**Git - Tutorial**

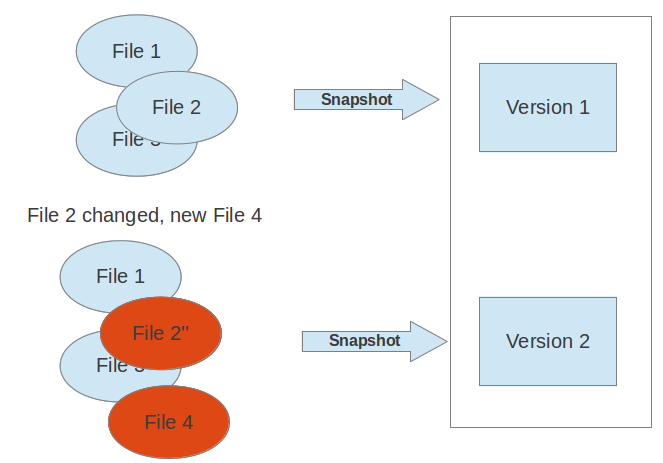
**What is a Versioning Control System?**

A version control system (VCS) allows you to track the history of a collection of files.

It supports creating different versions of this collection.

Each version captures a snapshot of the files at a certain point in time and the VCS allows you to switch between these versions.

These versions are stored in a specific place, typically called a repository.



**Localized and Centralized version control system:**

A localized version control system keeps local copies of the files. This approach can be as simple as creating a manual copy of the relevant files.

A centralized version control system provides a server software component which stores and manages the different versions of the files. A developer can copy (checkout) a certain version from the central sever onto their individual computer.

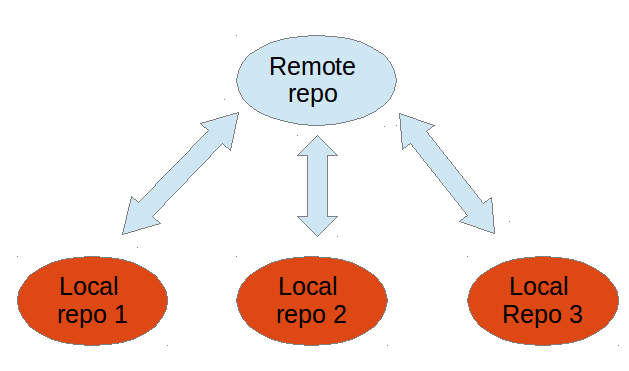
Both approaches have the drawback that they have one single point of failure. In a localized version control systems it is the individual computer and in a centralized version control systems it is the server machine. Both system makes it also harder to work in parallel on different features.

**Distributed Version Control System :**

In a distributed version control system each user has a complete local copy of a repository on his individual computer. The user can copy an existing repository. This copying process is typically called cloning and the resulting repository can be referred to as a clone.

Every clone contains the full history of the collection of files and a cloned repository has the same functionality as the original repository.

Every repository can exchange versions of the files with other repositories by transporting these changes. This is typically done via a repository running on a server which is, unlike the local machine of a developer, always online. Typically, there is a central server for keeping a repository but each cloned repository is a full copy of this repository. The decision which of the copies is considered to be the central server repository is pure convention.



**What is Git?**

Git is currently the most popular implementation of a distributed version control system.

Git originates from the Linux kernel development and was founded in 2005 by Linus Torvalds. Nowadays it is used by many popular open source projects, e.g., the Android or the Eclipse developer teams, as well as many commercial organizations.

**Git repository**

A Git repository contains the history of a collection of files starting from a certain directory. The process of copying an existing Git repository via the Git tooling is called cloning. After cloning a repository the user has the complete repository with its history on his local machine. Of course, Git also supports the creation of new repositories.

If you want to delete a Git repository, you can simply delete the folder which contains the repository.

If you clone a Git repository, by default, Git assumes that you want to work in this repository as a user. Git also supports the creation of repositories targeting the usage on a server.

**Working Tree**

A local non-bare Git repository is typically called local repository. A local repository provides at least one collection of files which originate from a certain version of the repository. This collection of files is called the working tree. It corresponds to a checkout of one version of the repository with potential changes done by the user. The user can change the files in the working tree by modifying existing files and by creating and removing files. A file in the working tree of a Git repository can have different states. These states are the following:

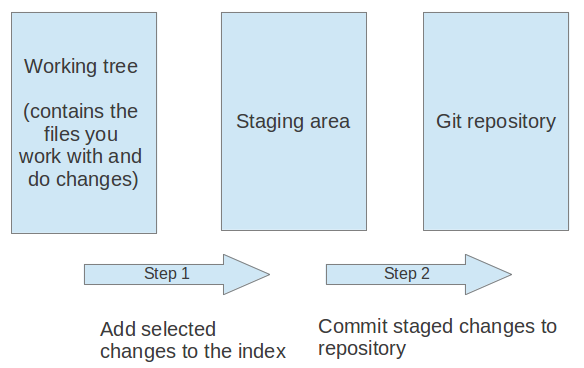
* untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed.
* tracked: committed and not staged
* staged: staged to be included in the next commit
* dirty / modified: the file has changed but the change is not staged

After doing changes in the working tree, the user can add these changes to the Git repository or revert these changes.

**After modifying your *working tree* you need to perform the following two steps to persist these changes in your local repository:**

* add the selected changes to the *staging area* (also known as index) via the git add command
* commit the staged changes into the Git repository via the git commit command

This process is depicted in the following graphic.



The git add command stores a snapshot of the specified files in the staging area. It allows you to incrementally modify files, stage them, modify and stage them again until you are satisfied with your changes.

After adding the selected files to the staging area, you can commit these files to add them permanently to the Git repository. Committing creates a new persistent snapshot (called commit or commit object) of the staging area in the Git repository. A commit object, like all objects in Git, is immutable.

The staging area keeps track of the snapshots of the files until the staged changes are committed.

For committing the staged changes you use the git commit command. If you commit changes to your Git repository, you create a new commit object in the Git repository.

**Branching**

Git supports branching which means that you can work on different versions of your collection of files. A branch allows the user to switch between these versions so that he can work on different changes independently from each other.

For example, if you want to develop a new feature, you can create a branch and make the changes in this branch. This does not affect the state of your files in other branches.

Branches in Git are local to the repository. A branch created in a local repository does not need to have a counterpart in a remote repository. Local branches can be compared with other local branches and with remote-tracking branches. A remote-tracking branch proxies the state of a branch in another remote repository.

**Git terminology**

Branch

A branch is a named pointer to a commit. Selecting a branch in Git terminology is called to checkout a branch. If you are working in a certain branch, the creation of a new commit advances this pointer to the newly created commit.

Commit

When you commit your changes into a repository this creates a new commit object in the Git repository. This commit object uniquely identifies a new revision of the content of the repository.

HEAD

HEAD is a symbolic reference most often pointing to the currently checked out branch.

Index

Index is an alternative term for the staging area.

Repository

A repository contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository.

Revision

Represents a version of the source code. Git implements revisions as commit objects (or short commits ).

Staging area

The staging area is the place to store changes in the working tree before the commit. The staging area contains a snapshot of the changes in the working tree (changed or new files) relevant to create the next commit and stores their mode (file type, executable bit).

Tag

A tag points to a commit which uniquely identifies a version of the Git repository.

URL

A URL in Git determines the location of the repository.

Working tree

The working tree contains the set of working files for the repository. You can modify the content and commit the changes as new commits to the repository.

### Basic Git commands

git init

initializes a brand new Git repository and begins tracking an existing directory.

It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

git clone

creates a local copy of a project that already exists remotely.

The clone includes all the project’s files, history, and branches.

git add

stages a change. Git tracks changes to a developer’s codebase, but it’s necessary to stage and take a snapshot of the changes to include them in the project’s history.

git commit

saves the snapshot to the project history and completes the change-tracking process.

In short, a commit functions like taking a photo. Anything that’s been staged with git add will become a part of the snapshot with git commit.

git status

shows the status of changes as untracked, modified, or staged.

git branch

shows the branches being worked on locally.

git merge

merges lines of development together. This command is typically used to combine changes made on two distinct branches.

For example, a developer would merge when they want to combine changes from a feature branch into the master branch for deployment.

git pull

updates the local line of development with updates from its remote counterpart.

Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

git push

updates the remote repository with any commits made locally to a branch.

## What is GitHub?

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

you need a github account

<https://github.com/>

### How GitHub works

GitHub builds collaboration directly into the development process. Work is organized into repositories, where developers can outline requirements or direction and set expectations for team members. Then, using the GitHub flow, developers simply create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.

### GitHub flow

GitHub flow is a lightweight, branch-based workflow built around core Git commands used by teams around the globe—including ours.

The GitHub flow has six steps, each with distinct benefits when implemented:

1. **Create a branch:** Topic branches created from the canonical deployment branch (usually master) allow teams to contribute to many parallel efforts. Short-lived topic branches, in particular, keep teams focused and results in quick ships.
2. **Add commits:** Snapshots of development efforts within a branch create safe, revertible points in the project’s history.
3. **Open a pull request:** Pull requests publicize a project’s ongoing efforts and set the tone for a transparent development process.
4. **Discuss and review code:** Teams participate in code reviews by commenting, testing, and reviewing open pull requests. Code review is at the core of an open and participatory culture.
5. **Merge:** Upon clicking merge, GitHub automatically performs the equivalent of a local ‘git merge’ operation. GitHub also keeps the entire branch development history on the merged pull request.
6. **Deploy:** Teams can choose the best release cycles or incorporate continuous integration tools and operate with the assurance that code on the deployment branch has gone through a robust workflow.

#### Example: Contribute to an existing repository

# download a repository on GitHub.com to our machine

git clone https://github.com/me/repo.git

# change into the `repo` directory

cd repo

# create a new branch to store any new changes

git branch my-branch

# switch to that branch (line of development)

git checkout my-branch

# make changes, for example, edit `file1.md` and `file2.md` using the text editor

# stage the changed files

git add file1.md file2.md

# take a snapshot of the staging area (anything that's been added)

git commit -m "my snapshot"

# push changes to github

git push --set-upstream origin my-branch

#### Example: Start a new repository and publish it to GitHub

# create a new directory, and initialize it with git-specific functions

git init my-repo

# change into the `my-repo` directory

cd my-repo

# create the first file in the project

touch README.md

# git isn't aware of the file, stage it

git add README.md

# take a snapshot of the staging area

git commit -m "add README to initial commit"

# provide the path for the repository you created on github

git remote add origin https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

# push changes to github

git push --set-upstream origin master

#### Example: contribute to an existing branch on GitHub

# assumption: a project called `repo` already exists on the machine, and a new branch has been pushed to GitHub.com since the last time changes were made locally

# change into the `repo` directory

cd repo

# update all remote tracking branches, and the currently checked out branch

git pull

# change into the existing branch called `feature-a`

git checkout feature-a

# make changes, for example, edit `file1.md` using the text editor

# stage the changed file

git add file1.md

# take a snapshot of the staging area

git commit -m "edit file1"

# push changes to github

git push

## Step 1. Create a Repository

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a README, or a file with information about your project.

### To create a new repository

1. In the upper right corner, next to your avatar or identicon, click  and then select **New repository**.
2. Name your repository hello-world.
3. Write a short description.
4. Select **Initialize this repository with a README**.



Click **Create repository**.

## Step 2. Create a Branch

**Branching** is the way to work on different versions of a repository at one time.

By default your repository has one branch named master which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to master.

When you create a branch off the master branch, you’re making a copy, or snapshot, of master as it was at that point in time. If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

This diagram shows:

* The master branch
* A new branch called feature (because we’re doing ‘feature work’ on this branch)
* The journey that feature takes before it’s merged into master



### To create a new branch

1. Go to your new repository hello-world.
2. Click the drop down at the top of the file list that says **branch: master**.
3. Type a branch name, readme-edits, into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.



Now you have two branches, master and readme-edits.

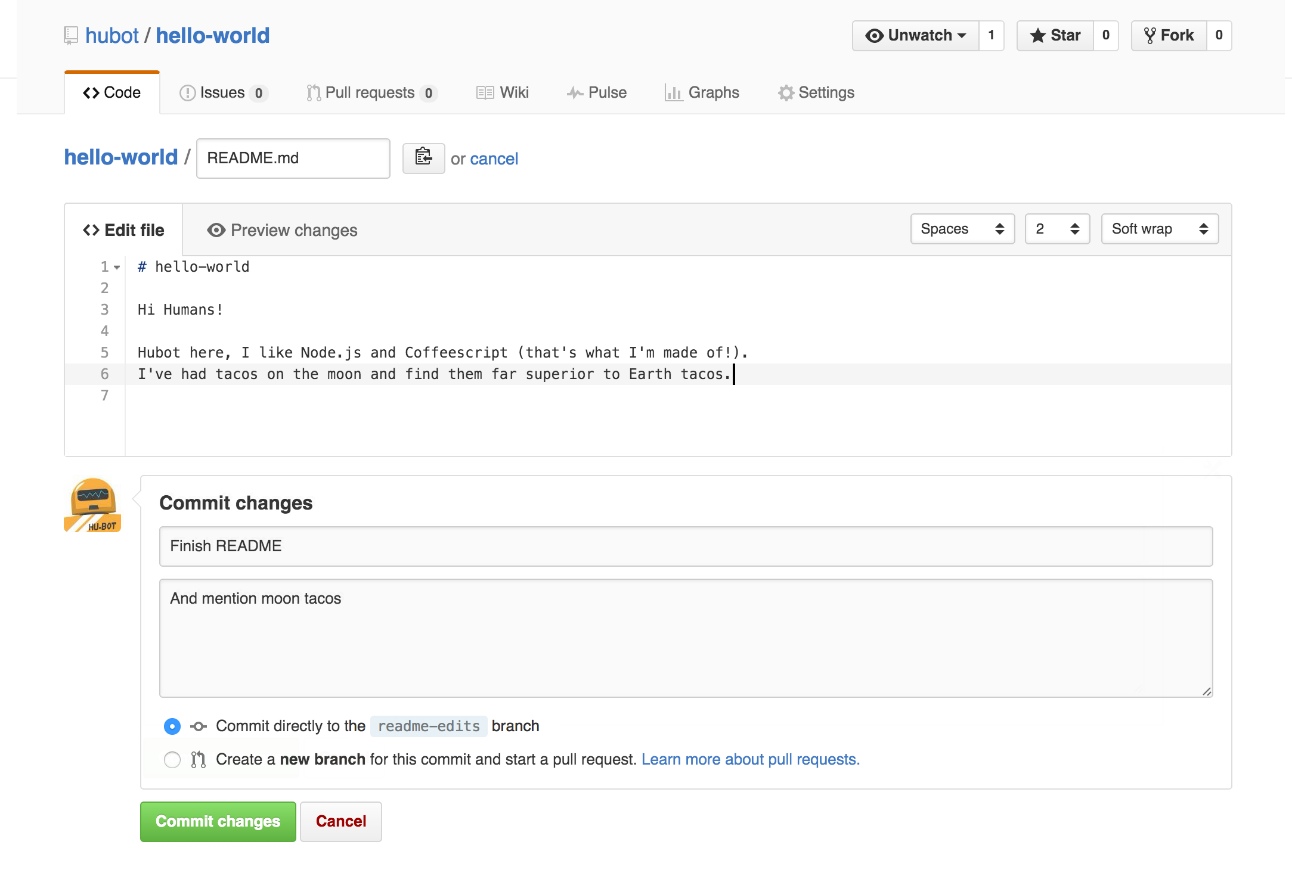
## Step 3. Make and commit changes

Bravo! Now, you’re on the code view for your readme-edits branch, which is a copy of master. Let’s make some edits.

On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you’ve done and why.

#### Make and commit changes

1. Click the README.md file.
2. Click the  pencil icon in the upper right corner of the file view to edit.
3. In the editor, write a bit about yourself.
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.



These changes will be made to just the README file on your readme-editsbranch.

## Step 4. Open a Pull Request

Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

#### Open a Pull Request for changes to the README

**Step**

Click the  **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button.

In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with master (the original).

Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit.

When you’re satisfied that these are the changes you want to submit, click the big green **Create Pull Request**button.

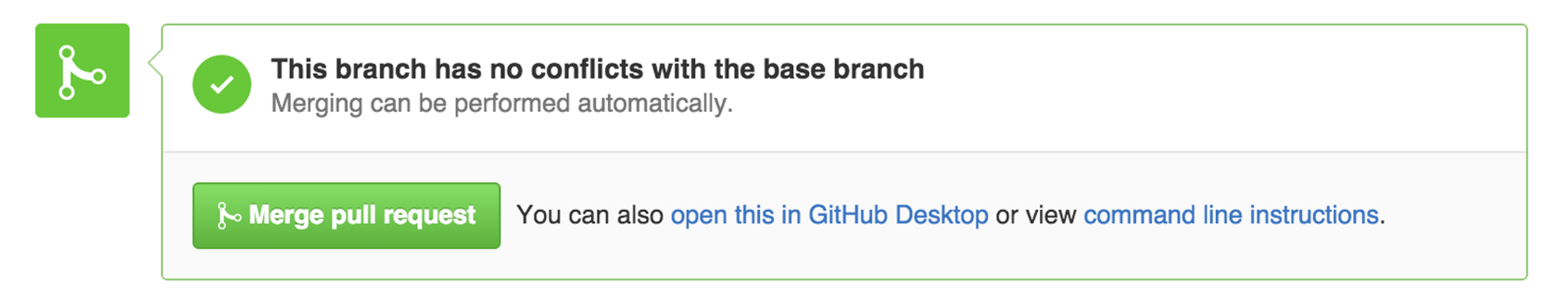
Give your pull request a title and write a brief description of your changes.

When you’re done with your message, click **Create pull request**!

## Step 5. Merge your Pull Request

In this final step, it’s time to bring your changes together – merging your readme-edits branch into the master branch.

1. Click the green **Merge pull request** button to merge the changes into master.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.



GitHub : <https://guides.github.com/activities/hello-world/#repository>