAAT@DESKTOP-186LH03 MINGW64 /c/git/ProjectGit (master)
$ git config --global user.name "HumbleGumble"
```

# ENVIRONMENT SETUP

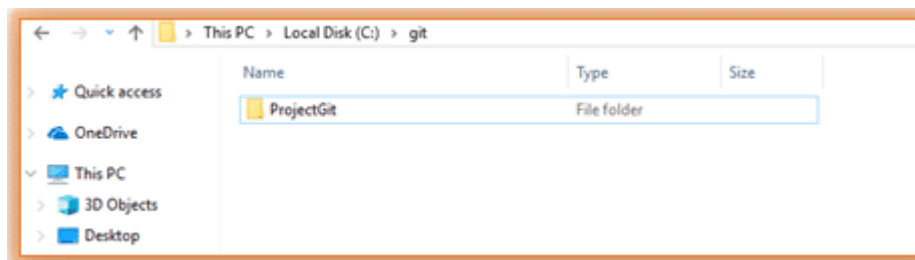
---

## Creating a Local Repository

- Open a terminal and move to wanted place for the project on the local machine using `cd` (change directory)
- Example



```
MINGW64:/c/git  
  
INTELLIPAAT@DESKTOP-1B6LH03 MINGW64 ~  
$ cd /c  
  
INTELLIPAAT@DESKTOP-1B6LH03 MINGW64 /c  
$ cd git  
  
INTELLIPAAT@DESKTOP-1B6LH03 MINGW64 /c/git  
$ mkdir ProjectGit
```



# GIT COMMANDS

## LOCAL COMMENDS

---

### Git init [repository name]

- Navigate to project directory
- type the command **git init** to initialize a Git repository for local project folder.
- Git will create a hidden **.git** directory and use it for keeping its files organized in other subdirectories.

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ cd

intellipaat@DESKTOP-SPC6JQB MINGW64 ~
$ cd ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ git init
Initialized empty Git repository in C:/Users/intellipaat/ProjectGit1/.git/

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ |
```

# GIT COMMANDS

## LOCAL COMMENDS

---

### Git touch

To add files to a Project

- Create a new file with command touch
- Will see the files present in the master branch

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ touch humble.txt

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ ls
humble.txt
```

# GIT COMMANDS

## LOCAL COMMENDS

---

### Git status

After creating the new file, you can use the Git status command and see the status of the files in the master branch.

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        humble.txt

nothing added to commit but untracked files present (use "git add" to track)
```

# GIT COMMANDS

## LOCAL COMMENDS

---

### **Git add [file(s) name]**

This will add the specified file(s) into the Git repository

*i.e.*, into the staging area where they are already being tracked by Git and now ready to be committed.

```
intellipaate@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git add 234master.txt
warning: LF will be replaced by CRLF in 234master.txt.
The file will have its original line endings in your working directory.
```

### **git add . , git add \* or git add -A**

This will take all our files into the Git repository

*i.e.*, into the staging area.

```
intellipaate@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git add .
warning: LF will be replaced by CRLF in 234.txt.
The file will have its original line endings in your working directory.
```

# GIT COMMANDS

## LOCAL COMMENDS

---

### Git status

This command will show the modified status of an existing file and the file addition status of a new file, if any, that have to be committed.

```
intellipaate@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   234.txt
        new file:   ls
```

# GIT COMMANDS

## LOCAL COMMENDS

---

### **Git commit-m "message"**

This command records or snapshots the file permanently in the version history.

All the files which are there in the directory right now are being saved in the git file system.

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git commit -m "Committing 234master.txt in master"
[master (root-commit) be73fc3] Committing 234master.txt in master
2 files changed, 2 insertions(+)
create mode 100644 123master.txt
create mode 100644 234master.txt
```



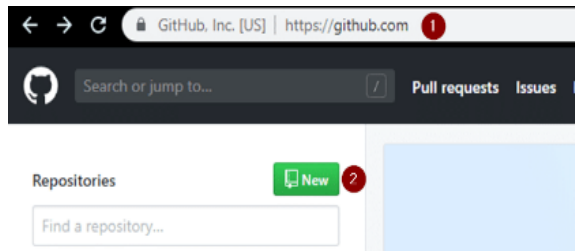
# GIT COMMANDS

## REMOTE COMMENDS

Once everything is ready on our local system, we can start pushing the code to the remote (central) repository of the project.

### Create new Repo

1. Go to GitHub account
2. New
3. Create a new repository



### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

HumbleGumble ▾

Repository name

humbleRepo ✓

Great repository names are short and memorable. Need inspiration? How about curly-fiesta.

Description (optional)

My new repository



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

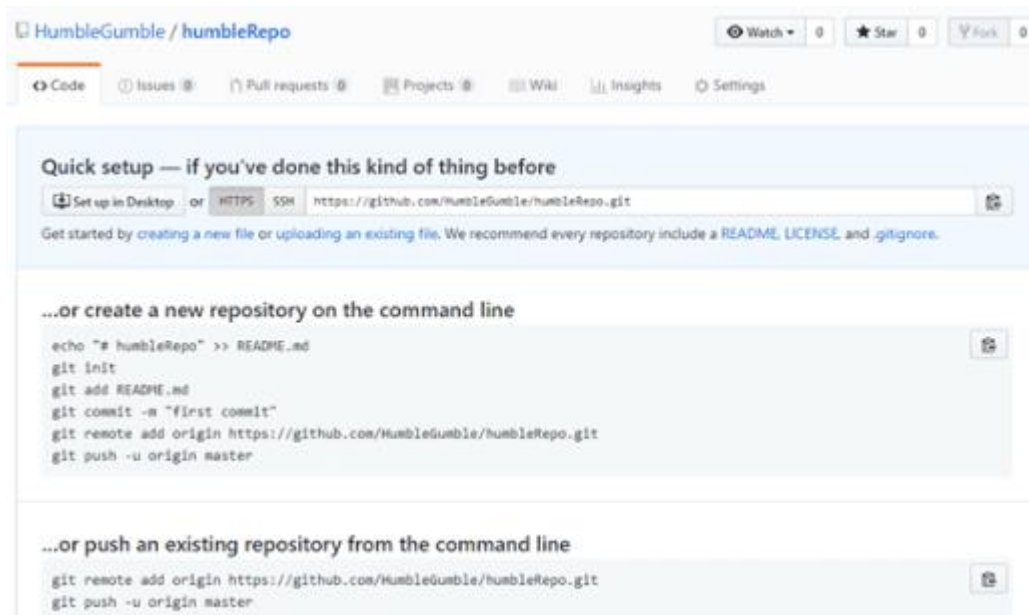
# GIT COMMANDS

## REMOTE COMMENDS

---

### Create new Repo

land on



# GIT COMMANDS

## REMOTE COMMENDS

---

### **Git remote add origin “[URL]”**

Operate remote commands in the repository

(copy repo url and paste as below)

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git remote add origin https://github.com/prabhpreetk/humbleRepo1.git
```

# GIT COMMANDS

## REMOTE COMMENDS

---

### **git push origin [branch name]**

By using the command 'git push', the local repository's files will be synced with the remote repository on Github.

(in case changes in the files need to push to remote repo)

```
intellipa@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git push origin branch1
fatal: HttpRequestException encountered.
An error occurred while sending the request.
Username for 'https://github.com': prabhpreetk
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 729 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:   https://github.com/prabhpreetk/humbleRepo1/pull/new/branch1
remote:
To https://github.com/prabhpreetk/humbleRepo1.git
 * [new branch]      branch1 -> branch1
```

# GIT COMMANDS

## REMOTE COMMENDS

---

### **git pull**

When it comes to syncing remote repository pull command comes in handy.

Let us take a look at the steps that leads to pulling operation in Git.

- remote origin
- pull master

```
INTELLIPAAT@DESKTOP-1B6LHD3 MINGW64 /c/git/ProjectGit (master)
$ git remote set-url origin "https://github.com/HumbleGumble/humbleRepo.git"

INTELLIPAAT@DESKTOP-1B6LHD3 MINGW64 /c/git/ProjectGit (master)
$ git pull origin master
From https://github.com/HumbleGumble/humbleRepo
 * branch                master      -> FETCH_HEAD
+ 5ad410f...169cfba master      -> origin/master (forced update)
Already up to date.
```

# GIT COMMANDS

## REMOTE COMMENDS

---

**How to work on exist project !**

# GIT COMMANDS

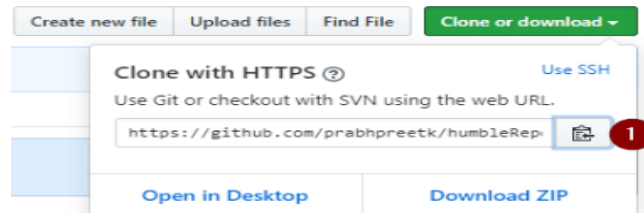
## REMOTE COMMENDS

---

### **git clone [url]**

It imports the files of project from the remote repository to the local system.

- Click Clone or Download.
- Copy the link and paste it on the terminal with git clone command.



- Create a local folder  
`mkdir [directory- name]`  
`cd [directory- name]`  
`git clone [URL]`

# GIT COMMANDS

## REMOTE COMMENDS

---

`git clone [url]`

```
intellipaate@DESKTOP-SPC6JQB MINGW64 ~/test
$ git clone https://github.com/prabhpreetk/humbleRepo1.git
Cloning into 'humbleRepo1'...
```

**Note:** Don't use the git remote add origin command because now if you push any new file , it knows where it has to go .



# GIT COMMEND

---

**WRAPUP!**

# WRAPUP GIT COMMEND

---

## **Get Documentation For a Command**

`$ man git-log`

OR

`$ git help log`

## **introduce yourself to Git with your name and public email address**

`$ git config --global user.name "Your Name Comes Here"`

`$ git config --global user.email you@yourdomain.example.com`

# WRAPUP GIT COMMEND

---

## Importing a new project

Assume you have a tarball `project.tar.gz` with your initial work. You can place it under Git revision control as follows.

```
$ tar xzf project.tar.gz
```

```
$ cd project
```

```
$ git init
```

Git will reply

Initialized empty Git repository in `.git/`

You've now **initialized** the working directory—you may notice a new directory created, named `".git"`.

Next, tell Git to take a snapshot of the contents of all files under the current directory (note the `.`), with **git add** :

```
$ git add .
```

This snapshot is now stored in a temporary staging area which Git calls the `"index"`. You can permanently store the contents of the index in the repository with **git commit**:

```
$ git commit
```

This will prompt you for a commit message. You've now stored the first version of your project in Git.

# WRAPUP GIT COMMEND

---

## Making changes

Modify some files, then add their updated contents to the index:

```
$ git add file1 file2 file3
```

You are now ready to commit. You can see what is about to be committed using git diff with the --cached option:

```
$ git diff --cached
```

(Without --cached, git diff will show you any changes that you've made but not yet added to the index.) You can also get a brief summary of the situation with git status:

```
$ git status
```

On branch master

Changes to be committed:

Your branch is up to date with 'origin/master'.

(use "git restore --staged <file>..." to unstage)

```
modified: file1
```

```
modified: file2
```

```
modified: file3
```

# WRAPUP GIT COMMEND

---

## Making changes

If you need to make any further adjustments, do so now, and then add any newly modified content to the index. Finally, commit your changes with:

```
$ git commit
```

This will again prompt you for a message describing the change, and then record a new version of the project.

instead of running git add beforehand, you can use

```
$ git commit -a
```

which will automatically notice any modified (but not new) files, add them to the index, and commit, all in one step.

## Git tracks content not files

Many revision control systems provide an **add** command that tells the system to start tracking changes to a new file. Git's **add** command does something simpler and more powerful: git add is used both for new and newly modified files, and in both cases it takes a snapshot of the given files and stages that content in the index, ready for inclusion in the next commit.

# WRAPUP GIT COMMEND

---

## Viewing project history

At any point you can view the history of your changes using

```
$ git log
```

If you also want to see complete diffs at each step, use

```
$ git log -p
```

Often the overview of the change is useful to get a feel of each step

```
$ git log --stat --summary
```

## Using Git for collaboration

Suppose that Alice has started a new project with a Git repository in `/home/alice/project`, and that Bob, who has a home directory on the same machine, wants to contribute.

Bob begins with:

```
bob$ git clone /home/alice/project myrepo
```

This creates a new directory "myrepo" containing a clone of Alice's repository. The clone is on an equal footing with the original project, possessing its own copy of the original project's history.