Git

[](https://www.w3schools.com/git/git_intro.asp?remote=github)

**What is Git?**

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

**It is used for:**

* Tracking code changes
* Tracking who made changes
* Coding collaboration

**What does Git do?**

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

**Working with Git**

* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

**What Does git do???**

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

**What is GitHub?**

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
* In this tutorial, we will focus on using Git with GitHub.

## Git Install

You can download Git for free from the following website: [https://www.git-scm.com/](https://git-scm.com/)

## Using Git with Command Line

To start using Git, we are first going to open up our Command shell.

For Windows, you can use Git bash, which comes included in Git for Windows. For Mac and Linux you can use the built-in terminal.

The first thing we need to do, is to check if Git is properly installed:

git config --global user.name "w3schools-test"

git config --global user.email "test@w3schools.com"

## Creating Git Folder

Now, let's create a new folder for our project:

mkdir myproject

cd myproject

mkdir **make**s a **new directory**.

cd **changes** the **current working directory**.

## Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

git init

Initialized empty Git repository in /Users/user/myproject/.git/

## Git Adding New Files

You just created your first local Git repo. But it is empty.

So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created

ls

index.html

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

git commit -m "First release of Hello World!”

## Working with Git Branches

In Git, a branch is a new/separate version of the main repository.

Let's say you have a large project, and you need to update the design on it.

How would that work without and with Git:

Without Git:

* Make copies of all the relevant files to avoid impacting the live version
* Start working with the design and find that code depend on code in other files, that also need to be changed!
* Make copies of the dependant files as well. Making sure that every file dependency references the correct file name
* EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
* Save all your files, making a note of the names of the copies you were working on
* Work on the unrelated error and update the code to fix it
* Go back to the design, and finish the work there
* Copy the code or rename the files, so the updated design is on the live version
* (2 weeks later, you realize that the unrelated error was not fixed in the new design version because you copied the files before the fix)

With Git:

* With a new branch called new-design, edit the code directly without impacting the main branch
* EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
* Create a new branch from the main project called small-error-fix
* Fix the unrelated error and merge the small-error-fix branch with the main branch
* You go back to the new-design branch, and finish the work there
* Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!

## Merge Branches

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch:

git checkout master

Switched to branch 'master'

## Understanding the GIT Workflow

GIT is the [**most widely used**](https://www.g2.com/categories/version-control-systems) open-source VCS (version control system) that allows you to track changes made to files. Companies and programmers usually use GIT to collaborate on developing software and applications.

A GIT project consists of three major sections: **the working directory**, **the staging area**, and **the git directory**.

The working directory is where you add, delete, and edit the files. Then, the changes are staged (indexed) in the staging area. After you commit your changes, the snapshot of the changes will be saved into the git directory.

Everyone can use GIT as it is available for [**Linux**](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git), [**Windows**](https://gitforwindows.org/), [**Mac**](https://git-scm.com/download/mac), and [**Solaris**](https://www.opencsw.org/packages/git/). The software may have a steep learning curve, but there are lots of [**tutorials**](https://www.hostinger.com/tutorials/git-basics-tutorial) ready to help you.

## Basic GIT Commands

Here are some basic GIT commands you need to know:

* **git init** will create a new local GIT repository. The following Git command will create a repository in the current directory:

git init

* Alternatively, you can create a repository within a new directory by specifying the project name:

git init [project name]

* **git clone** is used to copy a repository. If the repository lies on a remote server, use:

git clone username@host:/path/to/repository

* Conversely, run the following basic command to copy a local repository:

git clone /path/to/repository

* **git add** is used to add files to the staging area. For example, the basic Git following command will index the temp.txt file:

git add <temp.txt>

* **git commit** will create a snapshot of the changes and save it to the git directory.

git commit –m “Message to go with the commit here”

* **git config** can be used to set user-specific configuration values like email, username, file format, and so on. To illustrate, the command for setting up an email will look like this:

git config --global user.email youremail@example.com

* The –global flag tells GIT that you’re going to use that email for all local repositories. If you want to use different emails for different repositories, use the command below:

git config --local user.email youremail@example.com

* **git status** displays the list of changed files together with the files that are yet to be staged or committed.

git status

* **git push** is used to send local commits to the master branch of the remote repository. Here’s the basic code structure:

git push origin <master>

* Replace <master> with the branch where you want to push your changes when you’re not intending to push to the master branch.
* **git checkout** creates branches and helps you to navigate between them. For example, the following basic command creates a new branch and automatically switches you to it:

command git checkout -b <branch-name>

* To switch from one branch to another, simply use:

git checkout <branch-name>

* **git remote** lets you view all remote repositories. The following command will list all connections along with their URLs:

git remote –v

* To connect the local repository to a remote server, use the command below:

git remote add origin <host-or-remoteURL>

* Meanwhile, the following command will delete a connection to a specified remote repository:

git remote rm <name-of-the-repository>

* **git branch** will list, create, or delete branches. For instance, if you want to list all the branches present in the repository, the command should look like this:

git branch

* If you want to delete a branch, use:

git branch –d <branch-name>

* **git pull** merges all the changes present in the remote repository to the local working directory.

git pull

* **git merge** is used to merge a branch into the active one.

git merge <branch-name>

* **git diff** lists down conflicts. In order to view conflicts against the base file, use

git diff --base <file-name>

* The following basic command is used to view the conflicts between branches before merging them:

git diff <source-branch> <target-branch>

* To list down all the present conflicts, use:

git diff

* **git tag** marks specific commits.  Developers usually use it to mark release points like v1.0 and v2.0.

git tag <insert-commitID-here>

* **git log** is used to see the repository’s history by listing certain commit’s details. Running the command will get you an output that looks like this:

commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw

Author: Alex Hunter <alexh@gmail.com>

Date:   Mon Oct 1 12:56:29 2016 -0600

* **git reset** command will reset the index and the working directory to the last git commit’s state.

git reset --hard HEAD

* **git rm** can be used to remove files from the index and the working directory.

git rm filename.txt

* **git stash** command will temporarily save the changes that are not ready to be committed. That way, you can go back to that project later on.

git stash

* **git show** is a command  used to view information about any git object.

git show

* **git fetch** allows users to fetch all objects from the remote repository that don’t currently reside in the local working directory.

git fetch origin

* **git ls-tree** allows you to view a tree object along with the name, the mode of each item, and the blob’s SHA-1 value. Let’s say you want to see the HEAD, use:

git ls-tree HEAD

* **git cat-file** is used to view the type and the size information of a repository object. Use the -p option along with the object’s SHA-1 value to view the information of a specific object, for example:

git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

* **git grep** lets users search through committed trees, working directory, and staging area for specific phrases and words. To search for www.hostinger.com in all files, use:

git grep "www.hostinger.com"

* **gitk** shows the graphical interface for a local repository. Simply run:

gitk

* **git instaweb** allows you to browse your local repository in the git-web interface. For instance:

git instaweb –httpd=webrick

* **git gc**will clean unnecessary files and optimize the local repository.

git gc

* **git archive** lets users create a zip or a tar file containing the constituents of a single repository tree. For instance:

git archive --format=tar master

* **git prune** deletes objects that don’t have any incoming pointers.

git prune

* **git fsck**performs an integrity check of the git file system and identifies any corrupted objects.

git fsck

* **git rebase** is used to apply certain changes from one branch to another. For instance:

git rebase master

**References:**

* [**https://www.hostinger.com/tutorials/basic-git-commands**](https://www.hostinger.com/tutorials/basic-git-commands)
* [**https://www.w3schools.com/git/git\_branch\_pull\_from\_remote.asp?remote=github**](https://www.w3schools.com/git/git_branch_pull_from_remote.asp?remote=github)
* [**https://www.geeksforgeeks.org/basic-git-commands-with-examples/**](https://www.geeksforgeeks.org/basic-git-commands-with-examples/)