**You’ll learn how to:**

* Create and use a repository
* Start and manage a new branch
* Make changes to a file and push them to GitHub as commits
* Open and merge a pull request

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

This tutorial teaches you GitHub essentials like *repositories*, *branches*, *commits*, and *Pull Requests*. You’ll create your own Hello World repository and learn GitHub’s Pull Request workflow, a popular way to create and review code.

## Step 1. Create a Repository

**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 main which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to main.

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

This diagram shows:

* The main 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 main



Have you ever saved different versions of a file? Something like:

* story.txt
* story-joe-edit.txt
* story-joe-edit-reviewed.txt

Branches accomplish similar goals in GitHub repositories.

Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our main (production) branch. When a change is ready, they merge their branch into main.

### 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: main**.
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, main and readme-edits. They look exactly the same, but not for long! Next we’ll add our changes to the new branch.

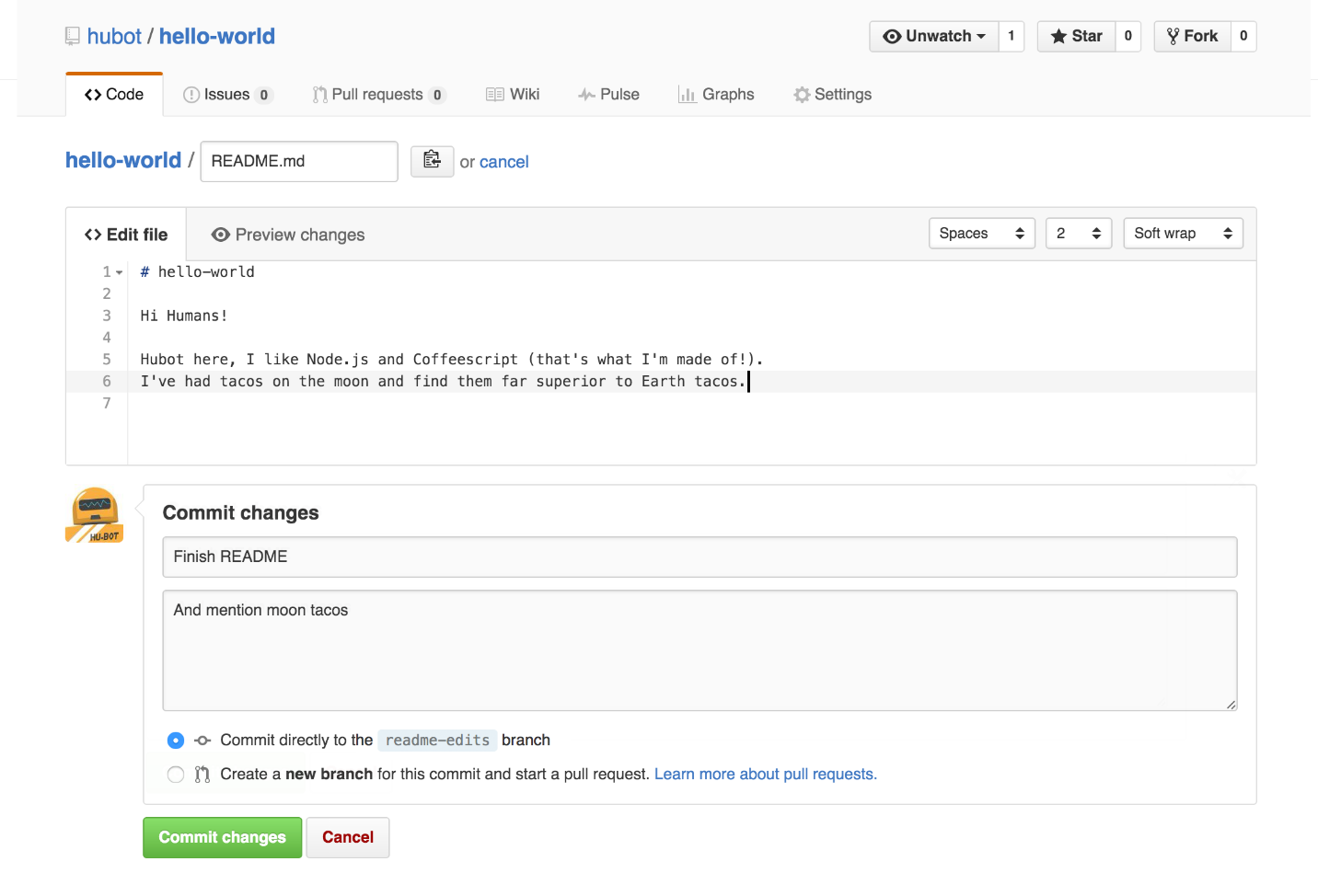
## Step 3. Make and commit changes

Bravo! Now, you’re on the code view for your readme-edits branch, which is a copy of main. 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-edits branch, so now this branch contains content that’s different from main.

## Step 4. Open a Pull Request

Nice edits! Now that you have changes in a branch off of main, you can 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.

By using GitHub’s [@mention system](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.

You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub flow before working on larger projects.

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

Click on the image for a larger version

| **Step** | **Screenshot** |
| --- | --- |
| Click the  **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button. | [pr-tab](https://guides.github.com/activities/hello-world/pr-tab.gif) |
| In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with main (the original). | [branch](https://guides.github.com/activities/hello-world/pick-branch.png) |
| Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit. | [diff](https://guides.github.com/activities/hello-world/diff.png) |
| When you’re satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button. | [create-pull](https://guides.github.com/activities/hello-world/create-pr.png) |
| Give your pull request a title and write a brief description of your changes. | [pr-form](https://guides.github.com/activities/hello-world/pr-form.png) |

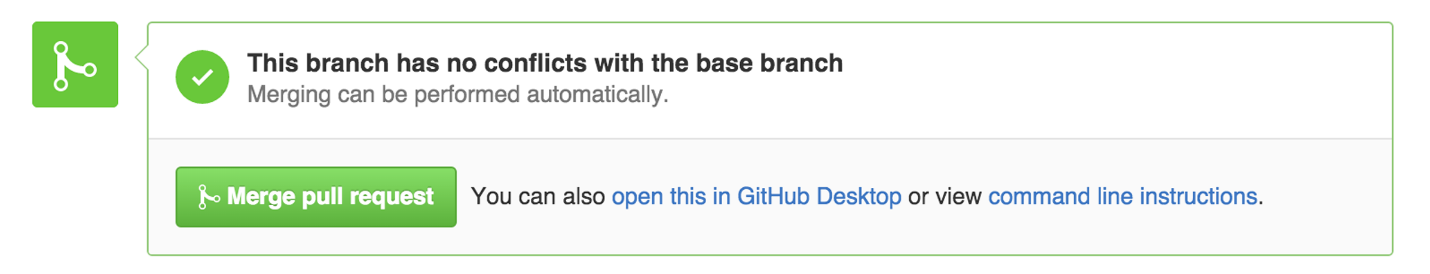
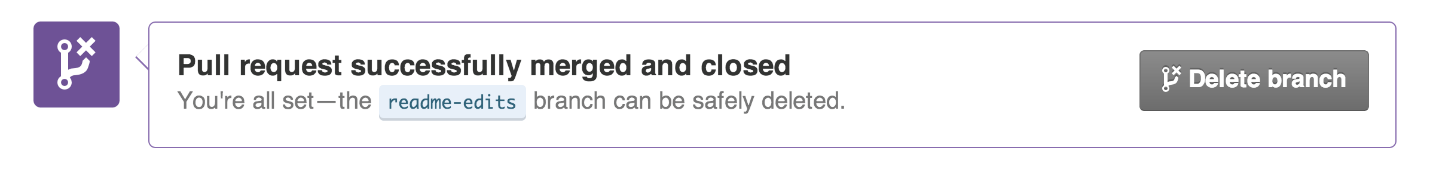
When you’re done with your message, click **Create pull request**!

**Tip**: You can use [emoji](https://help.github.com/articles/basic-writing-and-formatting-syntax/#using-emoji) and [drag and drop images and gifs](https://help.github.com/articles/file-attachments-on-issues-and-pull-requests/) onto comments and Pull Requests.

## 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 main branch.

1. Click the green **Merge pull request** button to merge the changes into main.
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.

### Celebrate!

By completing this tutorial, you’ve learned to create a project and make a pull request on GitHub!

Here’s what you accomplished in this tutorial:

* Created an open source repository
* Started and managed a new branch
* Changed a file and committed those changes to GitHub
* Opened and merged a Pull Request

Take a look at your GitHub profile and you’ll see your new [contribution squares](https://help.github.com/articles/viewing-contributions)!

To learn more about the power of Pull Requests, we recommend reading the [GitHub flow Guide](http://guides.github.com/overviews/flow/). You might also visit [GitHub Explore](http://github.com/explore) and get involved in an Open Source project.

## **GitBash Commands:**

Below link is quite useful:

<https://www.datacamp.com/community/tutorials/git-push-pull>

Ls

Cd

Git init

Git remote add origin <gitlink>

Git remote -v

Git add

Git commit -m “First Commit”

Git push origin main

Git help

First clone your github repository to local system.

Git clone “link”

Then if you make any change in local:

Git add . or git add file names separated by space

Then

Git commit -m “Comments”

Then finally push the changes from local to online github repository

Git push -u origin main

<https://www.freecodecamp.org/news/10-important-git-commands-that-every-developer-should-know/>

Top 10 git commands:

1. Clone(download existing source code from remote repo to local)
   1. Git clone <https://name-of-the-repository-link>
2. Branch(by using branches several developers can work in parallel on same project simultaneously)

Creating a new local branch,

git branch <branch-name> (usually branch name= main)

to push new branch into remote repository use following:

git push -u <remote> <branch-name> (usually remote= origin, branch-name= main)

to view branches

git branch or git branch –list

deleting a branch

git branch -d <branch-name>

1. Checkout (used for switching from one branch to another, also used for checking out files and commits)

git checkout <name-of-your-branch>

There are some steps you need to follow for successfully switching between branches:

* The changes in your current branch must be committed or stashed before you switch.
* The branch you want to check out should exist in your local.

There is also a shortcut command that allows you to create and switch to a branch at the same time:

git checkout -b <name-of-your-branch>

This command creates a new branch in your local (-b stands for branch) and checks the branch out to new right after it has been created.

1. Status (The Git status command gives us all the necessary information about the current branch).

git status

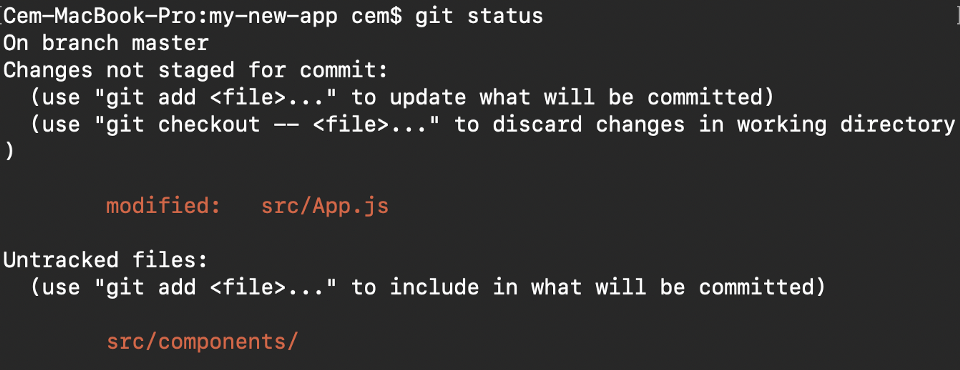
We can gather information like:

Whether the current branch is up to date

Whether there is anything to commit, push or pull

Whether there are files staged, unstaged or untracked

Whether there are files created, modified or deleted



1. Add (When we create, modify or delete a file, these changes will happen in our local and won't be included in the next commit (unless we change the configurations). We need to use the git add command to include the changes of a file(s) into our next commit.

To add a single file:

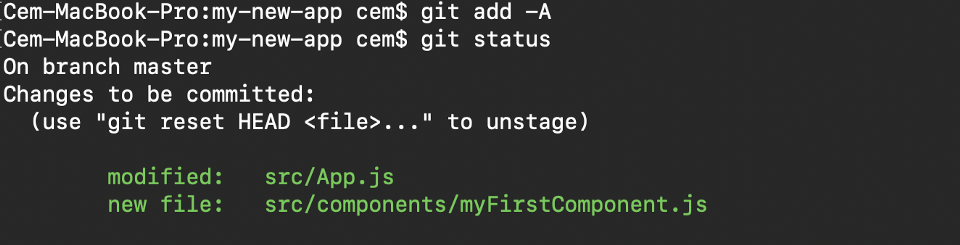
git add <file>

To add everything at once:

git add -A

When you visit the screenshot above in the 4th section, you will see that there are file names that are red - this means that they're unstaged files. The unstaged files won't be included in your commits.

To include them, we need to use git add:



Files with green are now staged with git add

Important: The git add command doesn't change the repository and the changes are not saved until we use git commit.

1. Commit

most-used command of Git. Once we reach a certain point in development, we want to save our changes (maybe after a specific task or issue).

Git commit is like setting a checkpoint in the development process which you can go back to later if needed.

We also need to write a short message to explain what we have developed or changed in the source code.

git commit -m "commit message"

Important: Git commit saves your changes only locally.

1. Push

After committing your changes, the next thing you want to do is send your changes to the remote server. Git push uploads your commits to the remote repository.

git push <remote> <branch-name>

However, if your branch is newly created, then you also need to upload the branch with the following command:

git push --set-upstream <remote> <name-of-your-branch>

or

git push -u origin <branch\_name>

Important: Git push only uploads changes that are committed.

1. Pull

The git pull command is used to get updates from the remote repo. This command is a combination of git fetch and git merge which means that, when we use git pull, it gets the updates from remote repository (git fetch) and immediately applies the latest changes in your local (git merge).

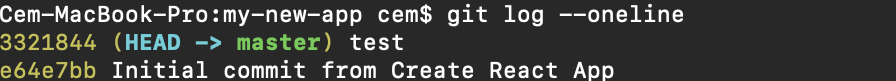
git pull <remote>

This operation may cause conflicts that you need to solve manually.

1. Revert

Sometimes we need to undo the changes that we've made. There are various ways to undo our changes locally or remotely (depends on what we need), but we must carefully use these commands to avoid unwanted deletions.

A safer way that we can undo our commits is by using git revert. To see our commit history, first we need to use git log -- oneline:

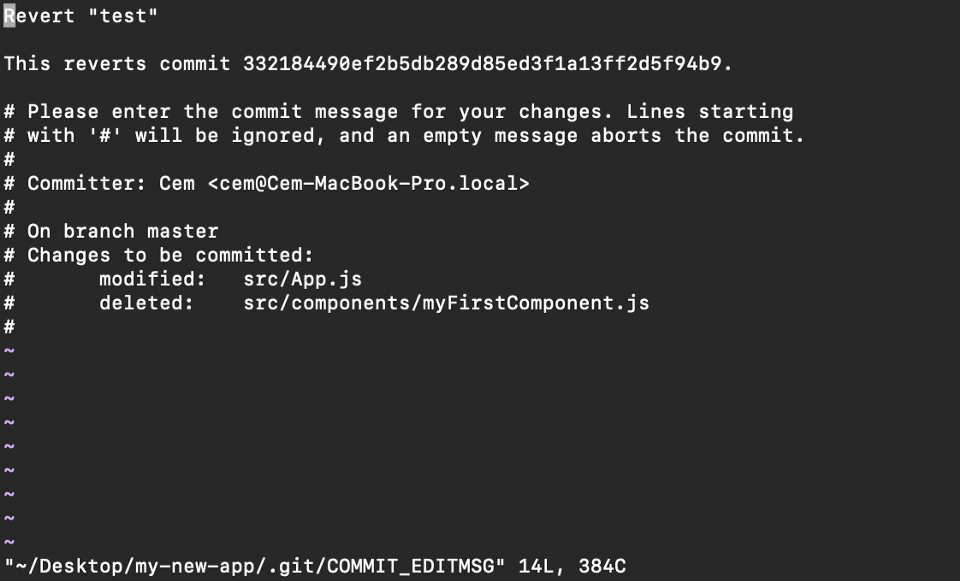


commit history of my main branch

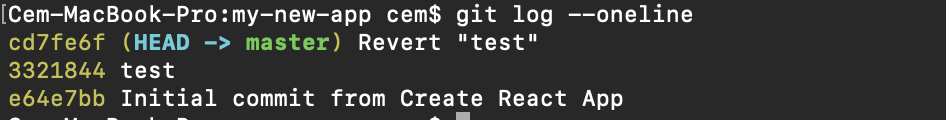
Then we just need to specify the hash code next to our commit that we would like to undo:

git revert 3321844

After this, you will see a screen like below - just press shift + q to exit:



The Git revert command will undo the given commit, but will create a new commit without deleting the older one:



new "revert" commit

The advantage of using git revert is that it doesn't touch the commit history. This means that you can still see all of the commits in your history, even the reverted ones.

Another safety measure here is that everything happens in our local system unless we push them to the remote repo. That's why git revert is safer to use and is the preferred way to undo our commits.

1. Merge

When you've completed development in your branch and everything works fine, the final step is merging the branch with the parent branch (dev or main). This is done with the git merge command.

Git merge basically integrates your feature branch with all of its commits back to the dev (or main) branch. It's important to remember that you first need to be on the specific branch that you want to merge with your feature branch.

For example, when you want to merge your feature branch into the dev branch:

First you should switch to the dev branch:

git checkout dev

Before merging, you should update your local dev branch:

git fetch

Finally, you can merge your feature branch into dev:

git merge <branch-name>

Hint: Make sure your dev branch has the latest version before you merge your branches, otherwise you may face conflicts or other unwanted problems.