GİT

**What is Git?**

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

Git is a distributed version control system for **tracking changes in source code** during software development. It is designed for coordinating work among programmers, but it can be used to track change.

Git has the functionality, performance, security and flexibility that most teams and individual developers need. Git is the most broadly adopted tool of its kind. This makes Git attractive for the following reasons. Vast numbers of developers already have Git experience and a significant proportion of college graduates may have experience with only Git.

**Brief History of Git**

As with many great things in life, Git began with a bit of creative destruction and fiery(ateşli) controversy(tartışma).

For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, the Linux kernel project began using a proprietary(tescilli) Version Control System called BitKeeper.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool’s free-of-charge status was revoked. This prompted the Linux development community (and in particular **Linus Torvalds**, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. **That's how Git was borned**.

**More About Git**

Git is an example of version control. Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

It allows you to:

* Revert files to previous state,
* Revert entire project back to previous state,
* Compare changes over time,
* See who modified what? **And much more...**

It means if you screw things up or lose files, you can easily recover.

Use cases:

* Individual development,
* Collaborative development,
* Offline usage.

Why Git?

* Everything is local (full history tree available offline),
* Everything is fast,
* Snapshots, not diffs,
* It is distributed not centralized,
* Great for those who hate: CVS/SVN (earlier [version control systems](https://lms.clarusway.com/mod/lesson/view.php?id=642)).

**Version Control Systems**

**What is a Version Control System (VCS)?**

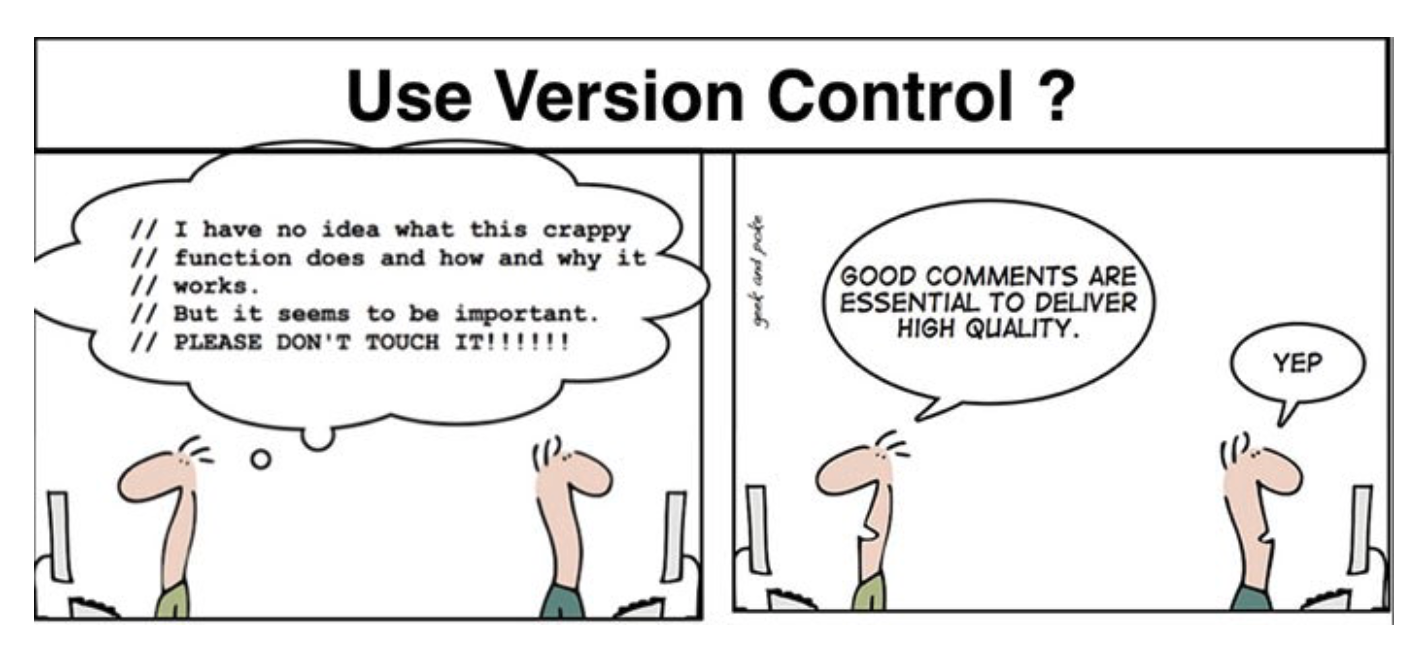
A **V**ersion **C**ontrol **S**ystem (also called Revision/Source Control System) is a software designed to record changes made to files over time so that you can recall specific versions later. It's like having an unlimited undo/redo feature under your belt.  It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. It's a very powerful tool for your project.

A VCS is a tool to track changes of source code and learning

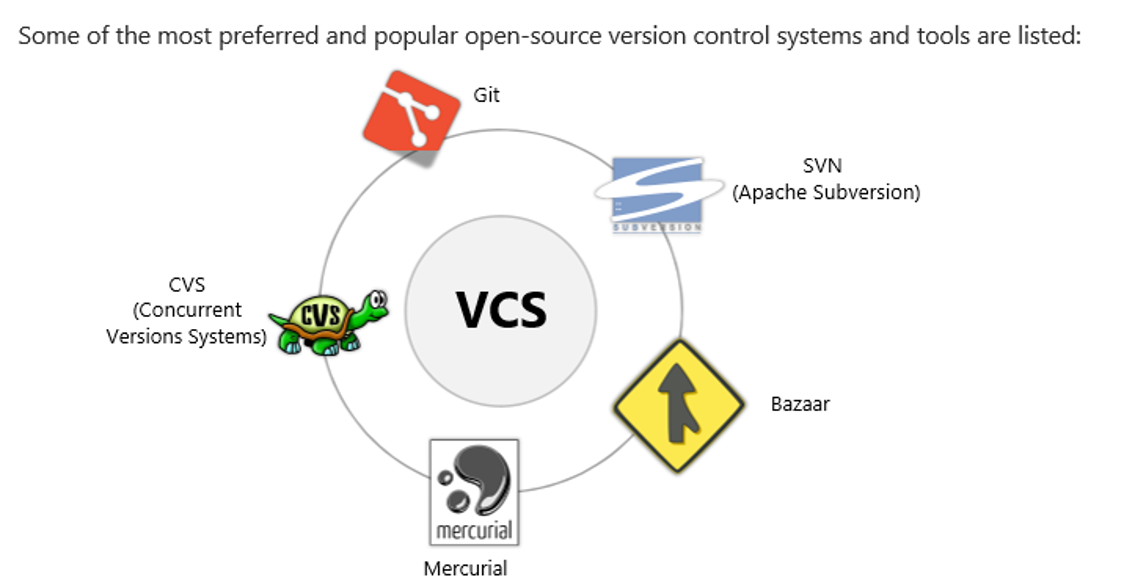
* What had changed
* When it changed
* Why it changed
* Who changed it

**Need of Version Control Systems**

There are many reasons to use a **Version Control System (VCS)** in a system. Here are the reasons to use it. Files need to be stored somewhere. You can store them anywhere you like , but if you store them in a VCS, you never lose them. Every reported change (check-in) associated with a file are also available. All the previous versions of a file can be easily extracted/restored.



**Popular Version Control Systems**



Formun Üstü

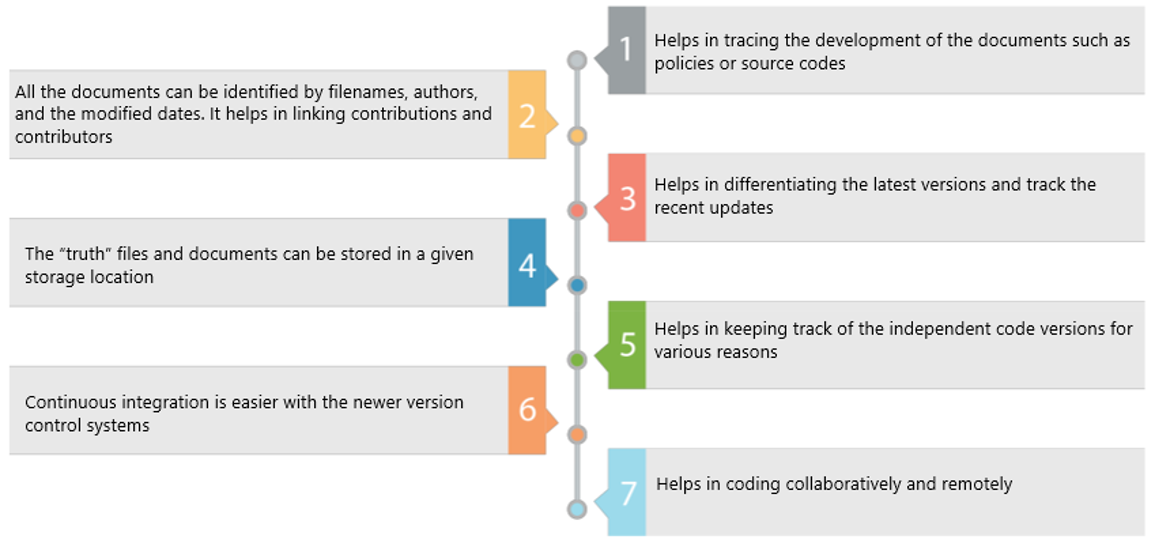
Formun Altı

Formun Üstü

Formun Altı

**Importance of Version Control Systems**

Why is the VCS important. Because it helps us.



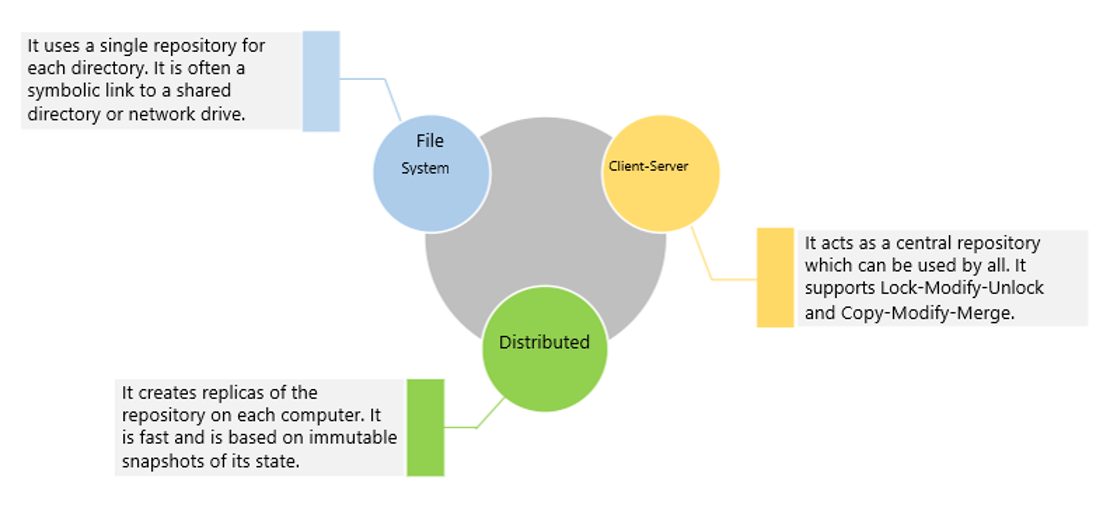
Types of Version Control Systems

There are **three** types of version control systems:

* **File-based,**
* **Client-server type,**
* **Distributed.**

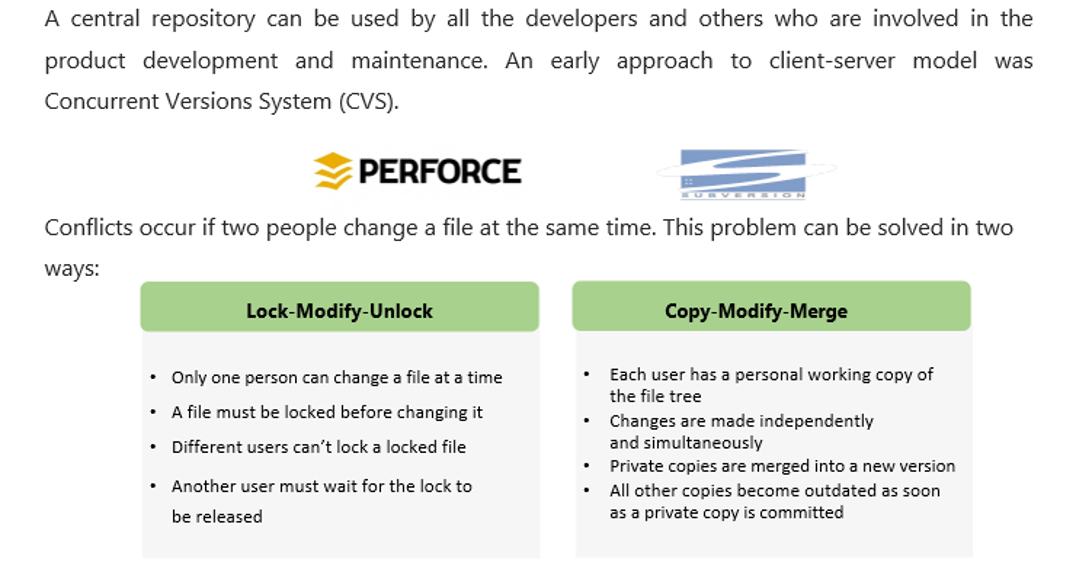
File-based version control systems are obsolete(modası geçmiş) and not much used. Distributed VCS's are more common on the market. Git is a distributed type of VCS.

Repository: depo



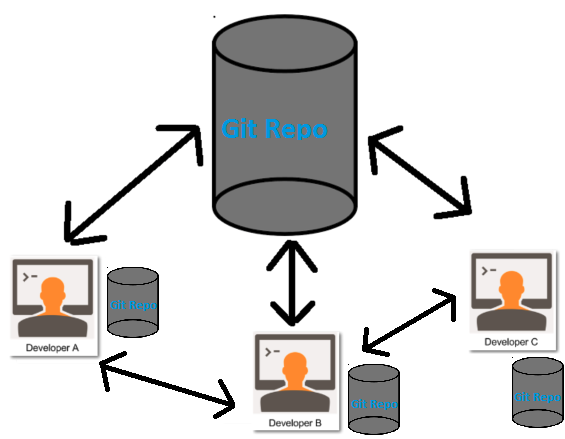
**Client-Server Based Version Control Systems**

These are properties of Client-Server Based Version Control Systems.



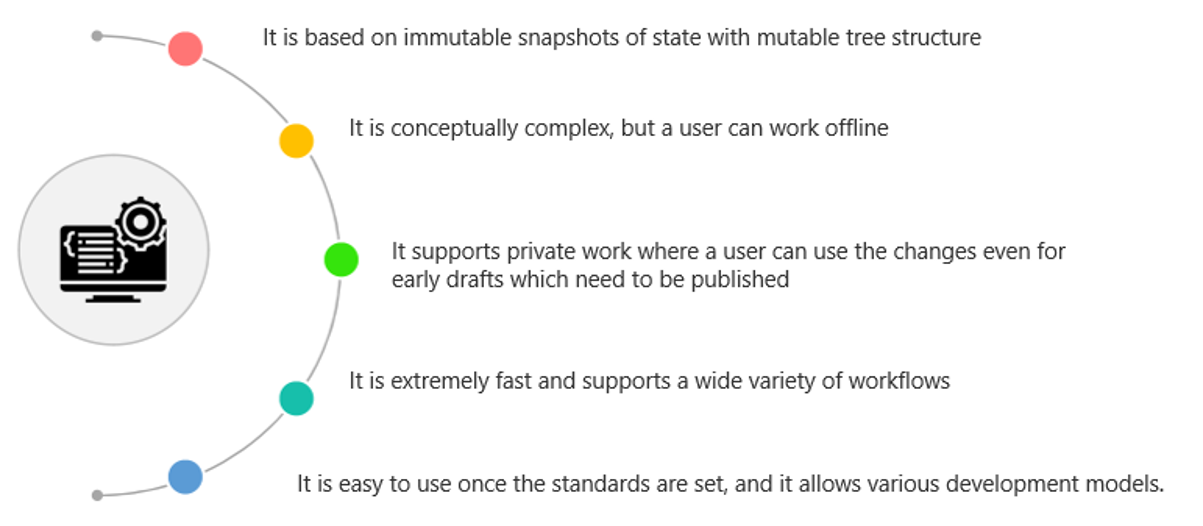
**Distributed Version Control Systems**

Distributed version control systems create replicas of the repository on each computer. Every user has to work on a replica and can do so even being disconnected from the network. They are suited for large projects and independent developers who can work independently and commit the changes for merging.



**Features of Distributed VCS**

Here are some features of Distributed VCS.



**Mac:**

<https://git-scm.com/download/mac>

When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.

**Linux:**

For Ubuntu:

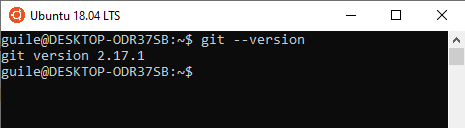
**$ sudo apt-get install git**

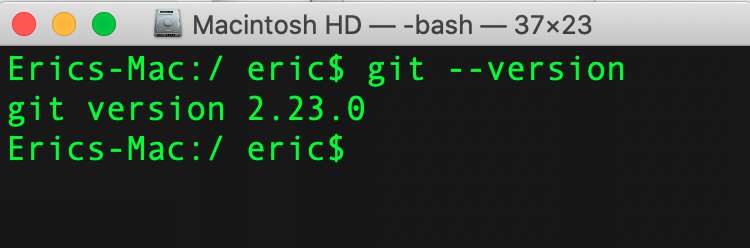
For Fedora (works also on **AWS Linux 2** Free Tier AMI):

**$ sudo yum install git**

Once installed, check Git Version:

**$ git –version**





If you see any version number, you are good to go! 👍

You can set your username and email as follows :

**$ git config --global user.name "Your Name"**

**$ git config --global user.email "youremail@example.com"**

Enable color in git:

**$ git config --global color.ui auto**

Checking your settings

**$ git config --list**

(press **q** if you need to exit from this list)

**What is a Repository?**

A repository is a directory or storage space where your projects can live. Sometimes it is shortened to “**repo**.” It can be local to a folder on your computer, or it can be a storage space on the cloud. You can keep code files, text files, image files, etc. inside a repository.

Let's create a local repo and see what it looks like.

Go to your desktop and create a folder named "git-projects".

**$ mkdir git-projects**

Get into the folder.

**$ cd git-projects**

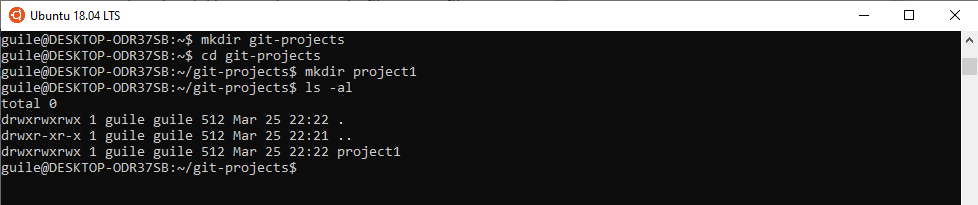
Make another folder called "project1".

**$ mkdir project1**

And check what is inside the "git-projects" folder.

**$ ls -al**

This is what you see **without** a repo.



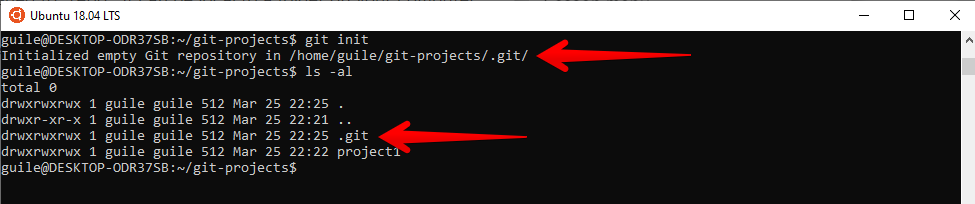
To create a repo, type

**$ git init**

And to see the updated files, type

**$ ls -al**

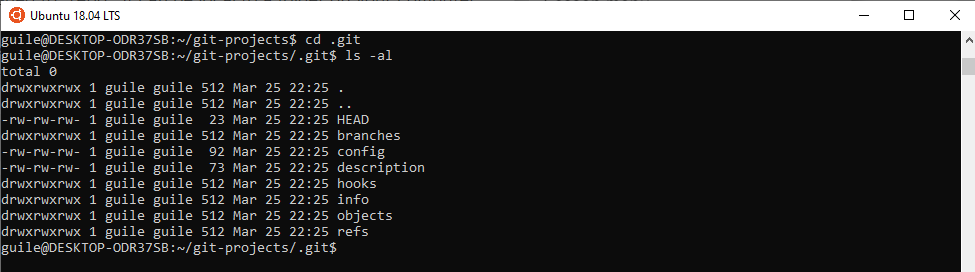
Now you have a repo named .git. It's a secret folder consists of many files and folders.



Let's check what is inside the repo.

**$ cd .git**

**$ ls -al**



And this is how a **Repo** looks like. Git understands all the versions of the files and folders in your project and saves them in a special way. All your files and folders will be stored in Git Database.

**Understanding How Git Manages Data**

In a Git repository your file can reside in three main states: **Modified**, **Staged**, and **Committed**.

* **Modified** means that you have changed the file but have not committed it to your database (repo) yet.
* **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
* **Committed** means that the data is safely stored in your local database.

This leads us to the three main sections of a Git project:

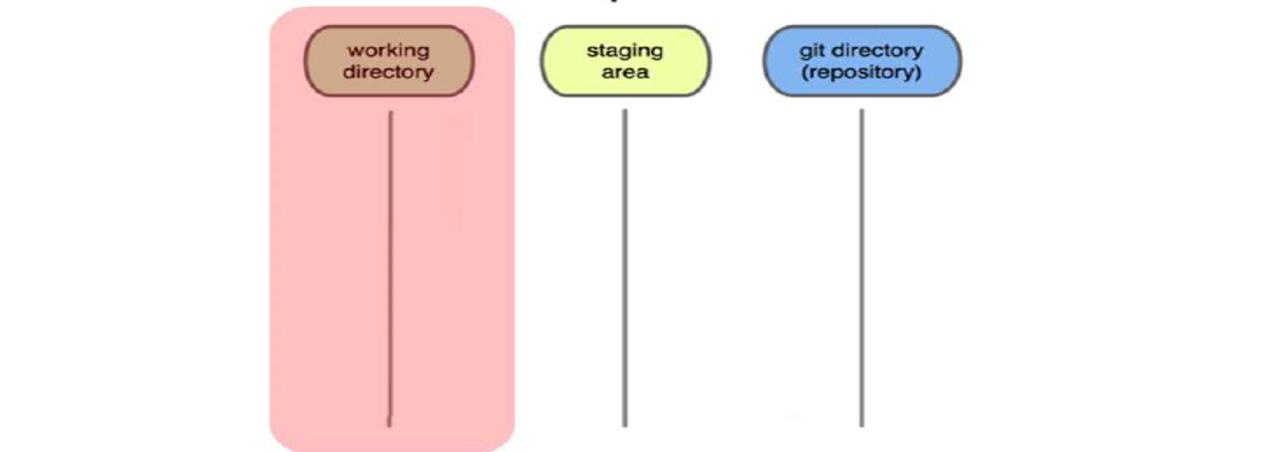
* The working tree,
* The staging area,
* The Git directory.

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.

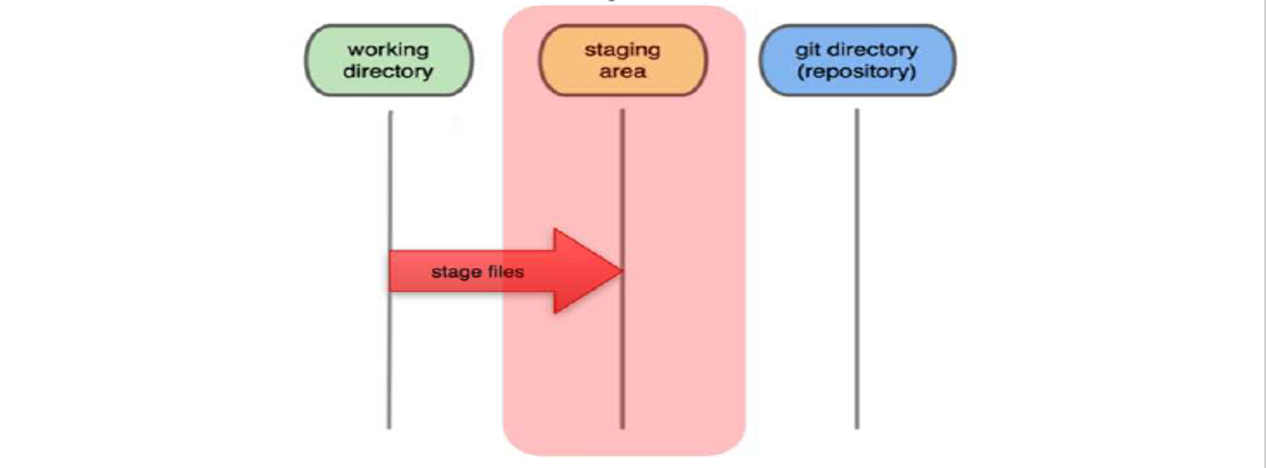
The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git terminology is the “index”, but the phrase “staging area” works just as well.

The Git directory (.git) is where Git stores the metadata and object database for your project.

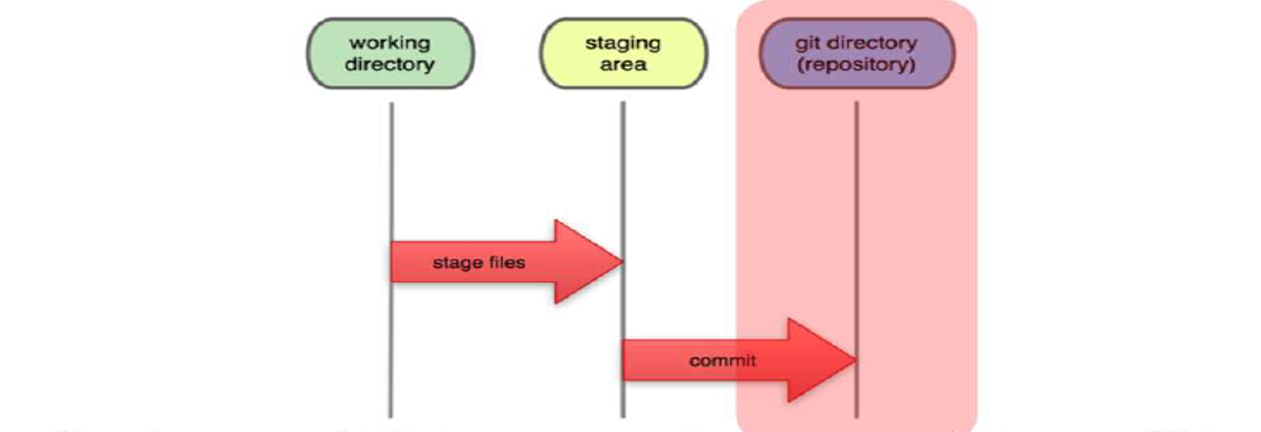
You modify files in your working directory.



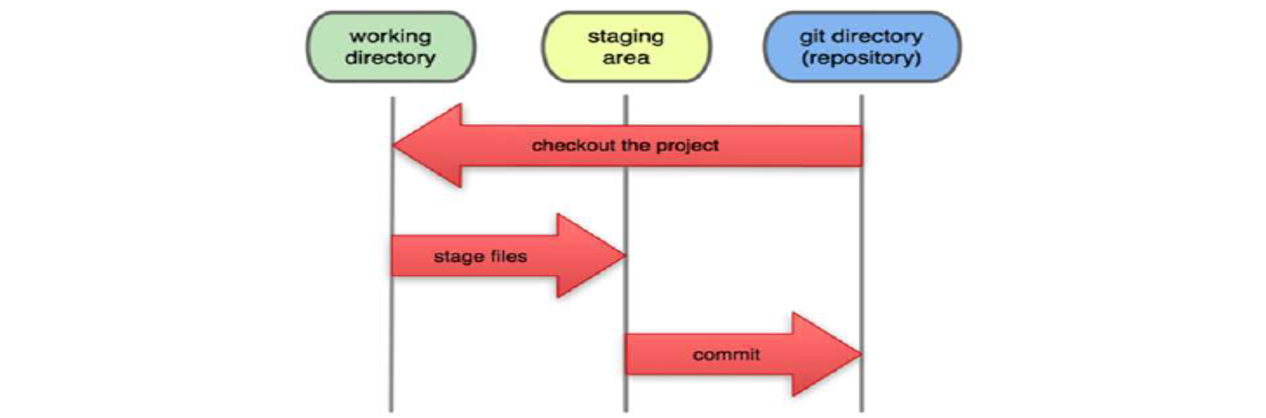
After you are done with your file in your working directory you add them to staging area, you can think of this as a temporary storage for your files. You stage the files, adding snapshots of them to your staging area.



Then you are sure that the files you have in your staging area are ready to go next step then you commit your files to your git local repo. You do a commit that stores snapshot permanently to your Git directory.



Once you commit your files to your local repo you can then checkout them at any time you can modify them then add and commit again as you wish.



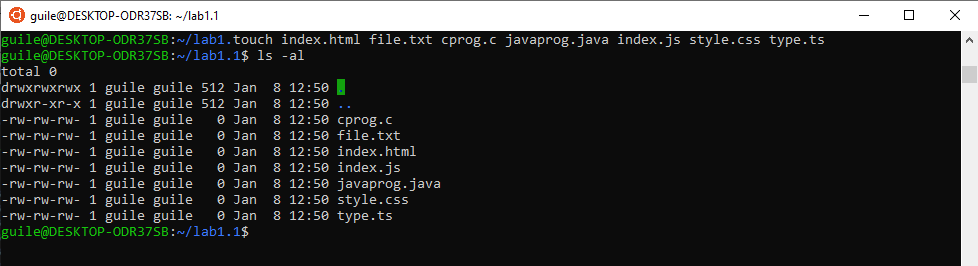
**Configuring Local Git Instance**

Now lets move on our local git instance configuration,

Lets create some files by entering "touch" command.

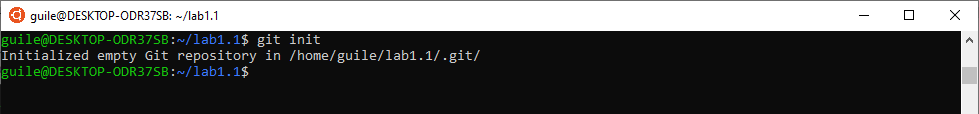
**$ touch index.html file.txt cprog.c javaprog.java index.js style.css type.ts**

**$ ls –al**



You then execute "git init" command to create an empty local repo.

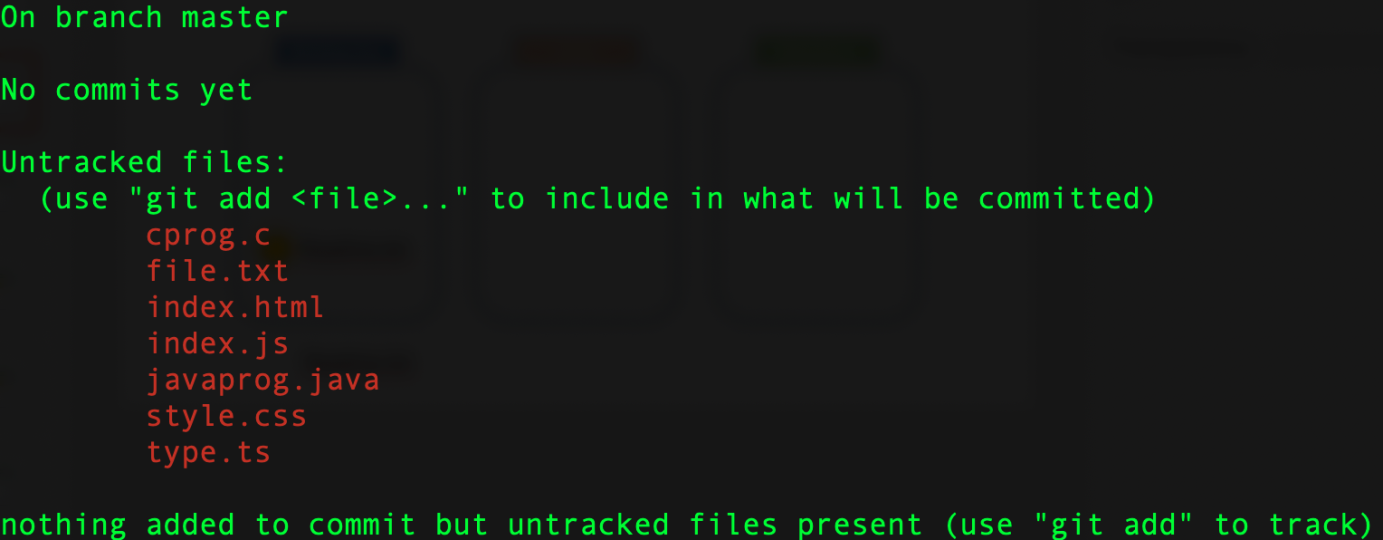
**$ git init**



Initialized empty Git repository in "home/guile/lab1.1/.git".

Check git status.

**$ git status**



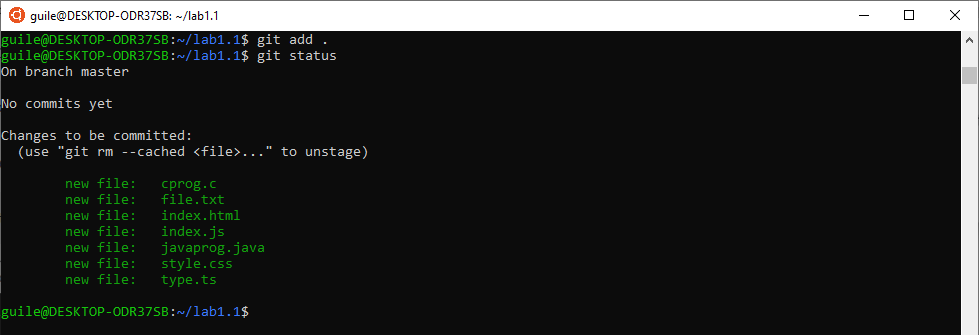
Those files colored red are sitting on working area but not being tracked by Git.

"Git add" to add the files to staging area from the working area.

**$ git add .**

And now we can see the new status of the files.(Notice green color)

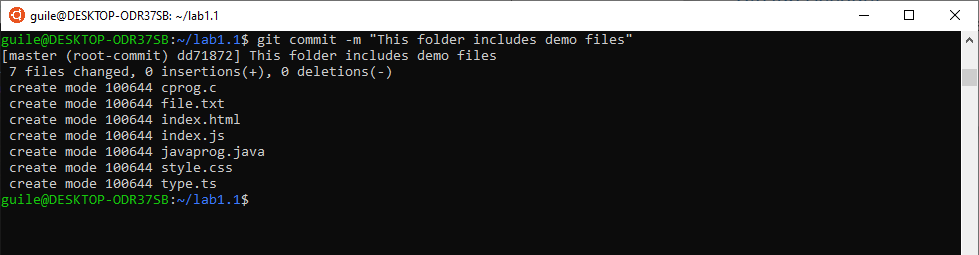
**$ git status**



Let's commit our files to local repo with the command

git commit –m "message/explanation"

**$ git commit –m** “This folder includes demo files”



Congratulations! You have completed your first commit to your local git repo.