What is Git and why should I use it?

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.

There are many ways it can be set up and configured, but at my job, here's how we use it: when a new employee starts, he downloads all the files from Github, which is an online server we're all connected to.

So he has *his* local version of the files, I have *my* local version, our boss has *his* local version, etc.

When I make a change to some files, I go through the following process in the Terminal. (There are GUI clients for Git, but I prefer working on the command line.)

```
> git pull
```

That pulls the latest changes down from github. If there are conflicts between those and my local ones, it tells me what they are, file-by-file, line-by-line, and I now have a chance to reconcile those differences.

After editing the files or creating new ones, I run this command:

```
> git add .
```

Which adds all of my local changes to git, so that git knows about them. The dot after add specifically means to add *all* the changes I've made, e.g. new files I've added to my local folder or changes I've made to existing files. If I want, I can add only specific files, e.g.

```
> git add myNewFile.js
```

I now write a comment about the adds I just made.

```
> git commit -m "Fixed a major bug which stopped reports from printing."
```

Finally, I upload my changes to the server.

```
> git push
```

Now, when my colleagues do a ...

```
> git pull
```

... they will get my changes, and they will be notified if any of them conflict with their local versions.

There are all kinds of cool, useful commands for rolling back changes to a particular time or state. But probably the most useful thing about Git is branching. Let's say my team is working on code for an Asteroids game, and I get the idea for making spinning asteroids. This will involve making some major changes to the existing asteroids code, and I'm a little scared to do that. No worries, I can just make a branch.

First of all, I'll check which branches exist:

```
 > git branchmaster*
```

So there's currently only one branch on my local machine, called master. The star by it means that's the branch I'm currently working in. I'll go ahead and create a new one:

```
> git branch spinningAsteroids
```

That creates a copy of all the files in master. I'll now move into that branch.

```
 > git checkout spinningAsteroids> git branchmasterspinningAsteroids*
```

I now spend a couple of hours in spinningAsteroids, doing whatever coding I need to do, not worrying about messing things up, because I'm in a branch. Meanwhile, I get a tech support call. They've found a critical bug and I need to fix it asap. No worries...

```
> git checkout master
```

... fix bug ...

```
 > git pull> git add .> git commit -m "Fixed critical bug with high scores."> git push
```

Now I can resume my work with spinningAsteroids.

```
 > git checkout spinningAsteroids> git branchmasterspinningAsteroids*

... work, work, work ...

Okay, I'm now happy with my spinning asteroids, and I want to merge that new code into the main code base, so...

 > git checkout master> git branchmaster*spinningAsteroids

> git merge spinningAsteroids

Now the code from my branch is merged into the main code-base. I can now upload it.

 > git pull> git add .> git commit -m "added new cool feature! Spinning asteroids!!!"> git push
```

There are many ways to use Git. This is just one of them.

Git is a version control system. For example, if you have a file on which you've been working on and reworking for a long time, all the versions of it are saved in Git, and you can easily get back to every version.

What is it for?

The version control system has the following benefits:

• You have access to all versions of all files in Git repository at any time, it's almost impossible to lose any part of a code.

• Multiple developers can work on one project at the same time without interfering with each other, and without fear of losing any changes made by a colleague. In Git, the possibilities of collaborative work are unlimited.

You will have to use Git every day, and this is a tool you should have a perfect command of.

Be sure to sign up on GitHub or Bitbucket. These services offer great manuals on how to start working with remote repositories. I like bitbucket for the possibility of creating unlimited number of private repositories. However, in terms of all other characteristics, GitHub is way ahead of bitbucket, and has turned into the standard and Mecca of the whole open source community.

Resources:

Try Git — an interactive introduction course on Git. If you study it closely and thoroughly you'll know enough to perform usual tasks;

Version control: best practices — an article on the best ways of using Git and Github;

A Git Workflow for Agile Teams — provides insight into the organization of work with Git repository for a team of three or more developers.

Jane is at city X and her mom is at city Y.

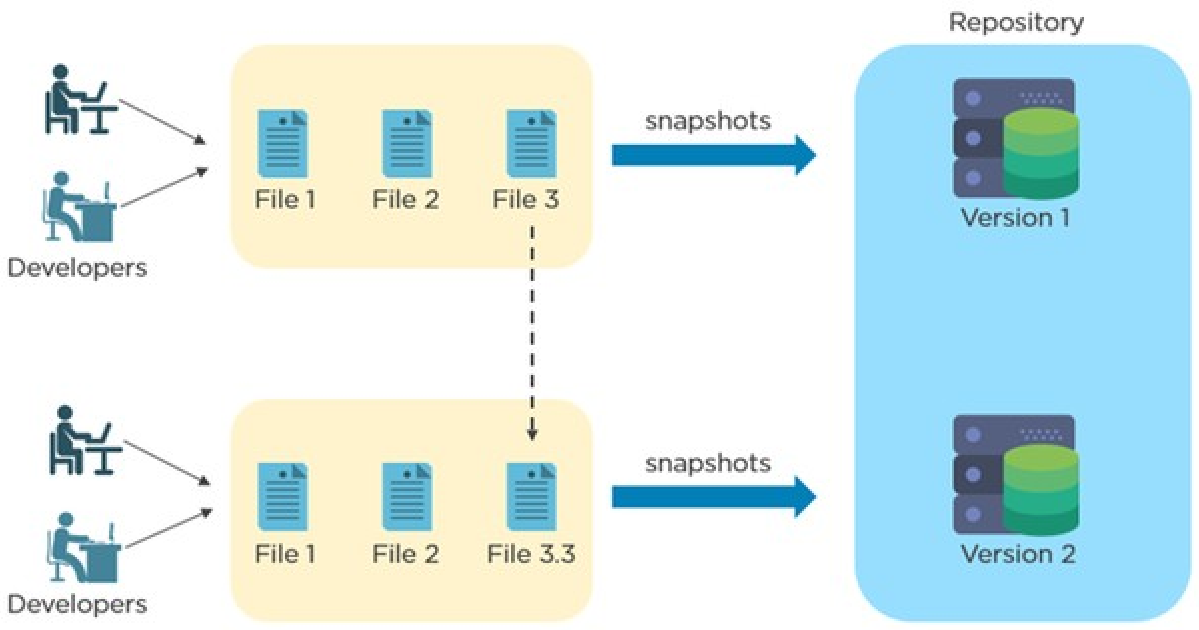
Jane's mom wants to cook a dish and wants Jane's help in it. They want to collaborate with each other in cooking this dish (building a software).

There are three utensils in this story -- one master pan, where the actual food is being prepared, two 'dev' pans; one for Jane and another for her mom, for them to experiment. They add stuff, balance ingredients, etc. in the 'dev' pan. In git terms, these are **branches** within a repository.

Once they find a mix that tastes good in the 'dev' pan, they **merge** it with the 'master' pan. And that's how they **commit** each of their contributions towards the dish.

Git is a **Version Control System** used in **DevOps lifecycle** for **Planning** and **Coding** phase. First let's understand what is a Version Control System (**VCS**).

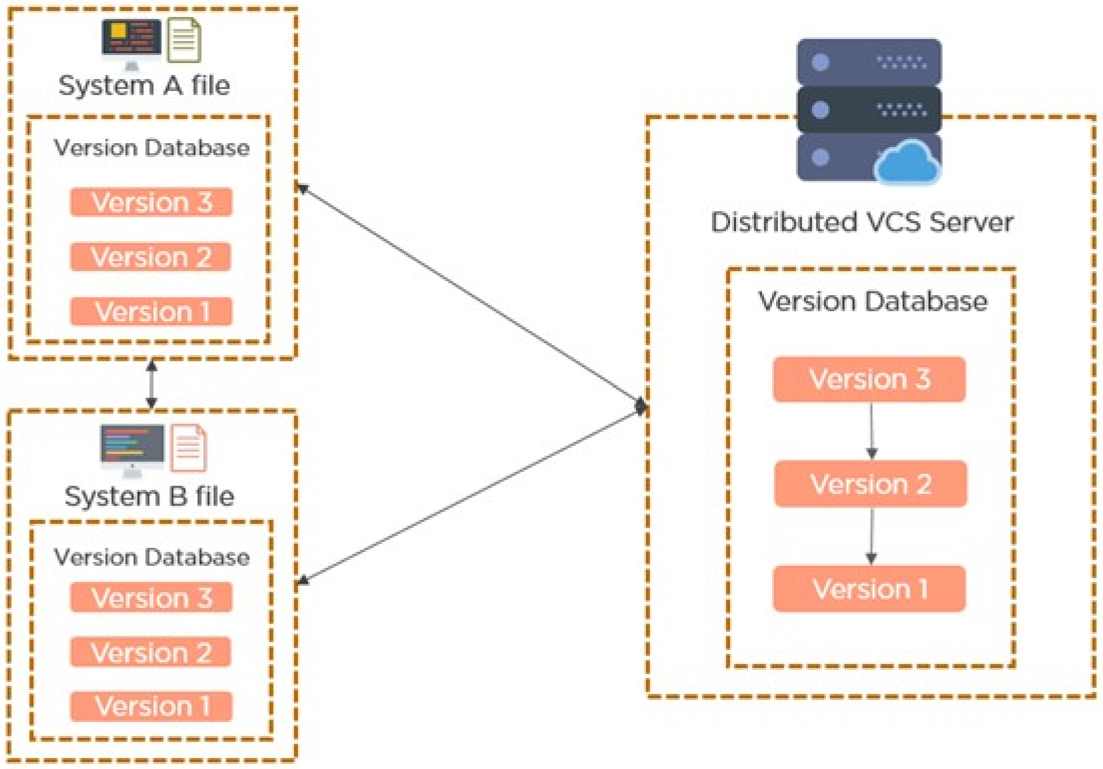
A VCS allows you to keep every change you make in the code repository. VCS stores changes as **Versions** by taking a **snapshot** of every change. Here a how the diagram of a VCS looks like:



As you can see, Developers are making changes to the file and each version of the file is being stored in a repository. This repository is called **GitHub** repository.

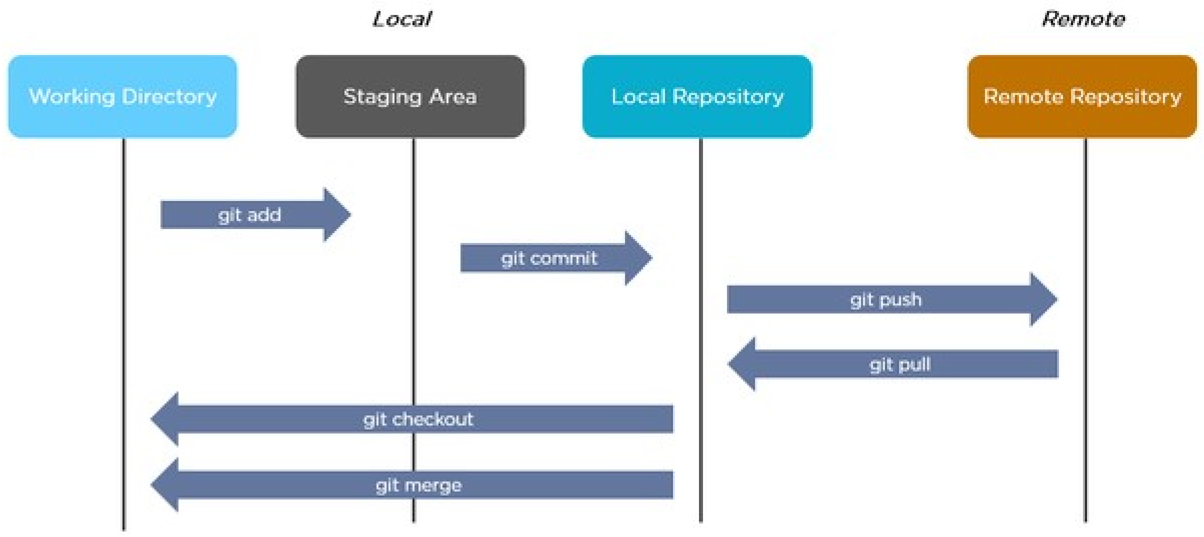
Git is a **Distributed Version Control** tool that allows multiple users/developers to write codes, edit and make commits to the server repository (GitHub).

Here is a diagram that shows how a distributed VCS architecture looks like:



As you can see, every developer has a copy of the entire code along with its history. This allows all the developers to update their local repositories with new data from the server (GitHub).

The basic architecture of Git can be shown as:

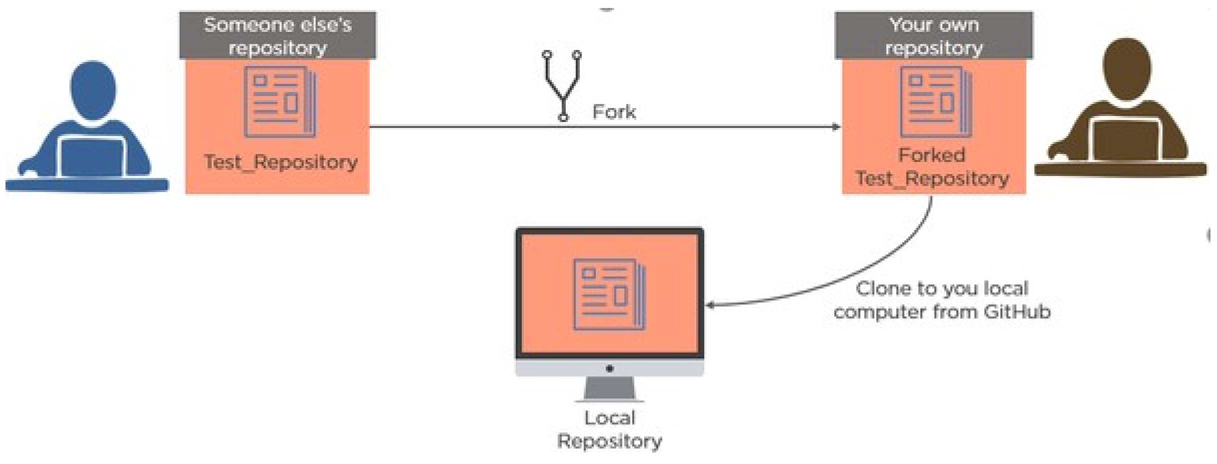


Let me explain about the architecture:

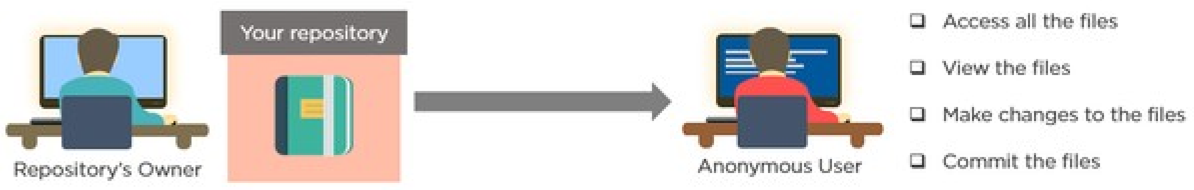
- Working directory is your current location of the folder you are working with.
- Then you add your files to the staging area before saving the changes (commit)
- After all the changes are made, you commit the files to the local repository
- Then you push the committed files to the remote repository
- If needed, you can the push the changes from the remote repository to the local repository
- Git allows you to create branches and switch to them when required
- After the changes are complete, you can merge the newly created branches to the master branch

Git provides a lot of features that are really helpful while managing a large scale project. Some of them are discussed below:

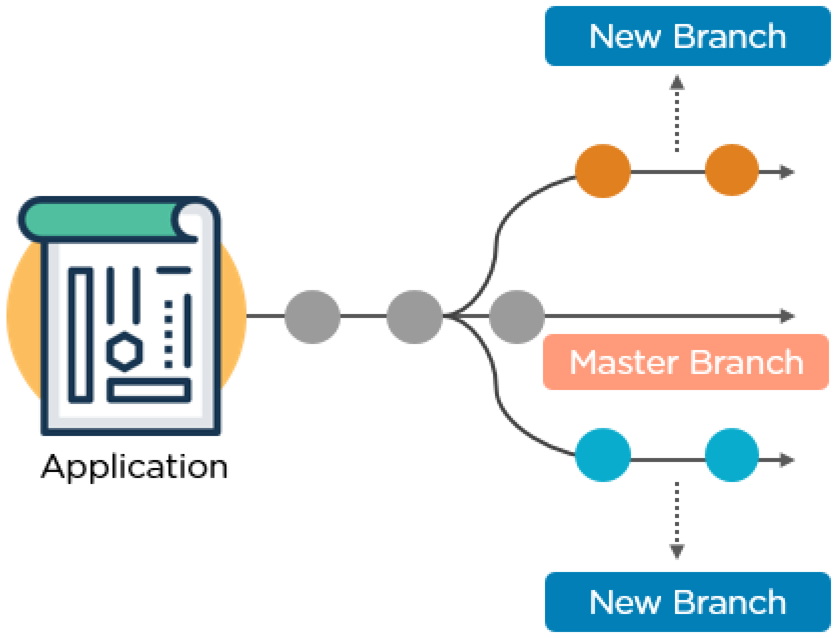
- **Fork and Clone**: Using Git, you can fork a remote repository and clone it to the local system.



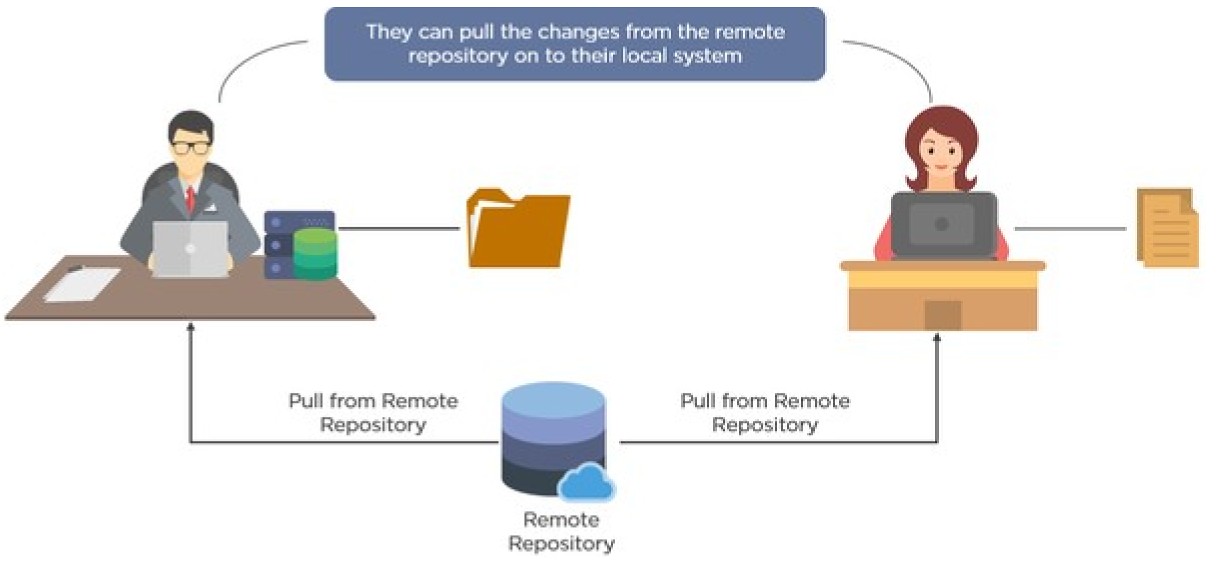
- **Collaborators**: Git allows you to give permission to edit a repository owned by someone else by them as collaborators.



- **Branch**: You can create a new branch and build a feature of an application and merge it back once the feature of the app is complete



- **Pull from a remote**: Using Git, you can pull any file that you want or the changes made by a collaborator to the local system.


They can pull the changes from the remote repository on to their local system
Pull from Remote Repository
Pull from Remote Repository
Remote Repository

Git is a software that is used for Version Control. It is free and open source.

Now, let’s understand what is Version Control.

Version Control is the management of changes to documents, computer programs, large websites and other collection of information.

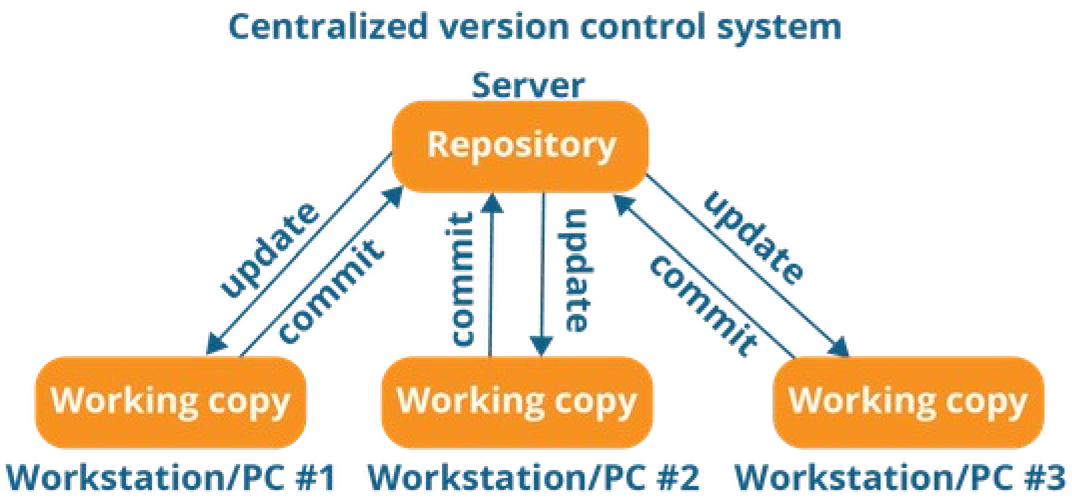
There are two types of VCS:

- Centralized Version Control System (CVCS)
- Distributed Version Control System (DVCS)

Centralized VCS

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

Please refer to the diagram below to get a better idea of CVCS:



The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.

Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.

Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

- It is not locally available; meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

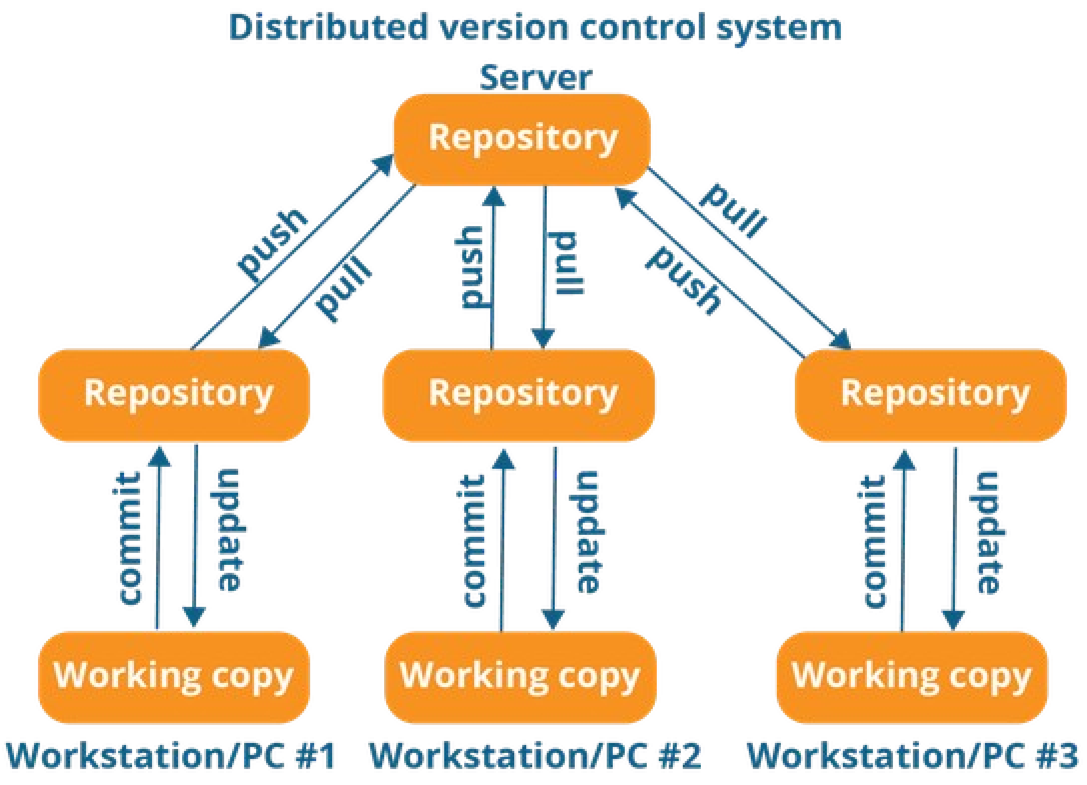
This is when Distributed VCS comes to the rescue.

Distributed VCS

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

You will understand it better by referring to the diagram below:



As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called "**pull**" and affect changes to the main repository by an operation called "**push**" from their local repository.

The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.
- Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.
- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
- If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.

Git is a DVCS.

This mostly covers up both of your questions pretty much. Now, let me tell you how easy it is to use Git.

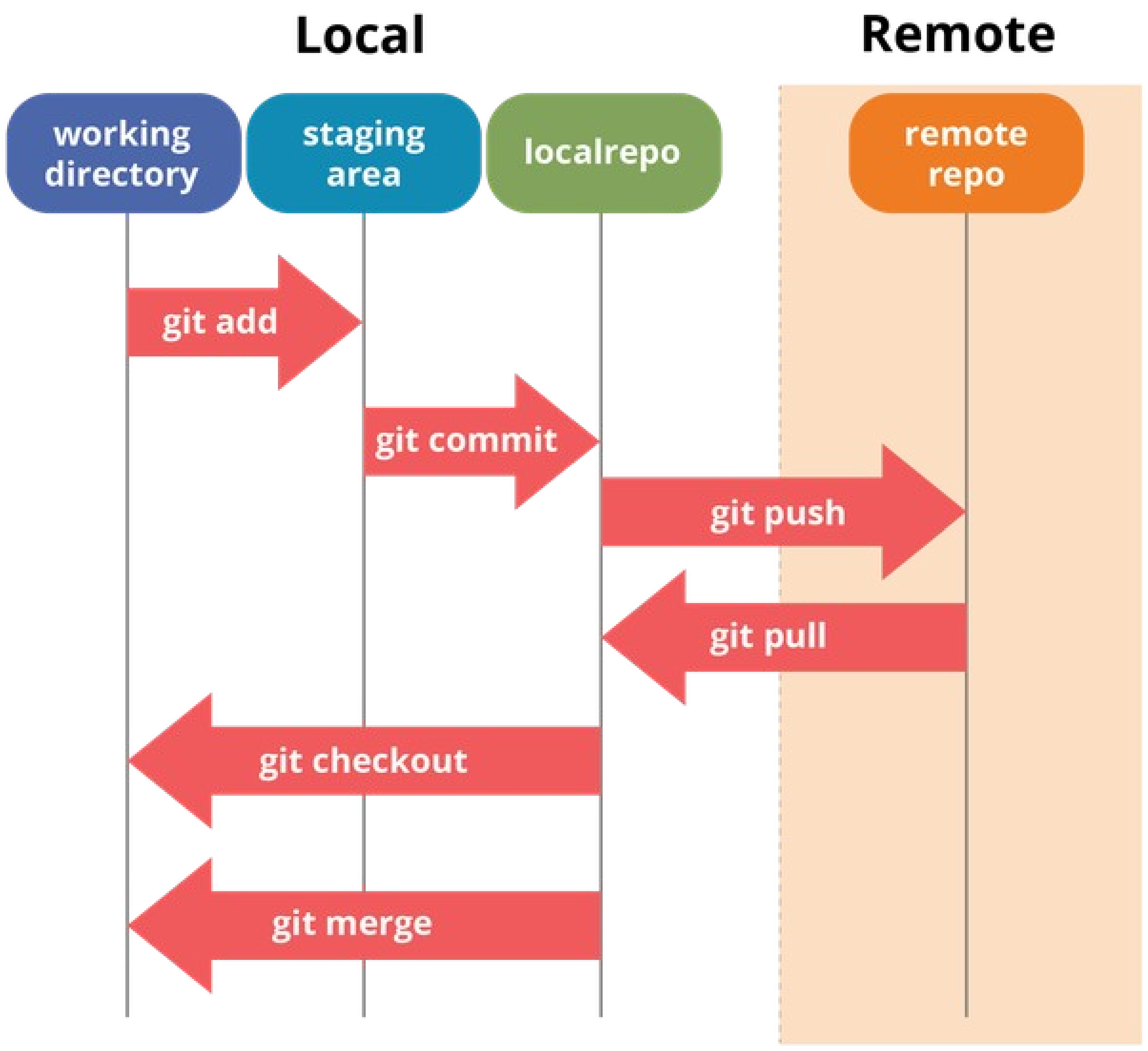
Some of the basic operations in Git are:

1. Initialize
2. Add
3. Commit
4. Pull
5. Push

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing

Let me first give you a brief idea about how these operations work with the Git repositories. Take a look at the architecture of Git below:

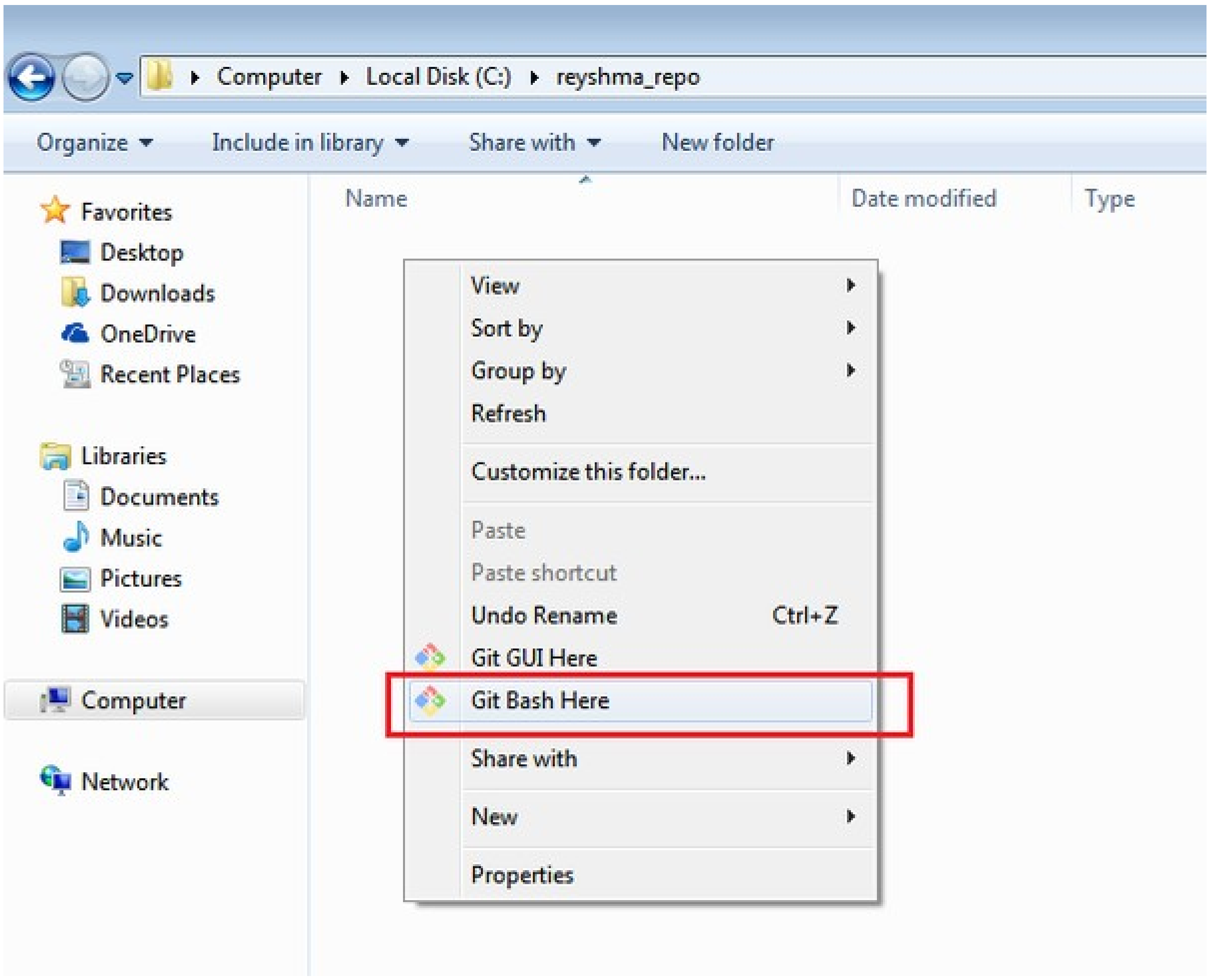


If you understand the above diagram well and good, but if you don't, you need not worry, I will be explaining these operations in this Git Tutorial one by one. Let us begin with the basic operations.

You need to install Git on your system first. If you need help with the installation, ***click here***.

In this Git Tutorial, I will show you the commands and the operations using Git Bash. Git Bash is a text-only command line interface for using Git on Windows which provides features to run automated scripts.

After installing Git in your Windows system, just open your folder/directory where you want to store all your project files; right click and select '***Git Bash here***'.

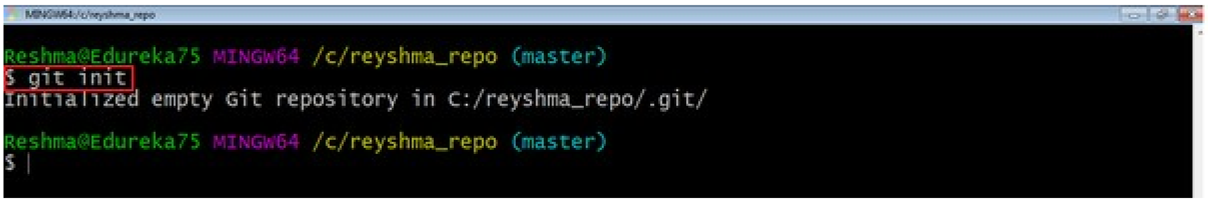


This will open up Git Bash terminal where you can enter commands to perform various Git operations.

Now, the next task is to initialize your repository.

Initialize

In order to do that, we use the command **git init.** Please refer to the below screenshot.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git init
Initialized empty Git repository in C:/reyshma_repo/.git/

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ |
```

git init creates an empty Git repository or re-initializes an existing one. It basically creates a **.git** directory with sub directories and template files. Running a **git init** in an existing repository will not overwrite things that are already there. It rather picks up the newly added templates.

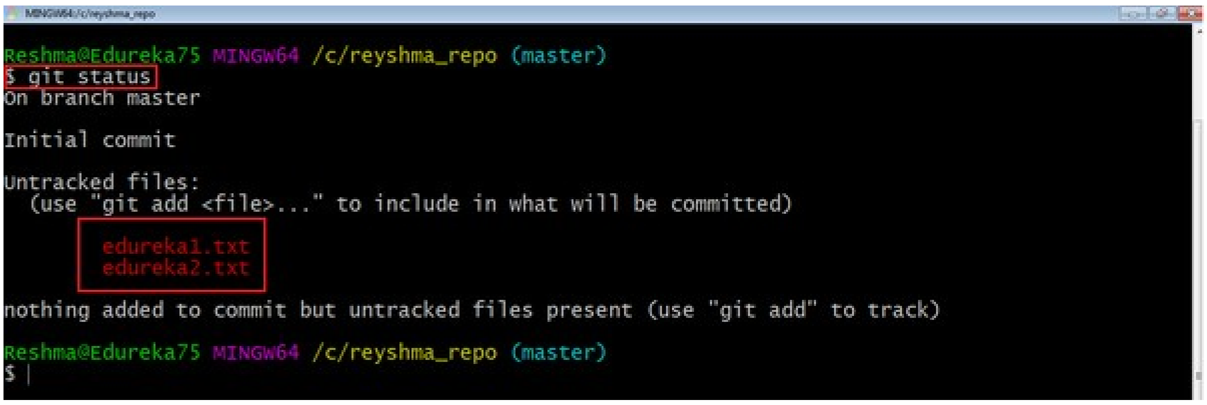
Now that my repository is initialized, let me create some files in the directory/repository. For e.g. I have created two text files namely *edureka1.txt* and *edureka2.txt*.

Let's see if these files are in my index or not using the command **git status**. The index holds a snapshot of the content of the working tree/directory, and this snapshot is taken as the contents for the next change to be made in the local repository.

Git status

The **git status** command lists all the modified files which are ready to be added to the local repository.

Let us type in the command to see what happens:



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git status
On branch master

Initial commit

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

        edureka1.txt
        edureka2.txt

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

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ |
```

This shows that I have two files which are not added to the index yet. This means I cannot commit changes with these files unless I have added them explicitly in the index.

Add

This command updates the index using the current content found in the working tree and then prepares the content in the staging area for the next commit.

Thus, after making changes to the working tree, and before running the **commit** command, you must use the**add** command to add any new or modified files to the index. For that, use the commands below:

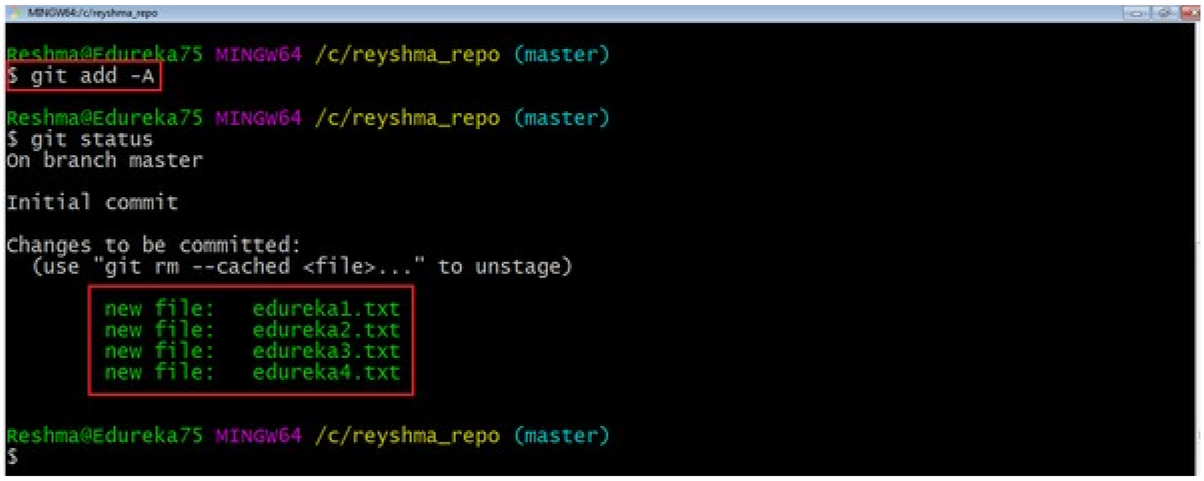
git add <directory>

or

git add <file>

Let me demonstrate the **git add** for you so that you can understand it better.

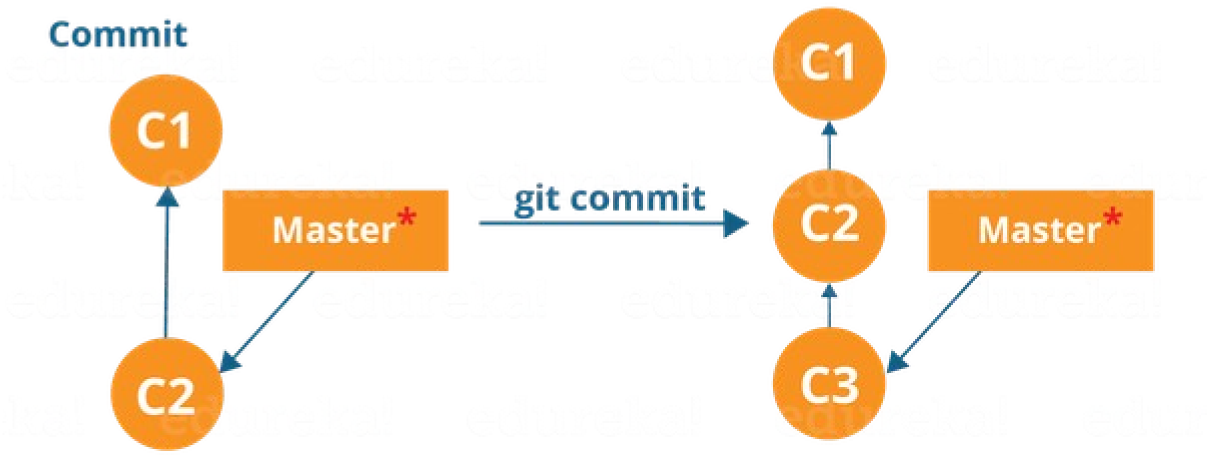
I have created two more files *edureka3.txt* and *edureka4.txt*. Let us add the files using the command **git add -A**. This command will add all the files to the index which are in the directory but not updated in the index yet.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git add -A

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   edureka1.txt
        new file:   edureka2.txt
        new file:   edureka3.txt
        new file:   edureka4.txt

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$
```

Commit

It refers to recording snapshots of the repository at a given time. Committed snapshots will never change unless done explicitly. Let me explain how commit works with the diagram below:



Here, C1 is the initial commit, i.e. the snapshot of the first change from which another snapshot is created with changes named C2. Note that the master points to the latest commit.

Now, when I commit again, another snapshot C3 is created and now the master points to C3 instead of C2.

Git aims to keep commits as lightweight as possible. So, it doesn't blindly copy the entire directory every time you commit; it includes commit as a set of changes, or "delta" from one version of the repository to the other. In easy words, it only copies the changes made in the repository.

You can commit by using the command below:

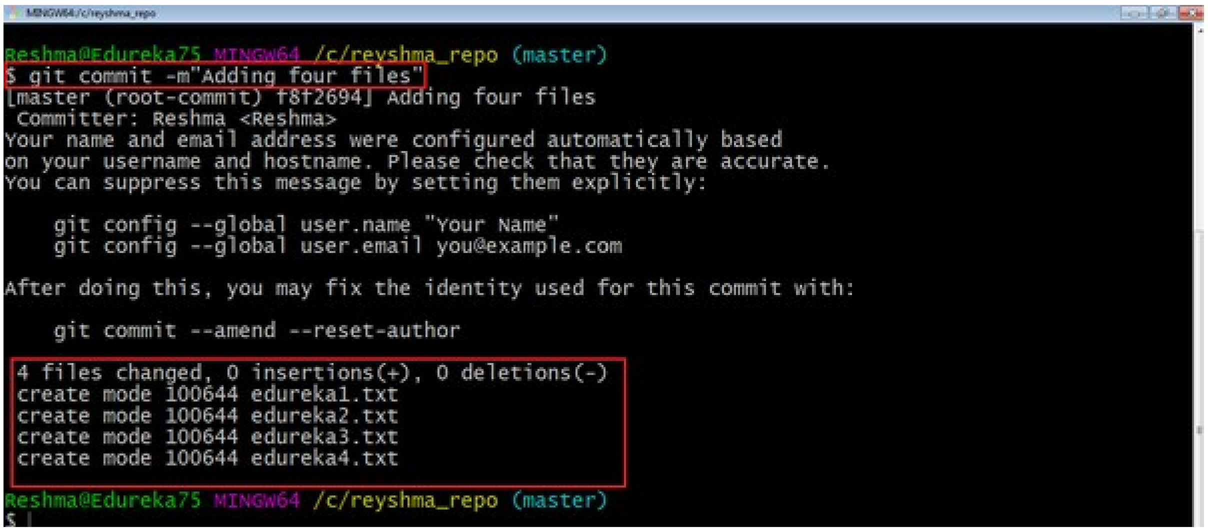
git commit

This will commit the staged snapshot and will launch a text editor prompting you for a commit message.

Or you can use:

git commit -m "<message>"

Let's try it out.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git commit -m"Adding four files"
[master (root-commit) f8f2694] Adding four files
 Committer: Reshma <Reshma>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 edureka1.txt
 create mode 100644 edureka2.txt
 create mode 100644 edureka3.txt
 create mode 100644 edureka4.txt

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$
```

As you can see above, the **git commit** command has committed the changes in the four files in the local repository.

Now, if you want to commit a snapshot of all the changes in the working directory at once, you can use the command below:

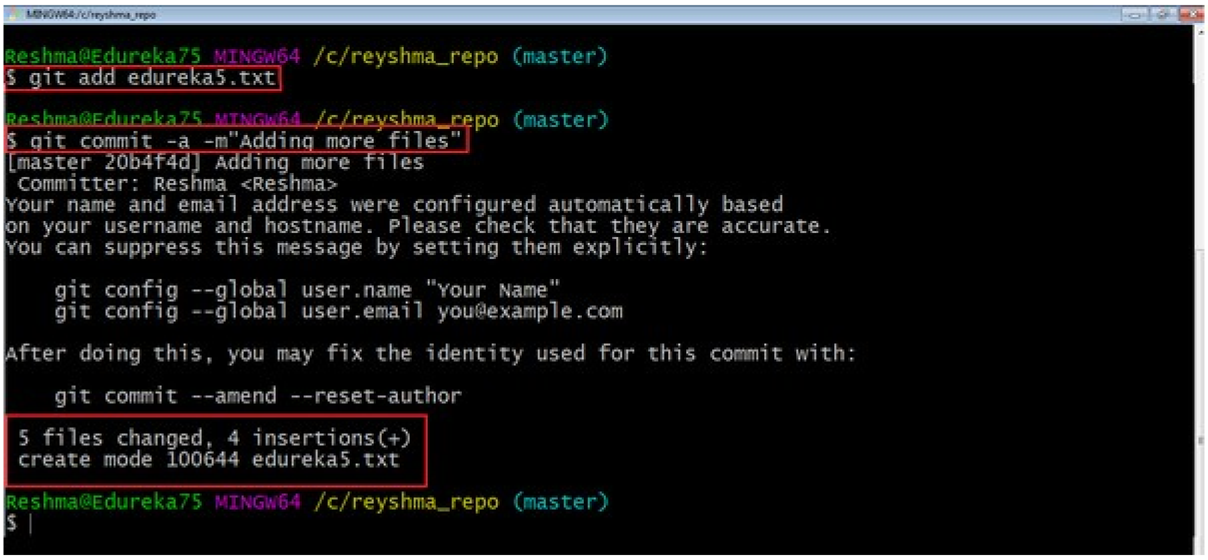
git commit -a

I have created two more text files in my working directory viz. *edureka5.txt* and *edureka6.txt* but they are not added to the index yet.

I am adding edureka5.txt using the command:

git add edureka5.txt

I have added *edureka5.txt* to the index explicitly but not *edureka6.txt* and made changes in the previous files. I want to commit all changes in the directory at once. Refer to the below snapshot.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git add edureka5.txt

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git commit -a -m"Adding more files"
[master 20b4f4d] Adding more files
 Committer: Reshma <Reshma>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 5 files changed, 4 insertions(+)
 create mode 100644 edureka5.txt

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ |
```

This command will commit a snapshot of all changes in the working directory but only includes modifications to tracked files i.e. the files that have been added with **git add** at some point in their history. Hence, *edureka6.txt*was not committed because it was not added to the index yet. But changes in all previous files present in the repository were committed, i.e. *edureka1.txt*, *edureka2.txt*, *edureka3.txt*, *edureka4.txt* and *edureka5.txt*.
Now I have made my desired commits in my local repository.

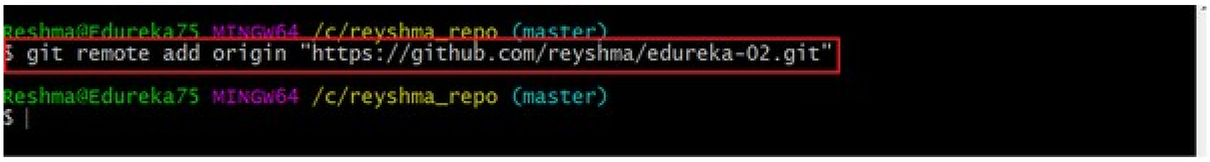
Note that before you affect changes to the central repository you should always pull changes from the central repository to your local repository to get updated with the work of all the collaborators that have been contributing in the central repository. For that we will use the **pull** command.

Pull

The **git pull** command fetches changes from a remote repository to a local repository. It merges upstream changes in your local repository, which is a common task in Git based collaborations.

But first, you need to set your central repository as origin using the command:

git remote add origin <link of your central repository>

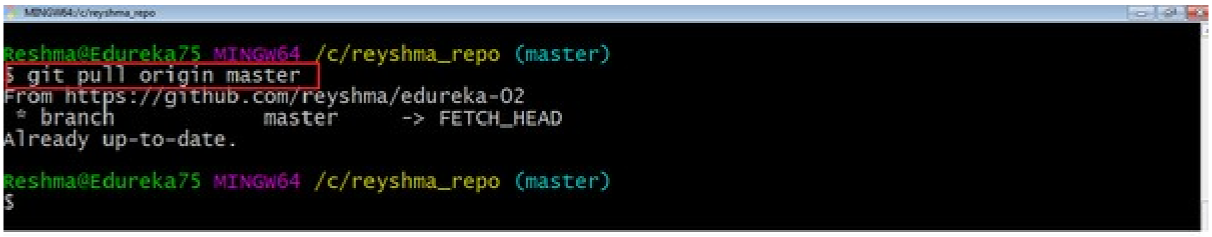


```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git remote add origin "https://github.com/reyshma/edureka-02.git"

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ |
```

Now that my origin is set, let us extract files from the origin using pull. For that use the command:

git pull origin master

This command will copy all the files from the master branch of remote repository to your local repository.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git pull origin master
From https://github.com/reyshma/edureka-02
 * branch            master     -> FETCH_HEAD
Already up-to-date.

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$
```

Since my local repository was already updated with files from master branch, hence the message is Already up-to-date. Refer to the screen shot above.

Note: *One can also try pulling files from a different branch using the following command:*

git pull origin <branch-name>

Your local Git repository is now updated with all the recent changes. It is time you make changes in the central repository by using the **push** command.

Push

This command transfers commits from your local repository to your remote repository. It is the opposite of pull operation.

Pulling imports commits to local repositories whereas pushing exports commits to the remote repositories .

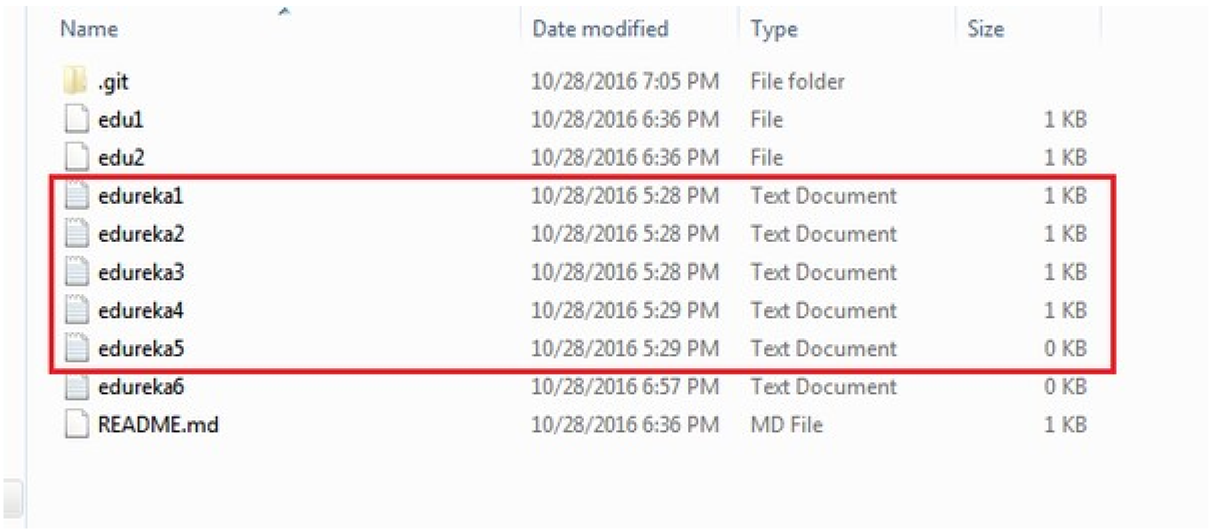
The use of **git push** is to publish your local changes to a central repository. After you've accumulated several local commits and are ready to share them with the rest of the team, you can then push them to the central repository by using the following command:

git push <remote>

Note *: This remote refers to the remote repository which had been set before using the pull command.*

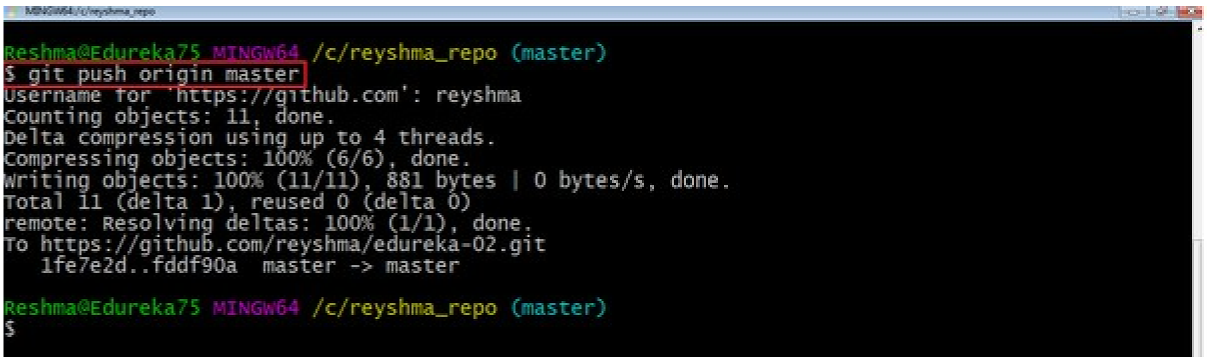
This pushes the changes from the local repository to the remote repository along with all the necessary commits and internal objects. This creates a local branch in the destination repository.

Let me demonstrate it for you.



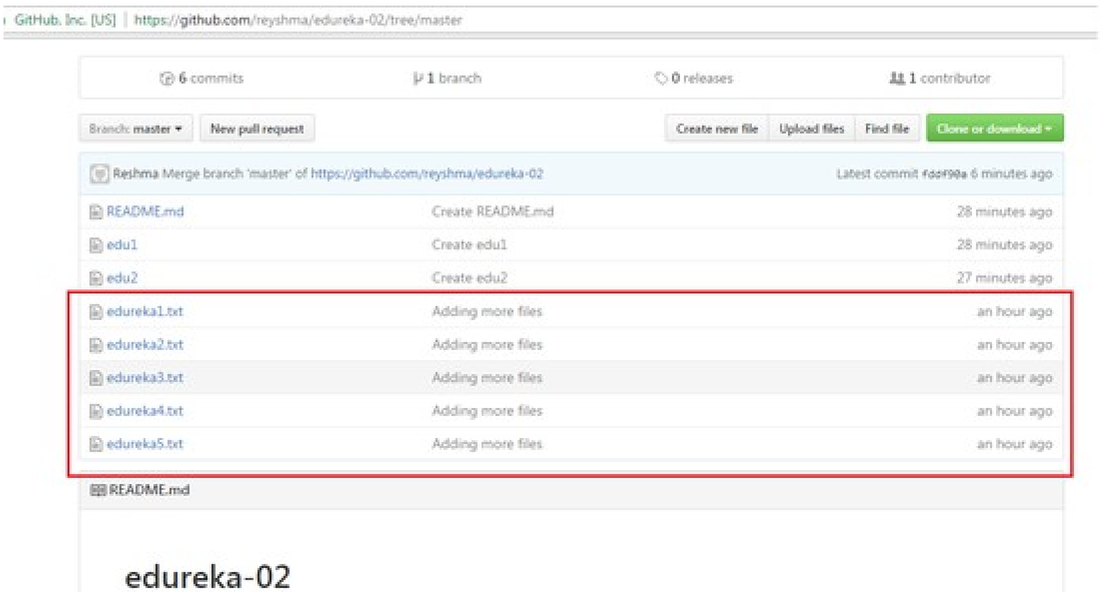
Name	Date modified	Type	Size
.git	10/28/2016 7:05 PM	File folder	
edu1	10/28/2016 6:36 PM	File	1 KB
edu2	10/28/2016 6:36 PM	File	1 KB
edureka1	10/28/2016 5:28 PM	Text Document	1 KB
edureka2	10/28/2016 5:28 PM	Text Document	1 KB
edureka3	10/28/2016 5:28 PM	Text Document	1 KB
edureka4	10/28/2016 5:29 PM	Text Document	1 KB
edureka5	10/28/2016 5:29 PM	Text Document	0 KB
edureka6	10/28/2016 6:57 PM	Text Document	0 KB
README.md	10/28/2016 6:36 PM	MD File	1 KB

The above files are the files which we have already committed previously in the commit section and they are all "*push-ready*". I will use the command **git push origin master** to reflect these files in the master branch of my central repository.



```
Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$ git push origin master
Username for 'https://github.com': reyshma
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (11/11), 881 bytes | 0 bytes/s, done.
Total 11 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/reyshma/edureka-02.git
   1fe7e2d..fddf90a  master -> master

Reshma@Edureka75 MINGW64 /c/reyshma_repo (master)
$
```

Let us now check if the changes took place in my central repository.



```
Reshma Merge branch 'master' of https://github.com/reyshma/edureka-02     Latest commit fddf90a 6 minutes ago
README.md        Create README.md        28 minutes ago
edu1             Create edu1             28 minutes ago
edu2             Create edu2             27 minutes ago
edureka1.txt     Adding more files       an hour ago
edureka2.txt     Adding more files       an hour ago
edureka3.txt     Adding more files       an hour ago
edureka4.txt     Adding more files       an hour ago
edureka5.txt     Adding more files       an hour ago
```

README.md

edureka-02

Yes, it did. :-)

To prevent overwriting, Git does not allow push when it results in a non-fast forward merge in the destination repository.

Note: *A non-fast forward merge means an upstream merge i.e. merging with ancestor or parent branches from a child branch.*

To enable such merge, use the command below:

git push <remote> –force

The above command forces the push operation even if it results in a non-fast forward merge.

If you want to learn how to create branches and how to merge them, read the full blog ***here***.

You can also take a look at this video of mine to learn more. Don’t forget to hit the “Like” button on YouTube if it was helpful to you. :)

I have noticed many answers mention how Git helps you to work together as a team. A few days back I was trying to get around this mystery called Git which was on everyone's lips. **GSoC? Git. Machine Learning? Git. Web Development? Git.** As a trainee under the Spring Fest, IIT Kharagpur web development team we were expected to build our own portfolio website and get it hosted by one of those free web-hosting sites.I chose Hostinger for my website. Now being an impatient brat I couldn't wait to see my website go live. So after every few small changes to the website I would have to zip up the entire folder containing the html files, css stylesheets and a few png files; upload them on the Hostinger server. Then I had to manually redirect the server to load a particular HTML file. Moreover I had to name the compressed folders Upload1, Upload 2. Now you might be getting a hang of what "Version Control" means. What I did by means of naming different folders Upload 1, Upload 2 and so on is a very primitive method of version control. So one fine day I found Hostinger supports Git. Now what I did was created a new repository on Github. Cloned it into a local repository. Then uploaded the files to the websites on the master branch.

Now whenever I would make some significant progress on the website I would update the Git repository with it.

$ git add file_name.extension

after adding the the files I wanted to upadate. I gave the following command

$ git commit -m “Update n”

followed by $ git push origin master

This helped me keep track of my updates and changes.

Here is my website Djokester

And here is the link to the repository Djokester/Portfolio

source of my answer Technical Developer(which is my blog) still quora needs moderation

WHAT IS GIT?

- Git is a tool/software like your IDE or VLC player. It is not a programming language.
- Git is not related to any particular programming language. It is a general tool.
- A very standard definition is: Git is a version control system (VCS). But that's just a Viva definition. We will understand why you should use it
- Companies love it when you have git on your resume.

WHO CREATED GIT?

Linus Trovaldus

He is responsible for the linux kernel.

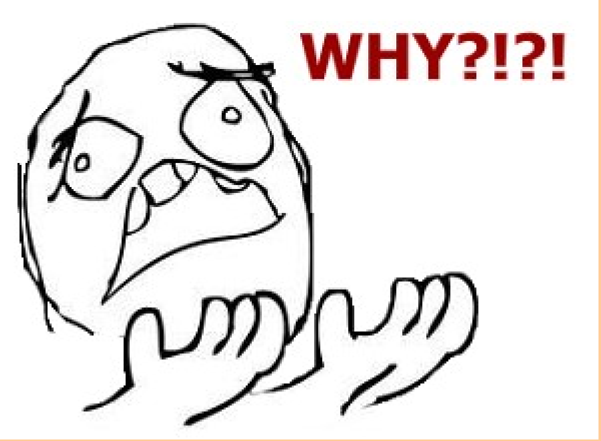
Linux kernel is used by Linux Os (Ubuntu/MacOs), Android

So without his contribution you wouldn't really have that Android phone

You might have also had to pay for whatsapp, facebook or instagram.

WHY IS GIT NEEDED?

I mean, there are already so much to learn, so many languages and frameworks!! Why should I bother with git?



There are two main reasons to learn git

1. Track changes to your code

2. Save your code to the cloud

Tracking changes to your own code is very important. It allows you to do a variety of things such as:

1. Maintain different versions of the same software (like different versions of an app)
2. Make sure you never lose any code, recover code changes at any point of time.
3. Collaborate with others

WHAT IS GIT?

Git is a version control system. In this article, I will focus on Git, which is the most popular version control system.

Version Control is a piece of software that helps teams or individuals manage their source code over time. It tracks changes in a project so that if something breaks while you are working on it, you can compare the difference between the current project and a previous version of the project. You can even "checkout" a previous version of the project and make changes to it. Also, with Git you can have separate "branches," which are separate, parallel "versions" of the project that can be worked on simultaneously. For example, one member of a team can work on a feature at the same time as another person (or the same person) works on a different feature. Once the features are finished, they can both be (hopefully seamlessly) be merged back into the main branch (called the master branch). There are also many, many other features of Git, some of which I will talk about in this tutorial.

Download Git Here.

WHY USE GIT AND GITHUB?

First of all, once you start using Git and GitHub, you'll never stop using it.

With Git and GitHub, you can easily:

- Keep track of your project's history, so you can revert back to an older version if the current version is failing, or if you need to use or compare code from an older version.
- The whole entire project history is stored within your project, in a folder named `.git` It is very hard to lose code with Git.
- Manage different versions. You can even tag different versions with tags like "`v1`, `v2.0.01b`, `b4`, etc." and then you can easily checkout the tagged versions. Or else you can make "releases" on GitHub and then easily link to different releases from your website/wherever.
- Collaborate with others. Collaboration, or at least the basics of collaboration features, is easy

with Git and GitHub.

- Keep different branches. With different branches, you can, for example: have different branches where you work on different features; have one branch for, say, the Python version of the project and one branch for, say, the Ruby version of the project, have one for the documentation, and one for the website…
- Easily duplicate any version of the project into a different branch, make changes, and then merge it back into the original branch.
- Git is extremely fast
- Keep backups of your projects on GitHub - although this is a benefit of using GitHub, don't start thinking of Git as a backup system, as that is *not* what it is.
- Publish or store your source code on one of the world's leading software development platforms: GitHub.
- Have one place online where you keep all your code, and all of its history, collaborators, documentation, etc.
- Git and GitHub is one of the most popular version control systems.
- Tons of other things.

Git is the most commonly used version control system today and is quickly becoming *the* standard for version control. Git is a distributed version control system, meaning your local copy of code is a complete version control repository. These fully-functional local repositories make it is easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository with the copy on the server. This paradigm differs from centralized version control where clients must synchronize code with a server before creating new versions of code.

Every time you save your work, Git creates a commit. A commit is a snapshot of all your files at a point in time. If a file has not changed from one commit to the next, Git uses the previously stored file. This design differs from other systems which store an initial version of a file and keep a record of deltas over time.

Branches

Each developer saves changes their own local code repository. As a result, you can have many different changes based off the same commit. Git provides tools for isolating changes and later merging them back together. *Branches,* which are lightweight pointers to work in progress, manage this separation. Once your work created in a branch is finished, merge it back into your team's main (or master) branch.

Benefits of Git

Simultaneous development

Everyone has their own local copy of code and can work simultaneously on their own branches. Git works when you're offline since almost every operation is local.

Faster releases

Branches allow for flexible and simultaneous development. The main branch contains stable, high-quality code from which you release. Feature branches contain work in progress, which you merge into the main branch upon completion. By separating your release branch from development in progress, you can manage your stable code better and ship updates more quickly.

Built-in integration

Due to its popularity, Git is integrated into most tools and products. Every major IDE has built-in Git support, and many tools that allow you to manage continuous integration, continuous deployment, automated testing, work item tracking, metrics, and reporting feature integration with Git. This integration simplifies your day to day workflow.

Strong community support

Git is open-source and has become the de facto standard for version control, and there is no shortage of tools and resources available for your team to leverage. The volume of community support for Git compared to other version control systems makes it easy to get help when you need it.

Git works with your team

Using Git with a source code management tool can increase your team's productivity by encouraging collaboration, enforcing policies, automating processes, and improving visibility and traceability of work. You may choose individual tools for version control, work item tracking, and continuous integration and deployment.

Use pull requests to discuss code changes with your team before merging them into your main branch. The discussions you have in pull requests are invaluable to ensuring code quality and increase knowledge across your team. Visual Studio Team Services offers a rich pull request experience where you can browse file changes, leave comments, inspect commits, view builds, and vote to approve the code.

What is Git

Simply git is a tool which is used in web development but not just a simple tool its one of the most popular git revision control and source code management system which keeps all the old and new versions of code and track who changed the code when changed the code and what part of code developer changed and also can see commit log to know why he changed and why he made that changes in code. Git was initially made by Linux Torvalds for Linux Kernal development.

Why to Use Git

There are many tools available in market right now like Git to revision control and SCM (source code management) but why Git is most popular ? Well the reason is :

- Git tracks state, history and integrate of the source tree
- Git keeps old versions for you if some developer did any mistake in code then you'll always have previous version to fix it
- Multiple developers can work together, once they write code in their local machine and commit it then other developers can pull it easily.
- Large developers community and online websites to upload your source codes or get others source codes to make your work easier
- Lots of software available for both who comfortable with command line and for others GUI tools
- Easy and clear documentation to get started with
- Git will not use much bandwidth you don't have to connect with your server always you just need to connect to push code when you are done

Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005 to develop Linux Kernel. Git has the functionality, performance, security and flexibility that most teams and individual developers need.

The best way to learn Git is to try out all the git commands step by step. Let me show you how -

Some of the basic operations in Git are:

1. Initialize
2. Add
3. Commit
4. Pull

5. Push

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing

Let me first give you a brief idea about how these operations work with the Git repositories.

What is git?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

How does it work?

Most version control systems work by calculating the differences in each file, and then by summing up these differences they can reach whichever version saved, but that's not the way Git works.

Git functions as a series of snapshots for your file system. Every time you change something in your files committing it, or just saving the state of your project, Git takes a picture of the whole system and saves a reference to it. If there's a file that hasn't been changed, then it stores a reference to the previous snapshot. We'll get to how this is one of the most powerful and enabling features in Git when we dive deeper into the features Git provides.

Why you should use it?

- Distributed model: This means your work is your own. You can let others see only what is necessary. Not everything has to be public. There are other advantages to the distributed model, such as the speed (since most everything is local) and possibility of working offline
- Branching and merging are easy: Branching is a walk in the park. It feels like a natural part of the workflow. They are cheap (fast and consume very little space) so that you can branch whenever you want. This means you can sandbox your features and ideas till they are ready for the mainstream.
- Workflow is flexible: Compared to Centralized VCS, git has the qualities that allow to choose your own workflow. It can be as simple as a centralised workflow to as hierarchical as the dictator-lieutenant workflow. Use the process that best fits you.