Git and 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!

### Why Git?

* 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/)

## Configure Git

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

Example:-

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

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

Change the user name and e-mail address to your own. You will probably also want to use this when registering to GitHub later on.

## Initialize Git

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

Example:-

$ git init

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

You just created your first Git Repository!

## Git Staging Environment

One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on.

Example:-

$ git add index.html

The file should be **Staged**. Let's check the status::

Example:-

$ git status

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached ..." to unstage)

    new file: index.html

## Git Add More than One File

You can also stage more than one file at a time.

Now add all files in the current directory to the Staging Environment:

Example:-

$ git add --all

Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

Example:-

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: README.md

new file: bluestyle.css

new file: index.html

Now all 3 files are added to the Staging Environment.

## Git Commit

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**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Example:-

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

[master (root-commit) 221ec6e] First release of Hello World!

3 files changed, 26 insertions(+)

create mode 100644 README.md

create mode 100644 bluestyle.css

create mode 100644 index.html

The commit command performs a commit, and the -m "message" adds a message.

The Staging Environment has been committed to our repo, with the message:  
"First release of Hello World!"

## Git Commit without Stage

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

Let's add a small update to index.html. We see the file we expected is modified. So let's commit it directly:

Example:-

$ git commit -a -m "Updated index.html with a new line"

[master 09f4acd] Updated index.html with a new line

1 file changed, 1 insertion(+)

## Git Commit Log

To view the history of commits for a repository, you can use the log command:

Example:-

git log

commit 09f4acd3f8836b7f6fc44ad9e012f82faf861803 (HEAD -> master)

Author: w3schools-test

Date: Fri Mar 26 09:35:54 2021 +0100

Updated index.html with a new line

commit 221ec6e10aeedbfd02b85264087cd9adc18e4b26

Author: w3schools-test

Date: Fri Mar 26 09:13:07 2021 +0100

First release of Hello World!

Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

* git *command* -help -  See all the available options for the specific command
* git help --all -  See all possible commands

## Working with Git Branches

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

## New Git Branch

Let add some new features to our index.html page.

We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch.

Example:-

$ git branch hello-world-images

Now we created a new branch called "hello-world-images"

Let's confirm that we have created a new branch:

Example:-

$ git branch

hello-world-images

\* master

We can see the new branch with the name "hello-world-images", but the \* beside master specifies that we are currently on that branch.

checkout is the command used to check out a branch. Moving us **from** the current branch, **to** the one specified at the end of the command:

Example:-

$ git checkout hello-world-images

Switched to branch 'hello-world-images'

Now we have moved our current workspace from the master branch, to the new branch.

## 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:

Example:-

$ git checkout master

Switched to branch 'master'

Now we merge the current branch (master) with emergency-fix:

Example:-

$ git merge emergency-fix

Updating 09f4acd..dfa79db

Fast-forward

index.html | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

Since the emergency-fix branch came directly from master, and no other changes had been made to master while we were working, Git sees this as a continuation of master. So it can "Fast-forward", just pointing both master and emergency-fix to the same commit.

As master and emergency-fix are essentially the same now, we can delete emergency-fix, as it is no longer needed:

Example:-

$ git branch -d emergency-fix

Deleted branch emergency-fix (was dfa79db).

## Merge Conflict

index.html has been changed in both branches. Now we are ready to merge hello-world-images into master. But what will happen to the changes we recently made in master?

Example:-

$ git checkout master

$ git merge hello-world-images

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

The merge failed, as there is conflict between the versions for index.html. Let us check the status:

Example:-

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Changes to be committed:

new file: img\_hello\_git.jpg

new file: img\_hello\_world.jpg

Unmerged paths:

(use "git add ..." to mark resolution)

both modified: index.html

This confirms there is a conflict in index.html, but the image files are ready and staged to be committed.

So we need to fix that conflict. Open the file in our editor:

Example:-

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:960px"></div>  
<p>This is the first file in my new Git Repo.</p>  
<<<<<<< HEAD  
<p>This line is here to show how merging works.</p>  
=======  
<p>A new line in our file!</p>  
<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>  
>>>>>>> hello-world-images  
  
</body>  
</html>

We can see the differences between the versions and edit it like we want. Now we can stage index.html and check the status:

Example:-

$ git add index.html

$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

new file: img\_hello\_git.jpg

new file: img\_hello\_world.jpg

modified: index.html

The conflict has been fixed, and we can use commit to conclude the merge:

Example:-

$ git commit -m "merged with hello-world-images after fixing conflicts"

[master e0b6038] merged with hello-world-images after fixing conflicts

And delete the hello-world-images branch.

# GitHub

* Go to [GitHub](https://www.github.com/) and sign up for an account.
* Now that you have made a GitHub account, sign in, and create a new Repo.
* Since we have already set up a local Git repo, we are going to push that to GitHub.

### Example:-

### $ git remote add origin <https://github.com/test-git/hello-world.git>

git remote add origin URL specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.

Now we are going to push our master branch to the origin url, and set it as the default remote branch:

Example:-

$ git push --set-upstream origin master

Enumerating objects: 22, done.

Counting objects: 100% (22/22), done.

Delta compression using up to 16 threads

Compressing objects: 100% (22/22), done.

Writing objects: 100% (22/22), 92.96 KiB | 23.24 MiB/s, done.

Total 22 (delta 11), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (11/11), done.

To https://github.com/w3schools-test/hello-world.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

## Pulling to Keep up-to-date with Changes

When working as a team on a project, it is important that everyone stays up to date.

Any time you start working on a project, you should get the most recent changes to your local copy.

With Git, you can do that with pull.

pull is a combination of 2 different commands:

* fetch
* merge

Let's take a closer look into how fetch, merge, and pull works.

## Git Fetch

fetch gets all the change history of a tracked branch/repo.

So, on your local Git, fetch updates to see what has changed on GitHub:

Example:-

$ git fetch origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 733 bytes | 3.00 KiB/s, done.

From https://github.com/w3schools-test/hello-world

e0b6038..d29d69f master -> origin/master

## Git Merge

merge combines the current branch, with a specified branch.

We have confirmed that the updates are as expected, and we can merge our current branch (master) with origin/master:

Example;-

$ git merge origin/master

Updating e0b6038..d29d69f

Fast-forward

README.md | 4 +++-

1 file changed, 3 insertions(+), 1 deletion(-)

## Git Pull

But what if you just want to update your local repository, without going through all those steps?

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

Use pull to update our local Git:

Example:-

$ git pull origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 794 bytes | 1024 bytes/s, done.

From https://github.com/w3schools-test/hello-world

a7cdd4b..ab6b4ed master -> origin/master

Updating a7cdd4b..ab6b4ed

Fast-forward

README.md | 2 ++

1 file changed, 2 insertions(+)

## Push Changes to GitHub

Let's try making some changes to our local git and pushing them to GitHub.

Now push our changes to our remote origin:

Example:-

$ git push origin

Enumerating objects: 9, done.

Counting objects: 100% (8/8), done.

Delta compression using up to 16 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (5/5), 578 bytes | 578.00 KiB/s, done.

Total 5 (delta 3), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (3/3), completed with 3 local objects.

To https://github.com/w3schools-test/hello-world.git

5a04b6f..facaeae master -> master

# Git GitHub Fork

## Add to Someone Else's Repository

At the heart of Git is collaboration. However, Git does not allow you to add code to someone else's repository without access rights.

In these next 3 chapters we will show you how to copy a repository, make changes to it, and suggest those changes be implemented to the original repository.

At the end of these chapters, you will have the opportunity to add a message to our public GitHub page: https://w3schools-test.github.io/

## Fork a Repository

A fork is a copy of a repository. This is useful when you want to contribute to someone else's project or start your own project based on theirs.

fork is not a command in Git, but something offered in GitHub and other repository hosts. Let's start by logging in to GitHub, and fork our repository:  
https://github.com/w3schools-test/w3schools-test.github.io

## Clone a Fork from GitHub

Now we have our own fork, but only on GitHub. We also want a clone on our local Git to keep working on it.

A clone is a full copy of a repository, including all logging and versions of files.

Move back to the **original** repository, and click the green "Code" button to get the URL to clone:

Open your Git bash and clone the repository:

Example:-

$ git clone https://github.com/w3schools-test/w3schools-test.github.io.git

Cloning into 'w3schools-test.github.io'...

remote: Enumerating objects: 33, done.

remote: Counting objects: 100% (33/33), done.

remote: Compressing objects: 100% (15/15), done.

remote: Total 33 (delta 18), reused 33 (delta 18), pack-reused 0

Receiving objects: 100% (33/33), 94.79 KiB | 3.16 MiB/s, done.

Resolving deltas: 100% (18/18), done.

Take a look in your file system, and you will see a new directory named after the cloned project:

Example:-

$ ls

w3schools-test.github.io/

Navigate to the new directory, and check the status:

Example:-

$ cd w3schools-test.github.io

$ git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

And check the log to confirm that we have the full repository data:

Example:-

$ git log

commit facaeae8fd87dcb63629f108f401aa9c3614d4e6 (HEAD -> master, origin/master, origin/HEAD)

Merge: e7de78f 5a04b6f

Author: w3schools-test

Date: Fri Mar 26 15:44:10 2021 +0100

Merge branch 'master' of https://github.com/w3schools-test/hello-world

commit e7de78fdefdda51f6f961829fcbdf197e9b926b6

Author: w3schools-test

Date: Fri Mar 26 15:37:22 2021 +0100

Updated index.html. Resized image

.....

Now we have a full copy of the original repository.

## Configuring Remotes

Basically, we have a full copy of a repository, whose origin we are not allowed to make changes to.

Let's see how the remotes of this Git is set up:

Example:-

$ git remote -v

origin https://github.com/w3schools-test/w3schools-test.github.io.git (fetch)

origin https://github.com/w3schools-test/w3schools-test.github.io.git (push)

We see that origin is set up to the original "w3schools-test" repository, we also want to add our own fork.

First, we rename the original origin remote:

Example:-

$ git remote rename origin upstream

$ git remote -v

upstream https://github.com/w3schools-test/w3schools-test.github.io.git (fetch)

upstream https://github.com/w3schools-test/w3schools-test.github.io.git (push)

Then fetch the URL of our own fork, and add that as origin:

Example:-

$ git remote add origin https://github.com/kaijim/w3schools-test.github.io.git

$ git remote -v

origin https://github.com/kaijim/w3schools-test.github.io.git (fetch)

origin https://github.com/kaijim/w3schools-test.github.io.git (push)

upstream https://github.com/w3schools-test/w3schools-test.github.io.git (fetch)

upstream https://github.com/w3schools-test/w3schools-test.github.io.git (push)

**Note:** According to Git naming conventions, it is recommended to name your own repository origin, and the one you forked for upstream

Now we have 2 remotes:

* origin - our own fork, where we have read and write access
* upstream - the original, where we have read-only access