# Git Concepts and Workflow for Beginners

## Learn the most popular git commands and a simple but effective branching model

Last updated on Aug 21, 2020 by Juan Cruz Martinez -

Software developers usually write tons of code every single day. They might be working on a new project or tweaking an existing one. But, after some time the codebase may grow enough that it is very difficult to manage or track any changes.

For example, what if a team of software developers is working on an application. Now, how can they track the team member who made a specific change?

It might not be an issue if a team only consists of two or three members. But, just imagine if they are around 10, 20, or even 1000?

Now, it seems like a daunting task. Isn’t it?

You don’t have to worry because this is where a **Version Control System (VCS)** comes into play.

Before executing any commands I think it’s better to first talk about the concept of VCS and Git. It will help you understand why we need it and how can it improve our existing workflow.

. . .

## What is a Version Control System?

It’s just a software that automatically maintains a record of every change in a project. This way, a team can focus on creating an actual product instead of dealing with non-productive tasks like who modified a specific file.

Also, as VCS is a computer software so it is more accurate and faster than us. Anyways, let’s have a look at different types of Version Control Systems.

1. **Centralized Version Control**

   As its name suggests, in the Centralized Version Control system a project is stored in a single/central location (e.g. Server). Here, a team member can download files that he wants to change and after doing the work simply upload them to the server.

   The most popular CVC software is known as Subversion .

2. **Distributed Version Control**

   In a Distributed Version Control System (DVCS) every team member has access to a full copy of the project on their local computer. It doesn’t require a central server but you might need one to easily collaborate with your teammates.

These days, the concept of DVCS is very popular due to the fact that it enables us to work on a project without worrying about an internet connection. We only need to connect whenever we want to upload changes to the server.

Git is an example of a Distributed Version Control System which you will learn today.

. . .

## Overview of Git

Git is the most commonly used Distributed Version Control System. It keeps a record of every change in a project by taking snapshots. It means that you can easily roll back to any previous version/state of a project.

In 2005, Linus Torvalds released the first version of Git. Don’t forget he is the same person who created the Linux operating system.

At first, Git was intended for use by software development teams. But, it is flexible enough to track any kind of file. So, no matter whether you are a teacher, businessman, graphic designer, or a content writer, Git can help you automatically track your projects.

. . .

## Collaborate With Teammates

In this section, you will learn how team members can use Git for collaboration. Basically, I’ll help you understand the actual workflow of Git. Later on, you’ll see how Git provides us predefined commands for each purpose. So, let’s get started.

1. Create a “Repository” on the central server. A repository is just a folder that is being tracked by Git software.

2. Download the repository on your computer. In terms of VCS, we usually refer to this step as “cloning”.
3. Add files in this repository and commit them. Git takes a snapshot of the project whenever you commit something. It then maintains a history of snapshots for later use.
4. Upload the modified files on the server.
5. Download any files that are modified by your teammates.

So, this is the basic workflow of any project that uses Git for version control. Now, let’s see how we can actually implement these concepts using Git commands.

. . .

## Create a Repository

By default, Git software doesn’t track each and every file on your computer. We’ve to inform Git about the specific folder that it must track.

To do so, open your command prompt or terminal inside the specific folder and execute the below command.

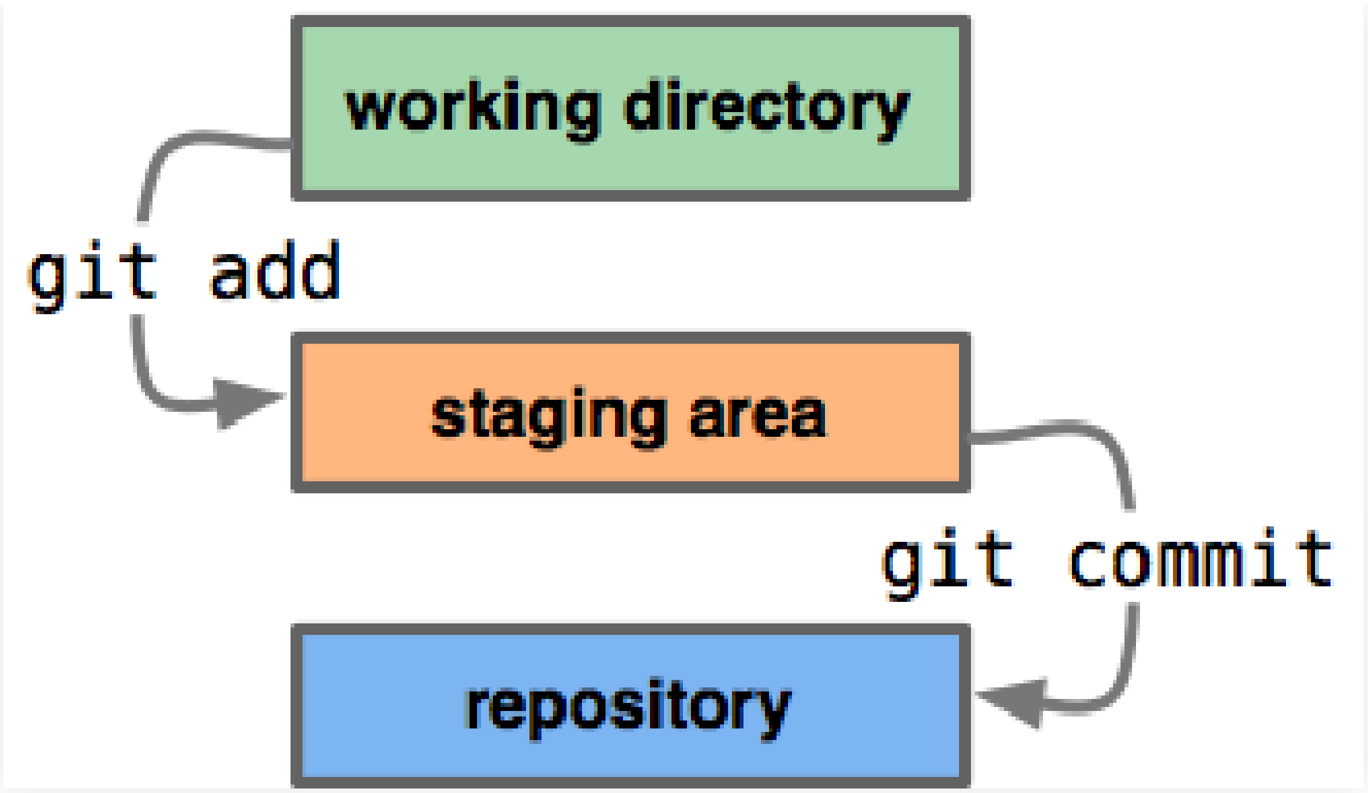
```
git init
```

This command converts the current directory into a repository. Basically, it tells the Git software to start tracking all files and folders inside this repository.

*A point to be noted is that all Git commands start with the keyword `git`.*

. . .

## Working Directory, Staging Area, and Repository



Git development environment sections

Basically, a Git development environment is consists of three sections. The concept behind these sections is of key importance. So, pay extra attention because everything else will depend on this.

1. **Working Directory**

   It simply refers to the current state of files and folders inside your file system. At this point, Git doesn't track these files.

2. **Staging Area**

   Before saving any file to a repository, you have to place it in a staging area. It's like a temporary location for your files and folders before commit. You can easily add or remove files from a staging area.

   If you've modified any files and you want to add them in a staging area then simply execute the below command.

   ```
   git add .
   ```

Here, the dot means all files that are modified.

3. **Repository**

   A repository holds your actual committed files. Git stores all this information inside a hidden folder called `.git`. When you commit something, whatever inside your staging area is permanently saved in a repository.

   You can think of a commit as a checkpoint. So basically, git compares your previous checkpoint with recent commit to only store the modified files.

   You can run the below command with a brief message about what is changed.

   ```
   git commit -m "Type a brief message here regarding this commit"
   ```

   Commit messages are really helpful as it informs other team members about what you did in the commit.

. . .

## Check Status

Git allows you to easily check which files are tracked/un-tracked inside a staging area. To do so, you can use the below command.

```
git status
```

Many people use it frequently to get the current status of the staging area.

. . .

## Connect Local Repository with Server

In a real-world scenario, you may find it easier to first create a Repository on a Git hosting provider (e.g. GitHub or Bitbucket ). Then, simply clone it to your local computer. The reason is that they automatically set the `remote` for you. Meaning that your local and remote repositories are connected by default.

But, in case you have created a repository on your local computer and then want to connect it with a server. Then, run the below command.

```
git remote add [remote_name] [remote_url]
```

You can specify anything as a `[remote_name]`. But, conventionally we name it as the `origin`.

Whereas, `remote_url` is the path to your Git repository on the server. This URL will be provided by your Git hosting provider.

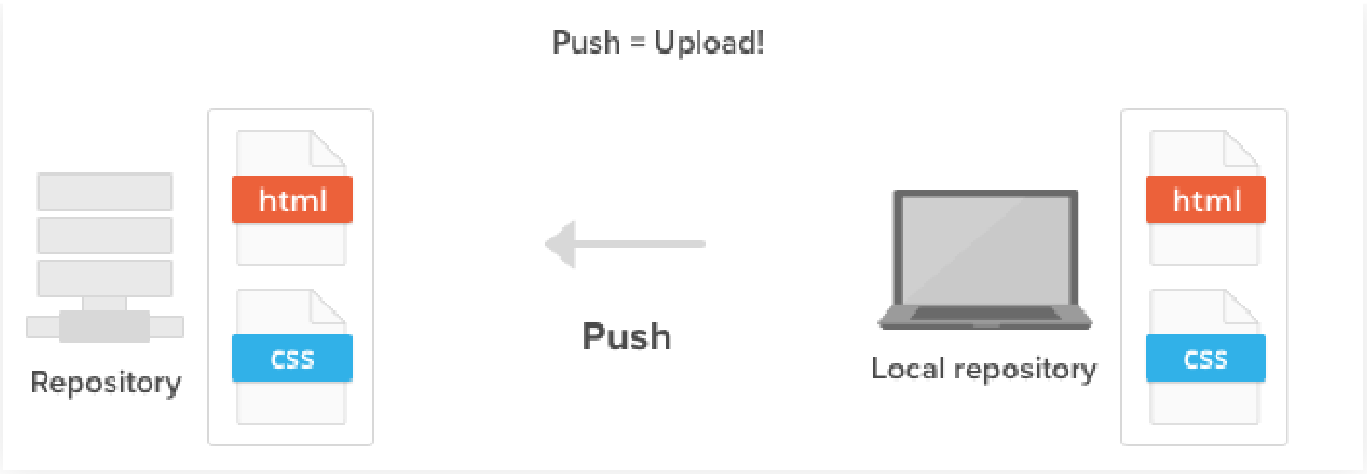
Here's an example of how this command will look in a real-world scenario.

```
git remote add origin https://github.com/twbs/bootstrap.git
```

In case, you want to check how many remote repositories are connected with your local repository. Simply use the below command.

```
git remote -v
```

## Push Changes to Server



Git push

Now that you have connected your local repository with a remote one. It's time to upload changes to the server. Have a look at the syntax of this command.

```
git push [remote_name] [branch_name]
```

It requires the name of your "remote" and "branch" where it will push the changes. By default, we only have one branch which is known as "master". You'll learn about branches later in this tutorial.

So, an example of this command will look like this.

```
git push origin master
```

## Pull Changes from Server

Git pull

In case your teammates are also working on the same project and they have pushed some changes to the server. You can retrieve those changes by using this command.

```
git pull [remote_name] [branch_name]
```

For example:-

```
git pull origin master
```

By default, when you run this command, Git downloads the modified files and then merge it with your repository. But, if you just want to check whether something is changed on the remote repository or not then use this command instead.

```
git fetch origin master
```

. . .

## Branches

Git branching model

Git branching model is similar to that of a Tree with branches. They are particularly useful when you want to add a new feature or fix a bug in your project.

Basically, you create a new branch to separately test something. When your work is done then simply merge it with your main branch (i.e. master). In case, you don't like the new idea then you can easily discard the changes without affecting the project.

Use this command to create a new branch.

```
git branch <branch_name>
```

You can also delete a branch using Git.

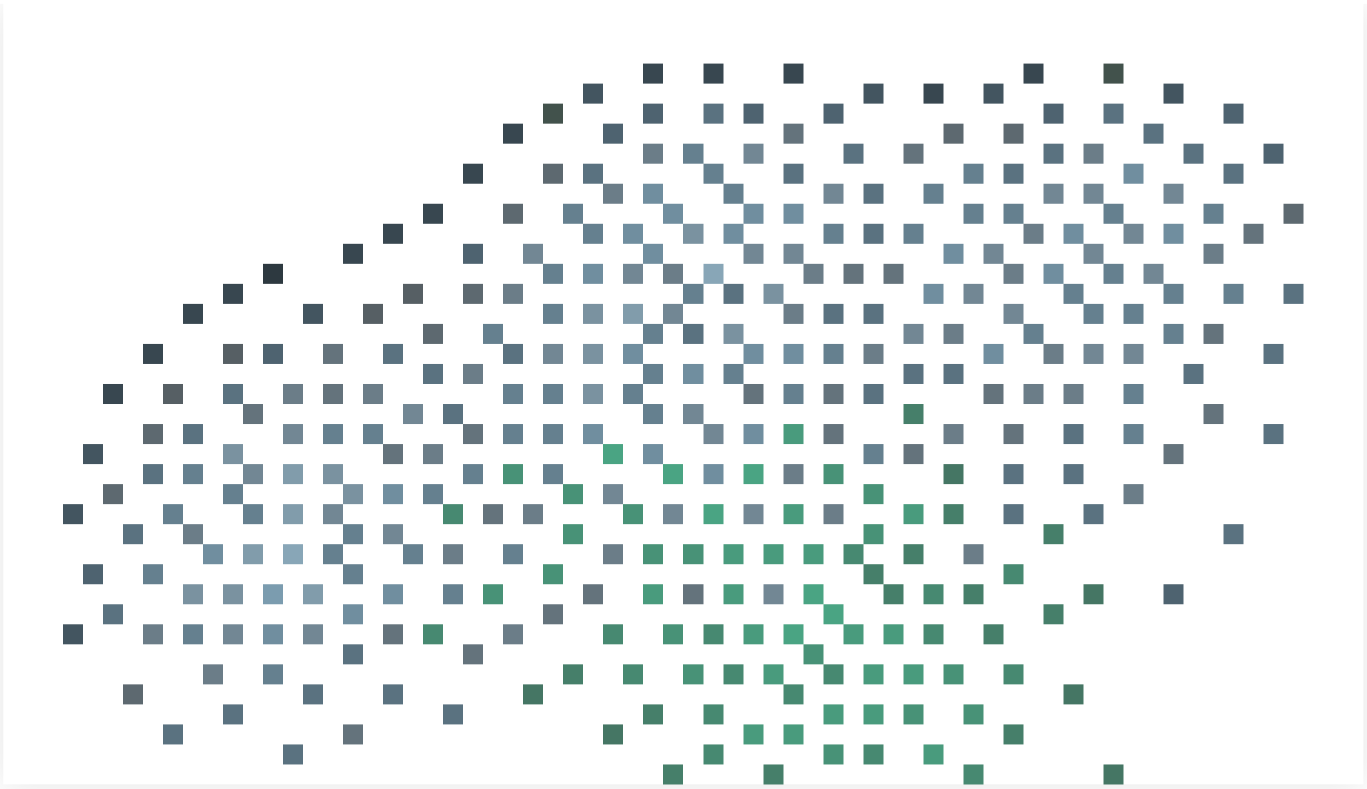
```
git branch -D <branch_name>
```

To make any changes in a specific branch, you have to move inside it first. To do so, use this command.

```
git checkout <branch_name>
```

. . .

## Merge



Git merge

Let’s suppose you have made some changes to a new branch and want to integrate them with your `master` branch. You can do that by performing a merge operation.

```
git merge <branch_name>
```

In simple words, this command is used to combine two branches. If a specific file is changed in both branches then it will create a merge conflict. In that case, you have to manually look into the file and select the change you want and remove the other one.

. . .

## Conclusion

You just saw how easy it is to start tracking the history of a project using Git. It is a tool that is generally used by programmers and software developers. But, you don’t need to know anything about software development to learn the Git software.

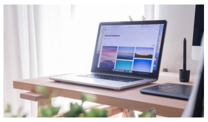
It is packed with commands and has a very simple syntax that anyone can understand.

Thanks for reading!

## You Might Be Interested In

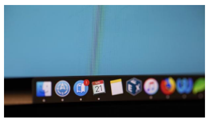
Tools to Improve a Developer’s Workflow

June 14, 2020



My Top 5 Tools for Web Development

May 20, 2020



Top 10 Programming Blogs in 2020

August 20, 2020

Testing in Node.js Using Mocha and Chai [1/2]

August 17, 2020