Introduction to Git

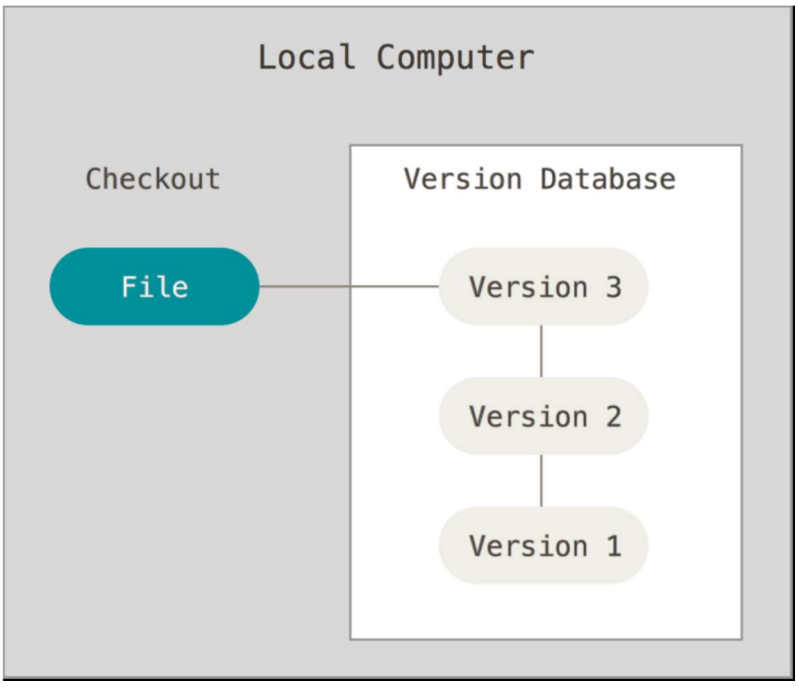
What is Version Control?

Version control systems are a category of software tools that help a software team manage **changes to source code** over time.

Version control software keeps track of every modification to the code in a special kind of database.

If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake.

Therefore, 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. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.



What is Git?

By far, the most widely used modern version control system in the world today is **Git**.

Git stores and thinks about information much differently than other systems.

Most other systems store information as a list of **file-based changes**. Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a set of **snapshots** of a miniature filesystem.

This makes Git much more efficient, and hence more commonly used.

Some basic terms

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

Committed means that the data is safely stored in your local database.

Modified means that you have changed the file but have not committed it to your database yet.

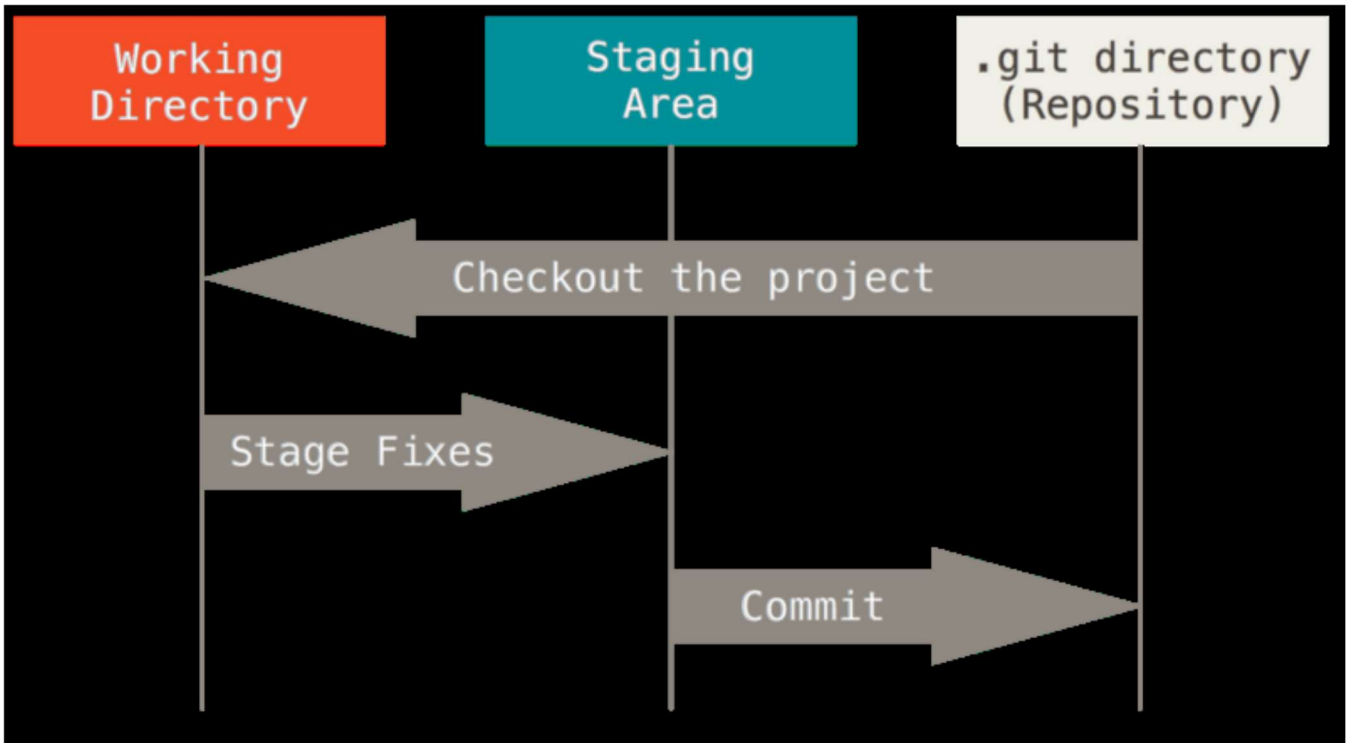
Staged means that you have marked a **modified** file in its current version to go into your next **commit** into the database.

The basic Git workflow goes something like this:

1. You **modify** files in your **working directory**.

2. You **stage** the files, **adding** snapshots of them to your **staging area**.

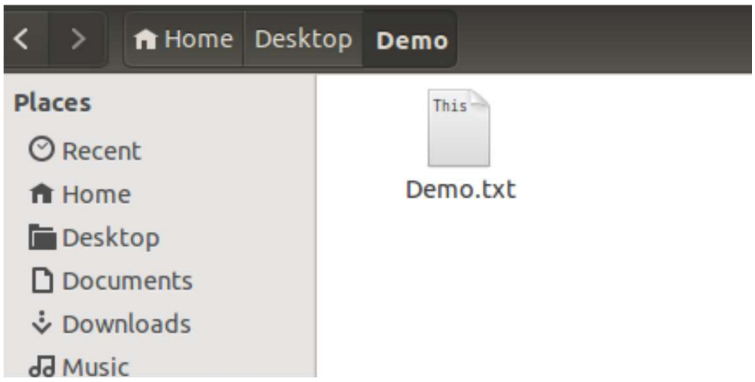
3. You do a **commit**, which takes the files as they are in the staging area and stores that **snapshot** permanently to your Git directory (repository).



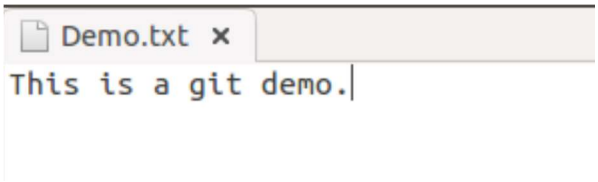
Basic git commands

Let us talk about the various steps in a typical Git workflow, and the Git commands associated with them.

1. First, we create/modify one or more files in our **working directory:** Suppose that the current working directory is **Desktop/Demo/**



We have created a file Demo.txt, whose contents are:



2. The next step would be to open **Terminal** (For Ubuntu/Mac) or **Git Bash** (For windows).
   This would be followed by switching to the current working directory using **cd**. The **ls** command lists the files and subfolders in the current folder.

   $ cd Desktop/Demo

   $ ls

   Demo.txt

3. Now, we will initialise a git repository in this directory. This is done using the command- **git init**

   $ git init

   Initialized empty Git repositor in /home/username/Desktop/Demo/.git/

4. We will then add Demo.txt from the working directory to the staging area, using the **git add** command:

   $ git add Demo.txt

   Alternatively, we can add **all files** from the working directory to the staging area, using:

   git add .

   OR

git add --all

When committing an **entire project**, we should use these commands.

5. Check the current status of the staging area using **git status**:

   $ git status

   On branch master

   Initial commit

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

   new file: Demo.txt

   This tells you that Demo.txt is a new file added to the staging area, and is ready to be committed to the Git repository.

6. We then commit the files in the staging area to the repository.

   This is done using the **git commit** command. This is usually followed by a message describing the changes made to the file, something like:

   git commit -m “message”

   Here, we will give a meaningful message to our first commit, which is the first version of the file.

   $ git commit -m "Demo.txt file created"

   [master (root-commit) 2312eeb] Demo.txt file created
   1 file changed, 1 insertion(+)
   create mode 100644 Demo.txt

7. Check the history of all commits made, using **git log**:

   $ git log

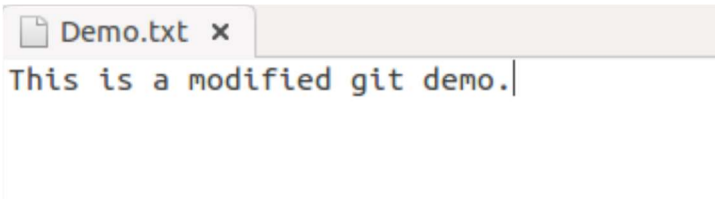
   commit 2312eeb9ca38dc986a174e4219c9468834a3812f
   Author: username <email>
   Date: Fri Dec 9 15:02:32 2016 +0530

   Demo.txt file created

   Currently, only a single commit has been made.

   Here ***2312eeb9ca38dc986a174e4219c9468834a3812f*** is the **unique ID** or **SHA** of the commit.

8. Now, suppose we make some changes to the file **Demo.txt**



9. This would be followed by the **git add**, **git status** and **git commit** commands:

   $ git add Demo.txt

   $ git status

   On branch master
   Changes to be committed:
   (use "git reset HEAD <file>..." to unstage)

   modified: Demo.txt

   Notice how Git acknowledges that Demo.txt has been modified.

10. We now commit the changes to the repository.

    $ git commit -m "Modified the contents of Demo.txt"

    [master c1bbd61] Modified the contents of Demo.txt
    1 file changed, 1 insertion(+), 1 deletion(-)

11. Now, we recheck the log:

    $ git log

    commit c1bbd6106a6bf4356edefe733d533601d386c7ad
    Author: username <email>
    Date: Fri Dec 9 15:14:14 2016 +0530

        Modified the contents of Demo.txt

    commit 2312eeb9ca38dc986a174e4219c9468834a3812f
    Author: username <email>
    Date: Fri Dec 9 15:02:32 2016 +0530

        Demo.txt file created

    Both commits are shown in the log.

12. Now, suppose this modification wasn't desired. We would like to **roll back** to the previous version.

    For that, we would perform a **hard reset**. This is done using **git reset --hard *commitID*.** Here the commit ID of the first commit is: ***2312eeb9ca38dc986a174e4219c9468834a3812f***

    $ git reset --hard 2312eeb9ca38dc986a174e4219c9468834a3812f

    HEAD is now at 2312eeb Demo.txt file created

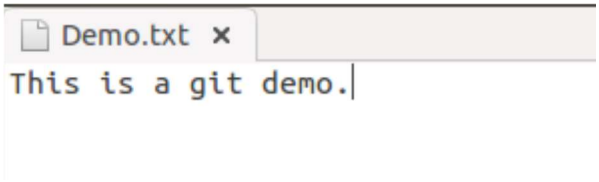
13. On rechecking the log:

$ git log

commit 2312eeb9ca38dc986a174e4219c9468834a3812f
Author: username <email>
Date: Fri Dec 9 15:02:32 2016 +0530

Demo.txt file created

Voila! You have now rolled back to the first version of the file. Open the file to confirm:



We now have a basic idea about Git and its most commonly used commands. There are many more commands that you will explore as you go further into open source development. For now, these commands should suffice.

In the next tutorial, we would take a look at **Github** and how to integrate a Github repository with your local Git repository.