Git W3School

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/

## Git Staging Environment

**Staged** files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

For now, we are done working with index.html. So we can add it to the Staging Environment:

### Example

git add index.html

Git Add More than One File

### Example

git add --all

## Git Commit

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!"

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

And check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:

### Example

git status --short

M index.html

**Note:** Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files

**Warning:** Skipping the Staging Environment is not generally recommended.

Skipping the stage step can sometimes make you include unwanted changes.

## Git Commit Log

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

### Example

git log

## 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.

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

* 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)

## New Git Branch

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

So we create a new branch:

### Example

git branch 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'

**Note:** Using the -b option on checkout will create a new branch, and move to it, if it does not exist

## 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

Now we can move over to hello-world-images and keep working. Add another image file (img\_hello\_git.jpg) and change index.html, so it shows it:

Example

git checkout hello-world-images

Switched to branch 'hello-world-images'

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>  
<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>  
  
</body>  
</html>

Now, we are done with our work here and can stage and commit for this branch:

Example

git add --all

git commit -m "added new image"

[hello-world-images 1f1584e] added new image

2 files changed, 1 insertion(+)

create mode 100644 img\_hello\_git.jpg

We see that 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:

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>  
<p>This line is here to show how merging works.</p>  
<div><img src="img\_hello\_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>  
  
</body>  
</html>

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:

Example

git branch -d hello-world-images

Deleted branch hello-world-images (was 1f1584e).

Now you have a better understanding of how branches and merging works. Time to start working with a remote repository!

# **Git GitHub Getting Started**

### Example

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

# What is the origin in Git?

In Git, "origin" is a shorthand name for the remote repository that a project was originally cloned from. More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier.

Note that origin is by no means a "magical" name, but just a standard convention. Although it makes sense to leave this convention untouched, you could perfectly rename it without losing any functionality.

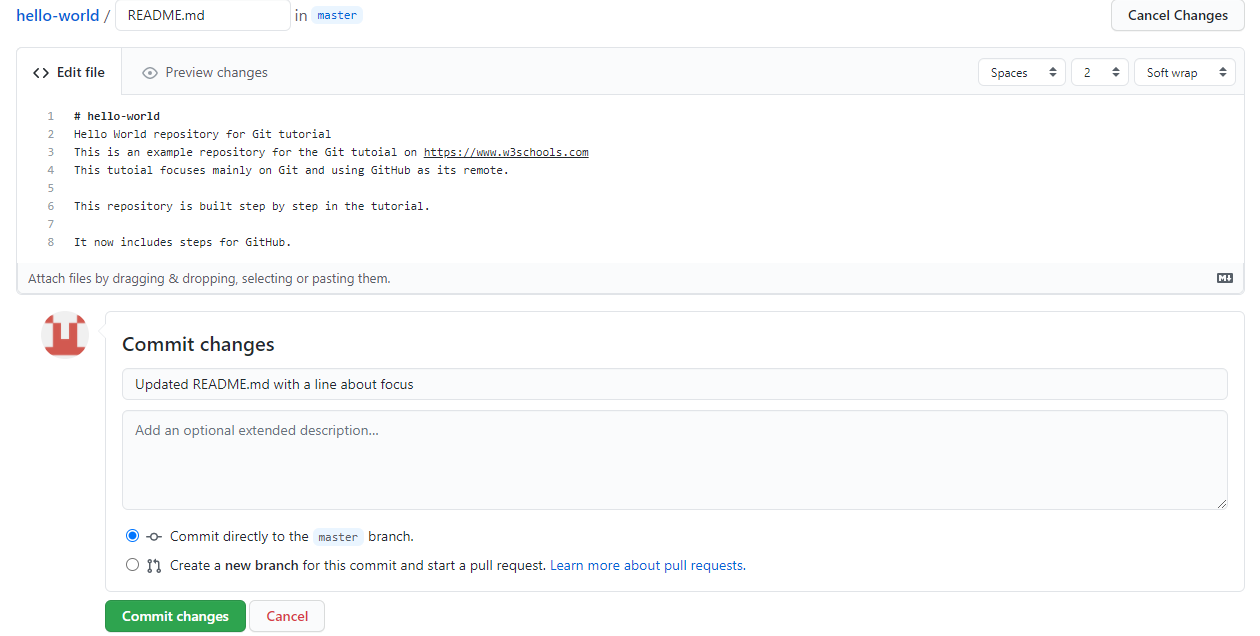
Now we are going to push our master branch to the origin url, and set it as the default remote branch:

### Example

git push --set-upstream origin master

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

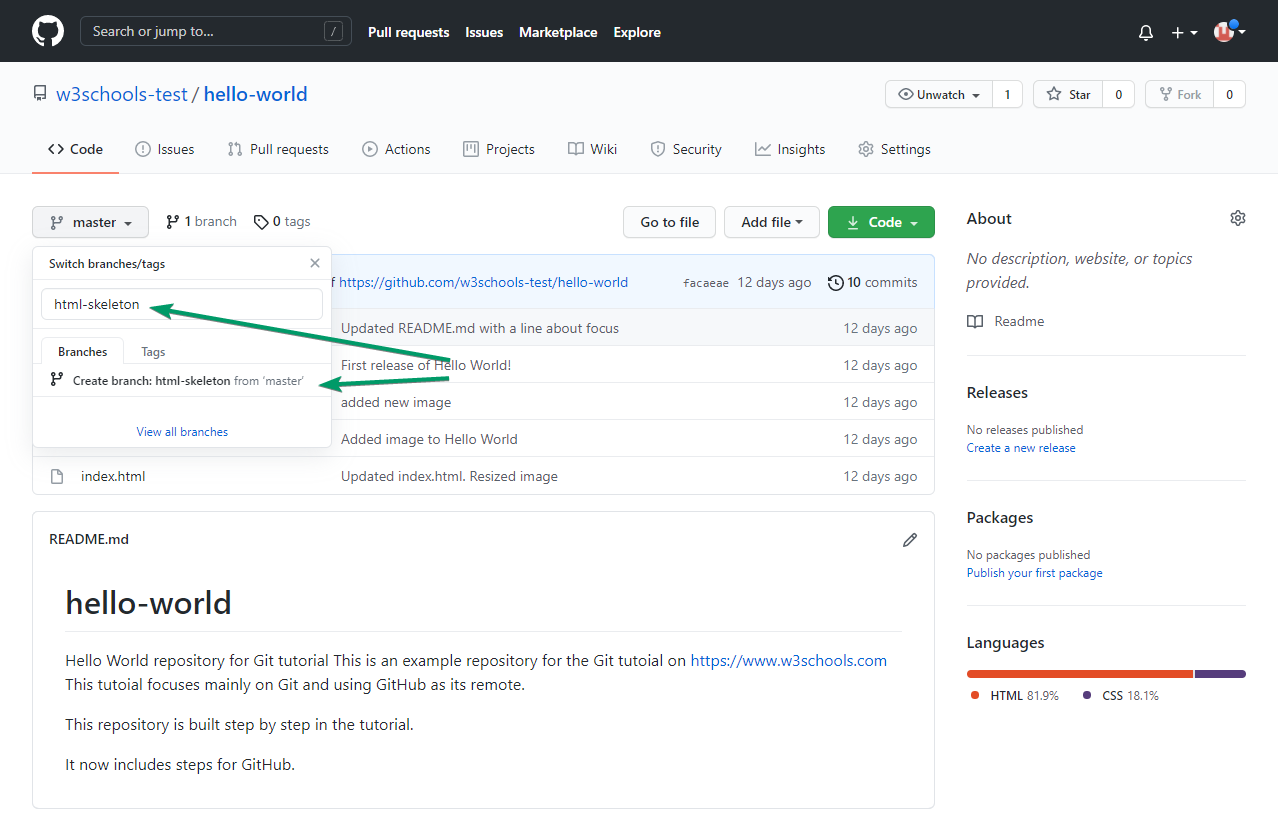
pull is a combination of 2 different commands:

* fetch
* merge
* 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
* Now that we have the recent changes, we can check our status:
* Example
* git status
* On branch master
* Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
* (use "git pull" to update your local branch)
* nothing to commit, working tree clean
* We are behind the origin/master by 1 commit. That should be the updated README.md, but lets double check by viewing the log:
* Example
* git log origin/master
* commit d29d69ffe2ee9e6df6fa0d313bb0592b50f3b853 (origin/master)
* Author: w3schools-test <77673807+w3schools-test@users.noreply.github.com>
* Date: Fri Mar 26 14:59:14 2021 +0100
* Updated README.md with a line about GitHub
* commit e0b6038b1345e50aca8885d8fd322fc0e5765c3b (HEAD -> master)
* Merge: dfa79db 1f1584e
* Author: w3schools-test
* Date: Fri Mar 26 12:42:56 2021 +0100
* merged with hello-world-images after fixing conflicts
* ...
* ...
* That looks as expected, but we can also verify by showing the differences between our local master and origin/master:
* Example
* git diff origin/master
* diff --git a/README.md b/README.md
* index 23a0122..a980c39 100644
* --- a/README.md
* +++ b/README.md
* @@ -2,6 +2,4 @@
* Hello World repository for Git tutorial
* This is an example repository for the Git tutoial on https://www.w3schools.com
* -This repository is built step by step in the tutorial.
* -
* -It now includes steps for GitHub
* +This repository is built step by step in the tutorial.
* \ No newline at end of file
* That looks precisely as expected! Now we can safely merge.
* 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(-)
* Check our status again to confirm we are up to date:
* Example
* git status
* On branch master
* Your branch is up to date with 'origin/master'.
* nothing to commit, working tree clean
* There! Your local git is up to date!
* 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.
* Make another change to the Readme.md file on GitHub.
* 
* Use pull to update our local Git:
* Example
* git pull origin

## Create a New Branch on GitHub

On GitHub, access your repository and click the "master" branch button.

There you can create a new Branch. Type in a descriptive name, and click Create branch:



The branch should now be created and active.

Change Platform:

[[](https://www.w3schools.com/git/git_branch_pull_from_remote.asp?remote=github)GitHub](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=bitbucket)Bitbucket](https://www.w3schools.com/git/git_branch_pull_from_remote.asp?remote=bitbucket)[[](https://www.w3schools.com/git/git_branch_pull_from_remote.asp?remote=gitlab)GitLab](https://www.w3schools.com/git/git_branch_pull_from_remote.asp?remote=gitlab)

Pulling a Branch from GitHub

Now continue working on our new branch in our local Git.

Lets pull from our GitHub repository again so that our code is up-to-date:

Example

git pull

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), 851 bytes | 9.00 KiB/s, done.

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

\* [new branch] html-skeleton -> origin/html-skeleton

Already up to date.

Now our main branch is up todate. And we can see that there is a new branch available on GitHub.

Do a quick status check:

Example

git status

On branch master

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

nothing to commit, working tree clean

And confirm which branches we have, and where we are working at the moment:

Example

git branch

\* master

So, we do not have the new branch on our local Git. But we know it is available on GitHub. So we can use the -a option to see all local and remote branches:

Example

git branch -a

\* master

remotes/origin/html-skeleton

remotes/origin/master

**Note:** branch -r is for remote branches only.

We see that the branch html-skeleton is available remotely, but not on our local git. Lets check it out:

Example

git checkout html-skeleton

Switched to a new branch 'html-skeleton'

Branch 'html-skeleton' set up to track remote branch 'html-skeleton' from 'origin'.

And check if it is all up to date:

Example

git pull

Already up to date.

Which branches do we have now, and where are we working from?

Example

git branch

\* html-skeleton

master

Now, open your favourite editor and confirm that the changes from the GitHub branch carried over.

## Push a Branch to GitHub

Let's try to create a new local branch, and push that to GitHub.

Start by creating a branch, like we did earlier:

### Example

git checkout -b update-readme

Switched to a new branch 'update-readme'

And we make some changes to the README.md file. Just add a new line.

So now we check the status of the current branch.

### Example

git status

On branch update-readme

Changes not staged for commit:

(use "git add ..." to update what will be committed)

(use "git restore ..." to discard changes in working directory)

modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")

We see that README.md is modified but not added to the Staging Environment:

### Example

git add README.md

Check the status of the branch:

### Example

git status

On branch update-readme

Changes to be committed:

(use "git restore --staged ..." to unstage)

modified: README.md

We are happy with our changes. So we will commit them to the branch:

### Example

git commit -m "Updated readme for GitHub Branches"

[update-readme 836e5bf] Updated readme for GitHub Branches

1 file changed, 1 insertion(+)

Now push the branch from our local Git repository, to GitHub, where everyone can see the changes:

### Example

git push origin update-readme

Enumerating objects: 5, done.

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

Delta compression using up to 16 threads

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

Writing objects: 100% (3/3), 366 bytes | 366.00 KiB/s, done.

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

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

remote:

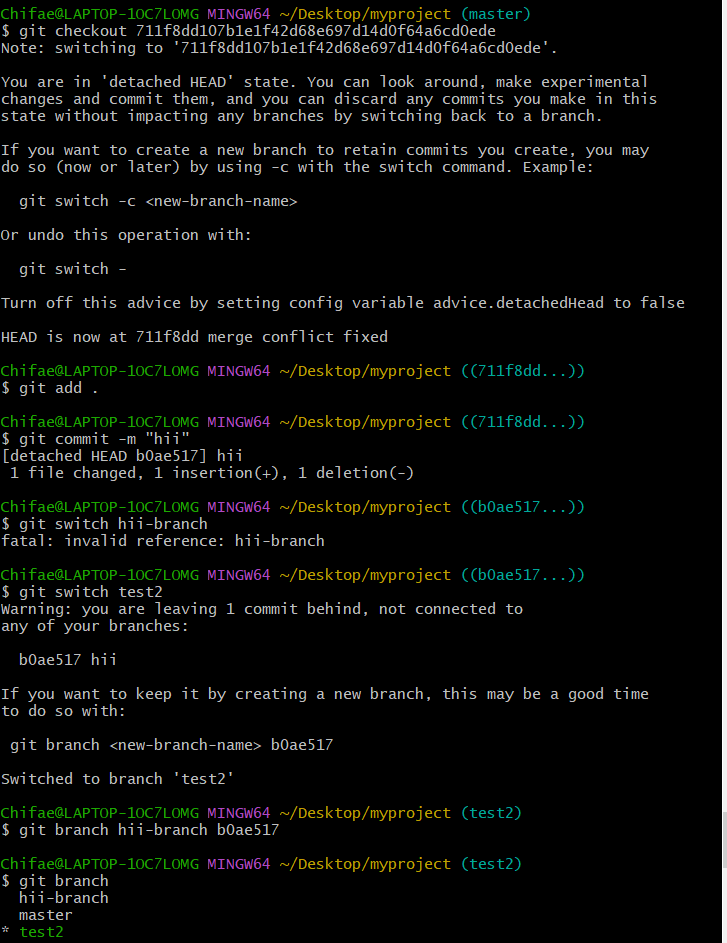
remote: Create a pull request for 'update-readme' on GitHub by visiting:

remote: https://github.com/w3schools-test/hello-world/pull/new/update-readme

remote:

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

\* [new branch] update-readme -> update-readme



## Working using the GitHub Flow

On this page, you will learn how to get the best out of working with GitHub.

The GitHub flow is a workflow designed to work well with Git and GitHub.

It focuses on branching and makes it possible for teams to experiment freely, and make deployments regularly.

The GitHub flow works like this:

* Create a new Branch
* Make changes and add Commits
* Open a Pull Request
* Review
* Deploy
* Merge

You should already have a good understanding of how this works from the previous chapters. This chapter focuses on understanding how the flow makes it easy for you to work together.

## Create a New Branch

Branching is the key concept in Git. And it works around the rule that the master branch is ALWAYS deployable.

That means, if you want to try something new or experiment, you create a new branch! Branching gives you an environment where you can make changes without affecting the main branch.

When your new branch is ready, it can be reviewed, discussed, and merged with the main branch when ready.

When you make a new branch, you will (almost always) want to make it from the master branch.

**Note:** Keep in mind that you are working with others. Using descriptive names for new branches, so everyone can understand what is happening.

## Make Changes and Add Commits

After the new branch is created, it is time to get to work. Make changes by adding, editing and deleting files. Whenever you reach a small milestone, add the changes to your branch by commit.

Adding commits keeps track of your work. Each commit should have a message explaining what has changed and why. Each commit becomes a part of the history of the branch, and a point you can revert back to if you need to.

**Note:** commit messages are very important! Let everyone know what has changed and why. Messages and comments make it so much easier for yourself and other people to keep track of changes.

## Open a Pull Request

Pull requests are a key part of GitHub. A Pull Request notifies people you have changes ready for them to consider or review.

 You can ask others to review your changes or pull your contribution and merge it into their branch.

## Review

When a Pull Request is made, it can be reviewed by whoever has the proper access to the branch. This is where good discussions and review of the changes happen.

Pull Requests are designed to allow people to work together easily and produce better results together!

If you receive feedback and continue to improve your changes, you can push your changes with new commits, making further reviews possible.

**Note:** GitHub shows new commit and feedback in the "unified Pull Request view".

## Deploy

When the pull request has been reviewed and everything looks good, it is time for the final testing. GitHub allows you to deploy from a branch for final testing in production before merging with the master branch.

If any issues arise, you can undo the changes by deploying the master branch into production again!

**Note:** Teams often have dedicated testing environments used for deploying branches.

## Merge

After exhaustive testing, you can merge the code into the master branch!

Pull Requests keep records of changes to your code, and if you commented and named changes well, you can go back and understand why changes and decisions were made.

**Note:** You can add keywords to your pull request for easier searching!

# **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.

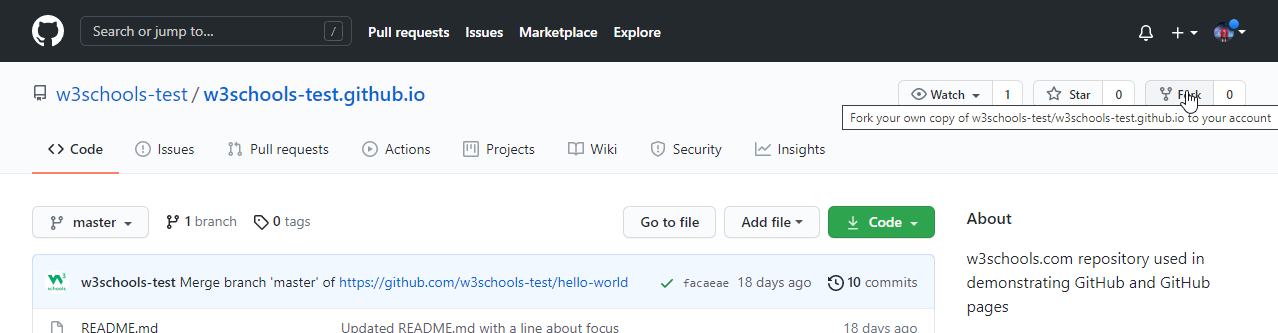
In these next 3 chapters we will show you how to copy a repository, make changes to it, and suggest those changes be implemented to the original repository.

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/

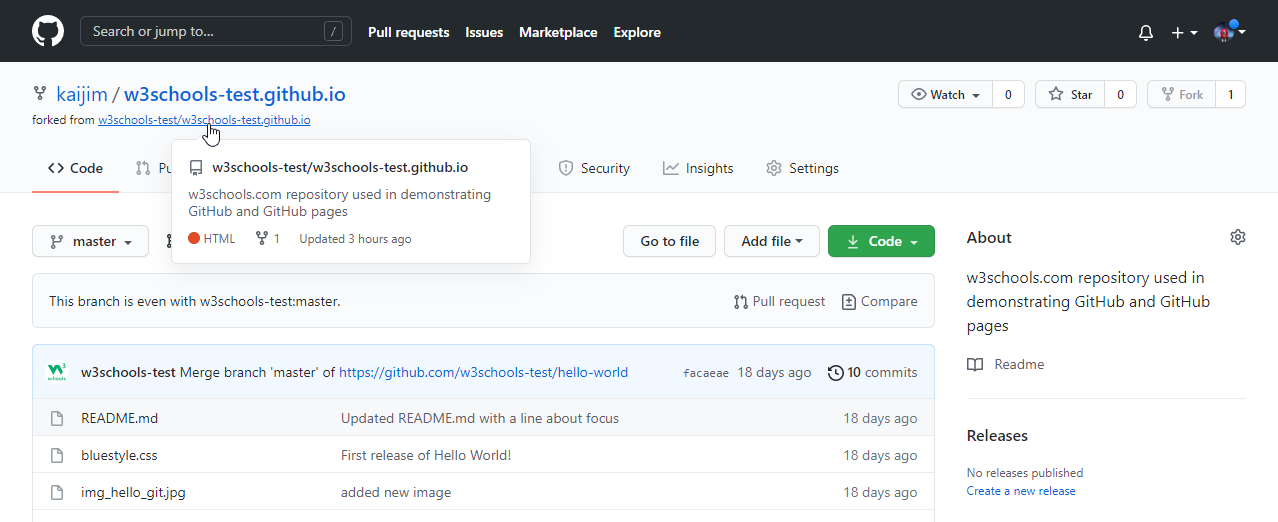
## Fork a Repository

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



Now we have our own copy of w3schools-test.github.io:



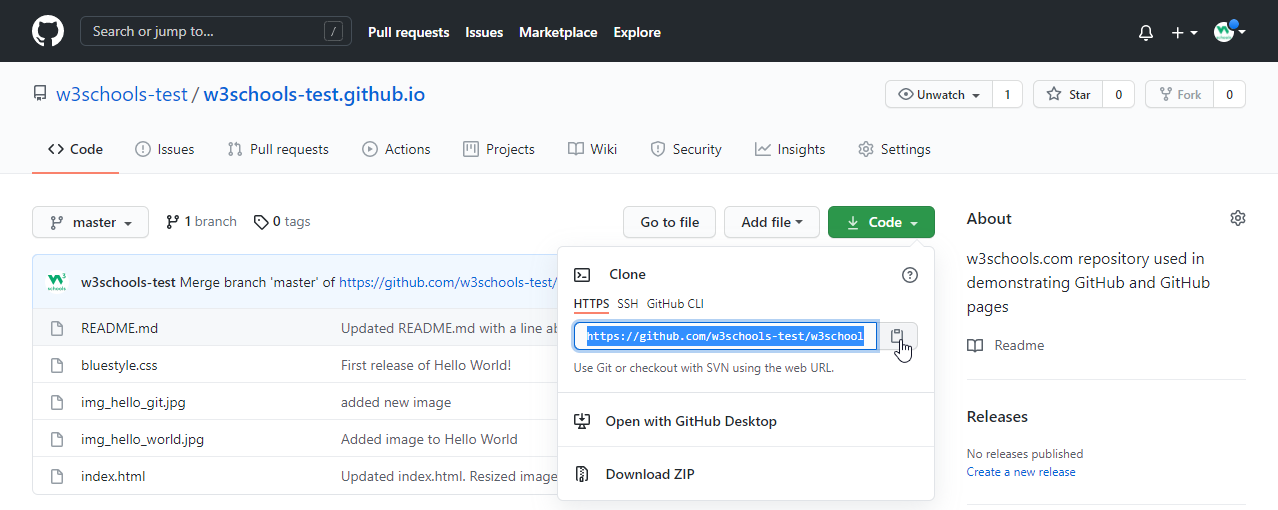
# **Git Clone from GitHub**

## 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/

**Note:** To specify a specific folder to clone to, add the name of the folder after the repository URL, like this: git clone https://github.com/w3schools-test/w3schools-test.github.io.git myfolder

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

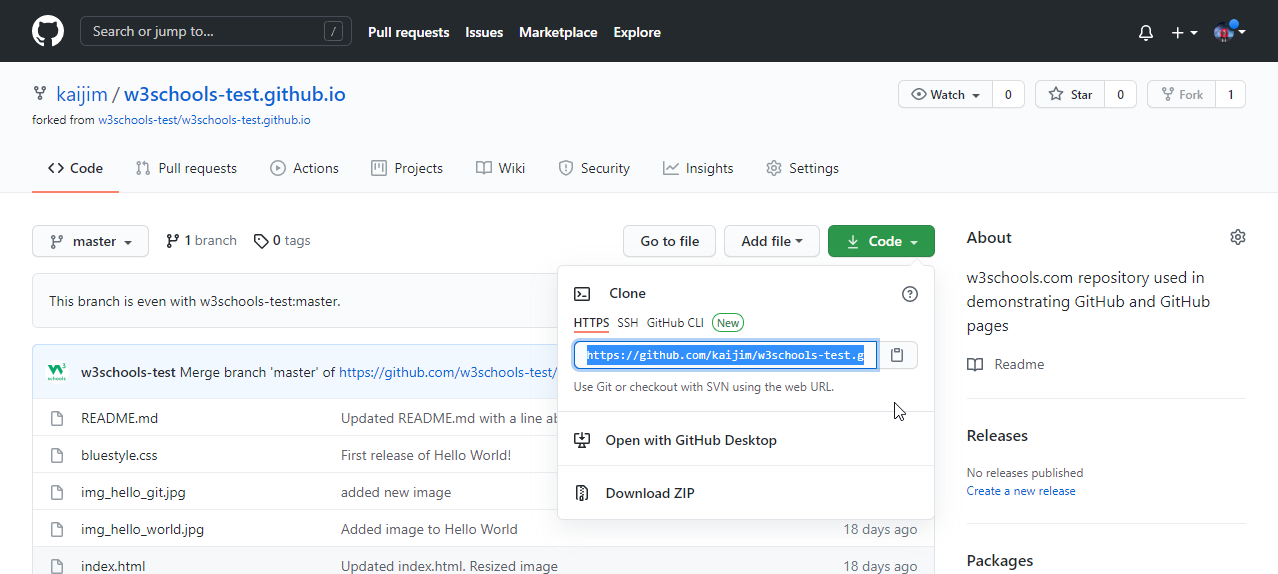
git remote rename origin upstream

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

Now we are going to make some changes to the code. In the next chapter, we will cover how we suggest those changes to the original repository.

# **Git GitHub Send Pull Request**

## Push Changes to Our GitHub Fork

We have made a lot of changes to our local Git.

Now we push them to our GitHub fork:

commit the changes:

### Example

git push origin

Enumerating objects: 8, done.

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

Delta compression using up to 16 threads

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

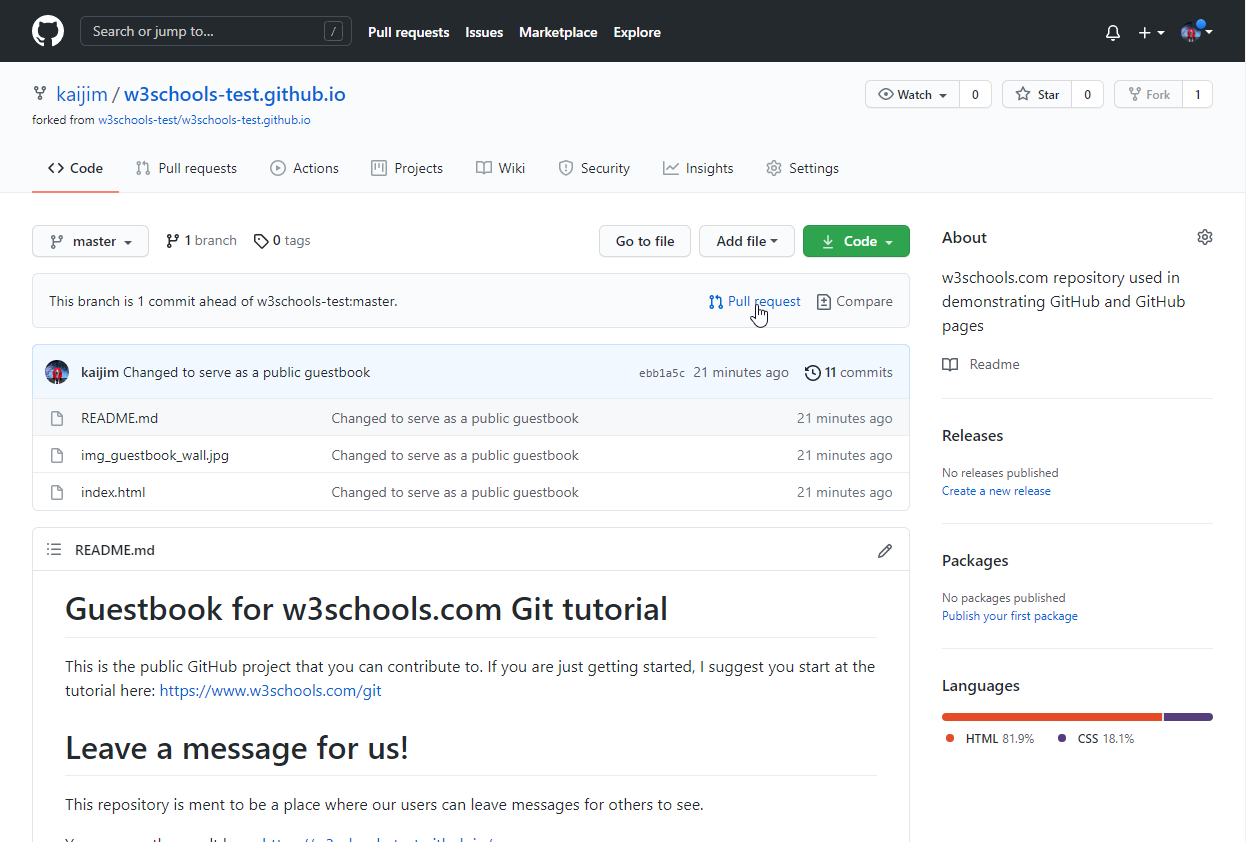
Writing objects: 100% (5/5), 393.96 KiB | 32.83 MiB/s, done.

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

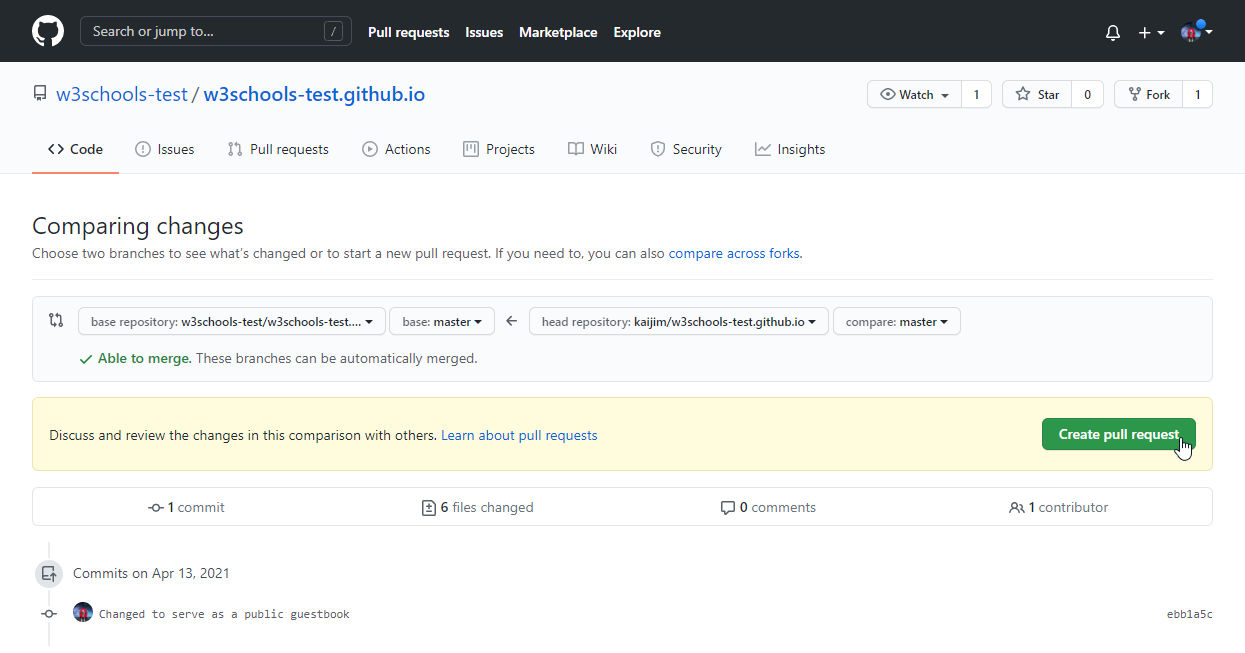
To https://github.com/kaijim/w3schools-test.github.io.git

facaeae..ebb1a5c master -> master

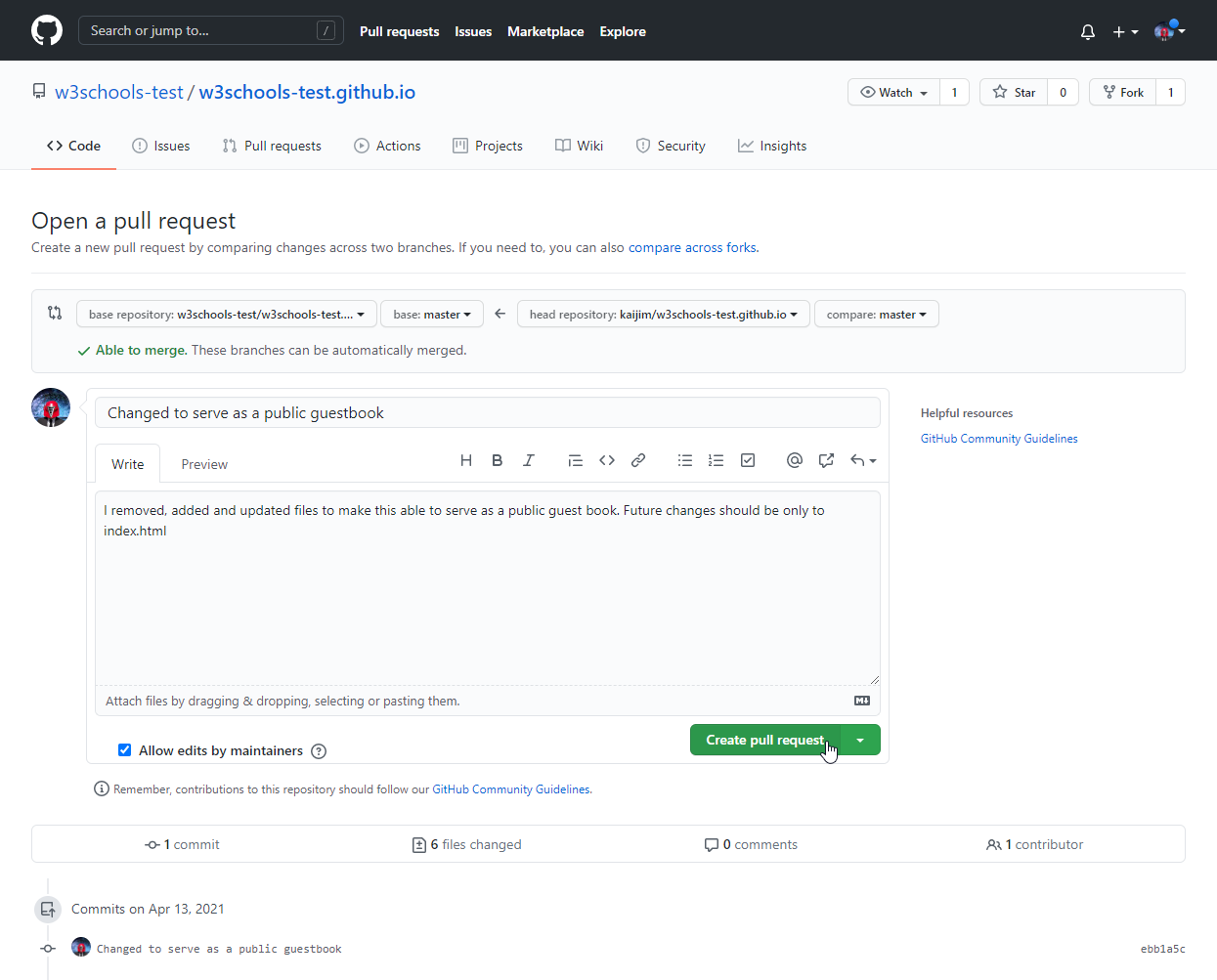
Go to GitHub, and we see that the repository has a new commit. And we can send a Pull Request to the original repository:



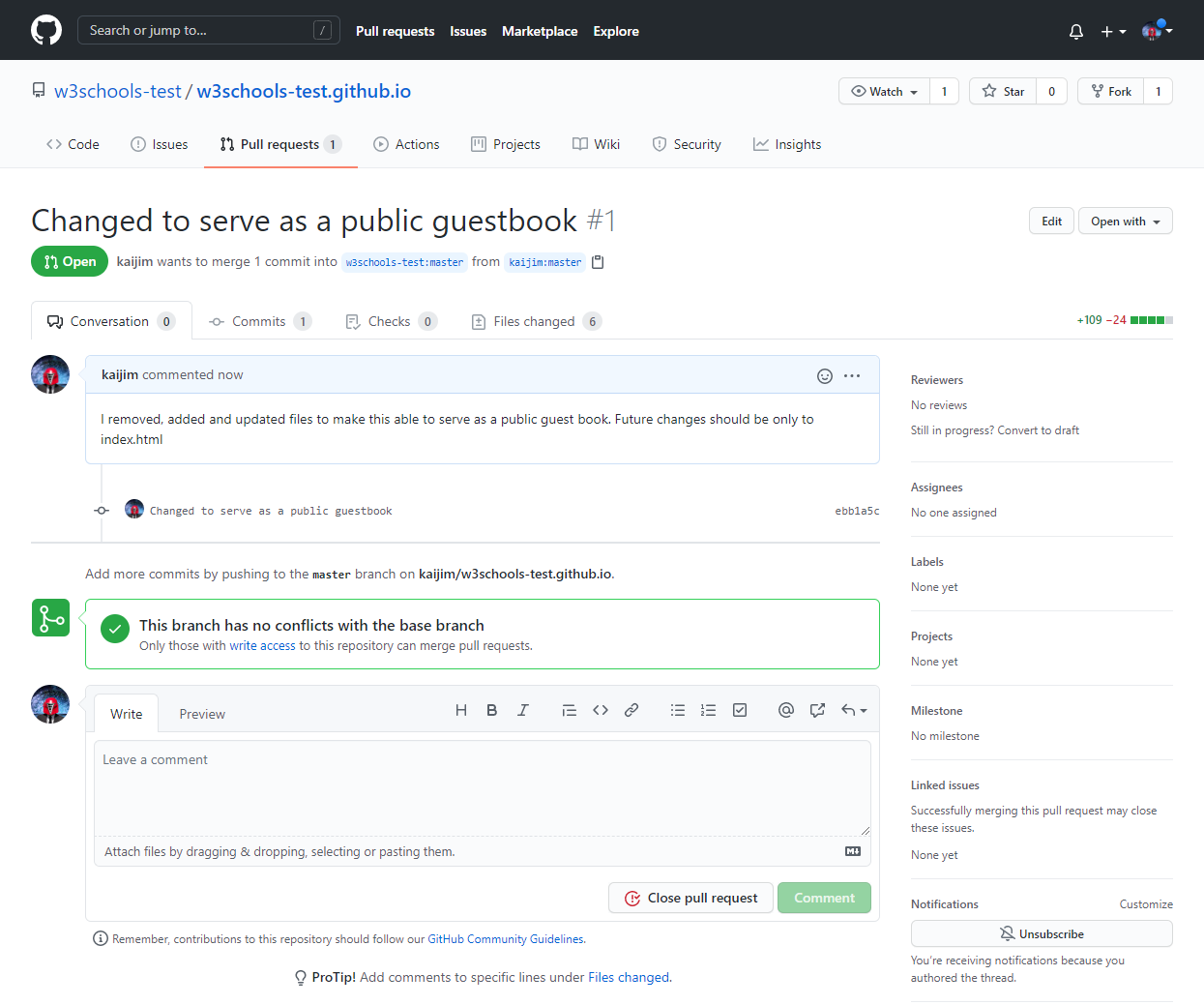
Click that and create a pull request:



Remember to add an explanation for the administrators.

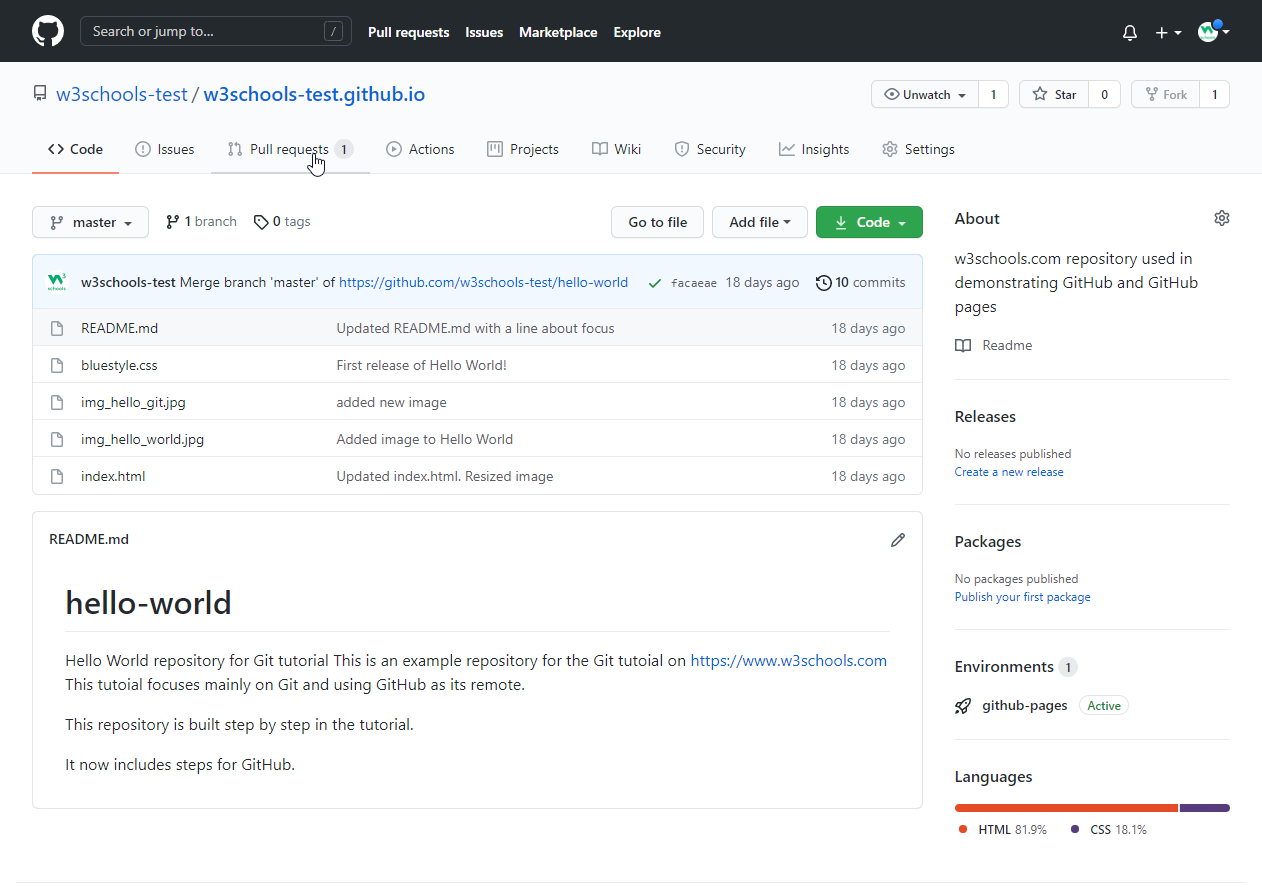


Pull Request is sent:

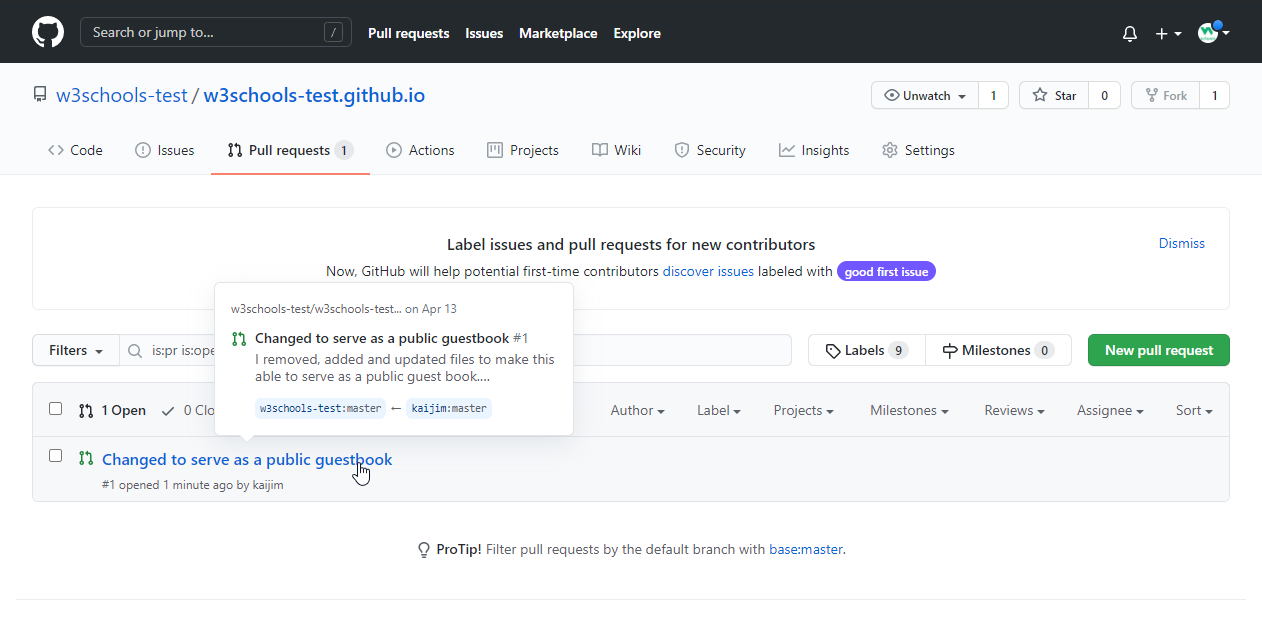


## Approving Pull Requests

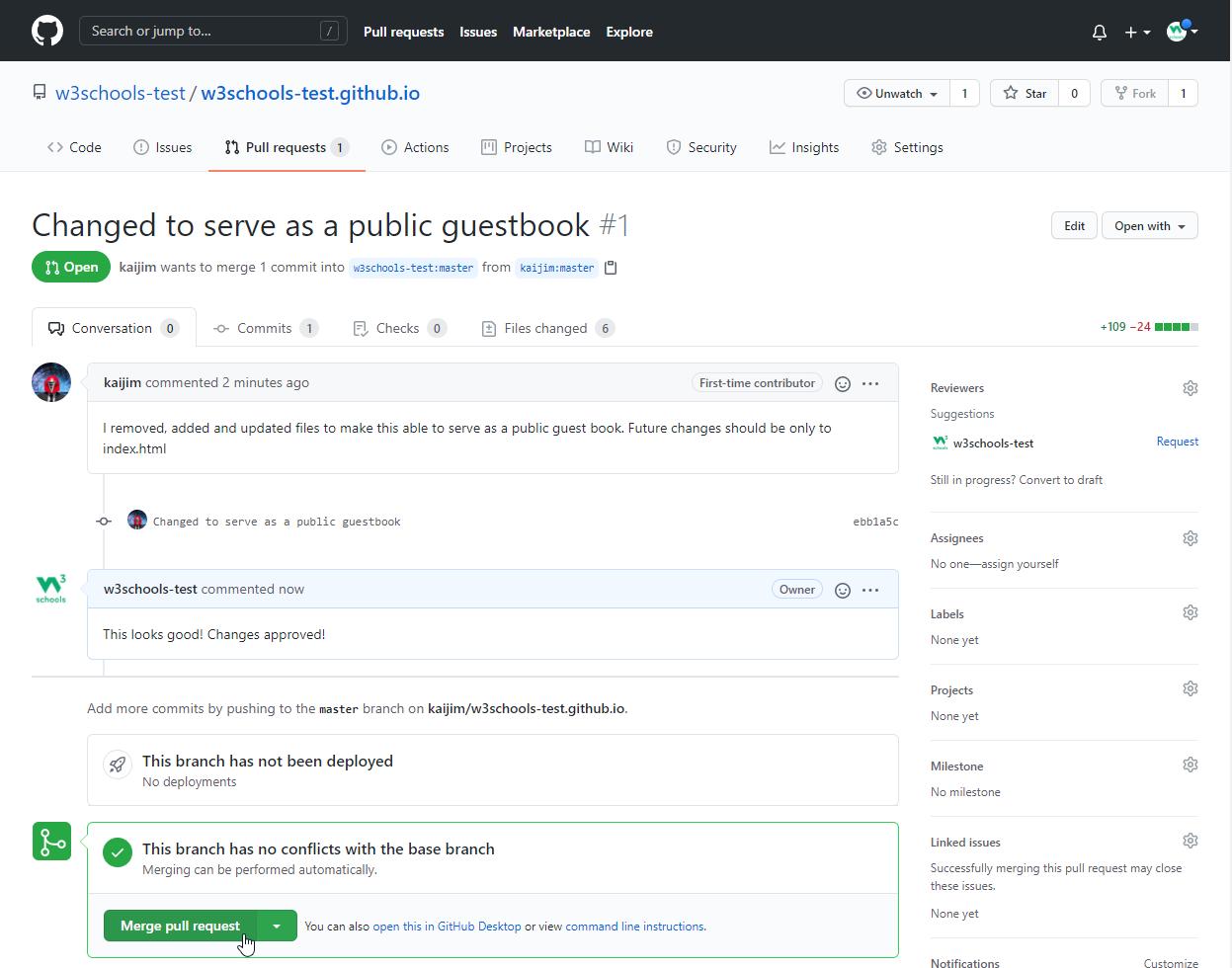
Now any member with access can see the Pull Request when they see the original repository:



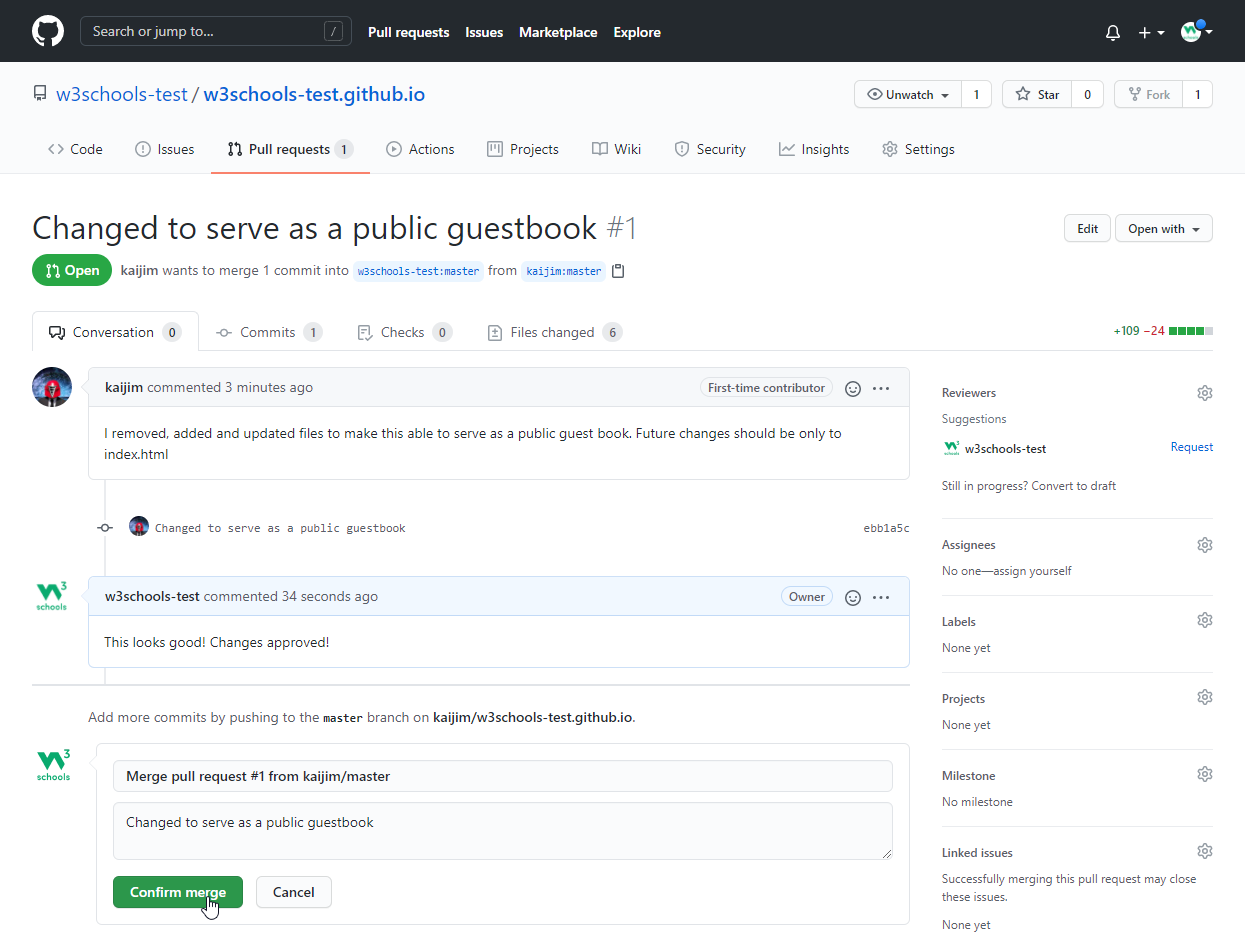
And they can see the proposed changes:



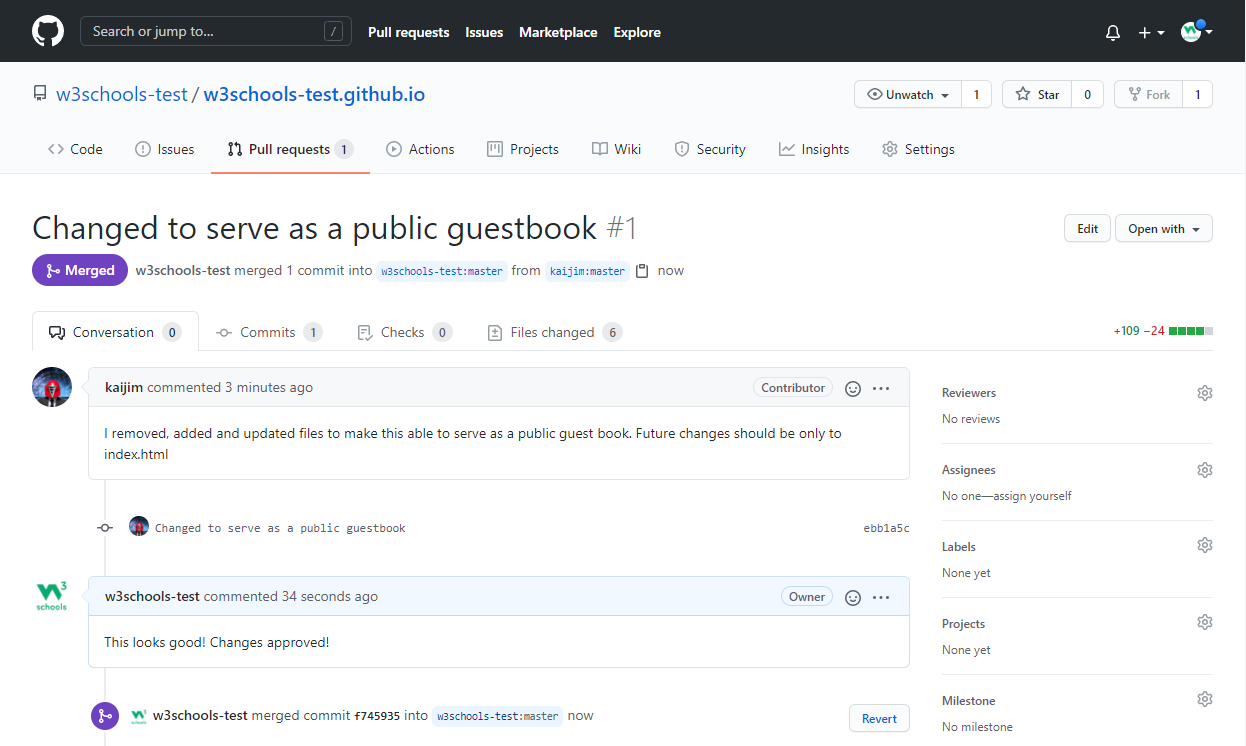
Comment on the changes and merge:



Confirm:



And changes have been merged with master:



# **Git Ignore and .gitignore**

## sGit Ignore

When sharing your code with others, there are often files or parts of your project, you do not want to share.

Examples

* log files
* temporary files
* hidden files
* personal files
* etc.

Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

Git will not track files and folders specified in .gitignore. However, the .gitignore file itself **IS** tracked by Git.

## Create .gitignore

To create a .gitignore file, go to the root of your local Git, and create it:

### Example

touch .gitignore

Now open the file using a text editor.

We are just going to add two simple rules:

* Ignore any files with the .log extension
* Ignore everything in any directory named temp

### Example

# ignore ALL .log files  
\*.log  
  
# ignore ALL files in ANY directory named temp  
temp/

Now all .log files and anything in temp folders will be ignored by Git.

**Note:** In this case, we use a single .gitignore which applies to the entire repository.

It is also possible to have additional .gitignore files in subdirectories. These only apply to files or folders within that directory.

## Rules for .gitignore

Here are the general rules for matching patterns in .gitignore files:

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Explanation/Matches** | **Examples** |
|  | Blank lines are ignored |  |
| # text comment | Lines starting with # are ignored |  |
| name | All name files, name folders, and files and folders in any name folder | /name.log /name/file.txt /lib/name.log |
| name/ | Ending with / specifies the pattern is for a folder. Matches all files and folders in any name folder | /name/file.txt /name/log/name.log  **no match:** /name.log |
| name.file | All files with the name.file | /name.file /lib/name.file |
| /name.file | Starting with / specifies the pattern matches only files in the root folder | /name.file  **no match:** /lib/name.file |
| lib/name.file | Patterns specifiing files in specific folders are always realative to root (even if you do not start with / ) | /lib/name.file  **no match:** name.file /test/lib/name.file |
| \*\*/lib/name.file | Starting with \*\* before / specifies that it matches any folder in the repository. Not just on root. | /lib/name.file /test/lib/name.file |
| \*\*/name | All name folders, and files and folders in any name folder | /name/log.file /lib/name/log.file /name/lib/log.file |
| /lib/\*\*/name | All name folders, and files and folders in any name folder within the lib folder. | /lib/name/log.file /lib/test/name/log.file /lib/test/ver1/name/log.file  **no match:** /name/log.file |
| \*.file | All files withe .file extention | /name.file /lib/name.file |
| \*name/ | All folders ending with name | /lastname/log.file /firstname/log.file |
| name?.file | ? matches a **single** non-specific character | /names.file /name1.file  **no match:** /names1.file |
| name[a-z].file | [range] matches a **single** character in the specified range (in this case a character in the range of a-z, and also be numberic.) | /names.file /nameb.file  **no match:** /name1.file |
| name[abc].file | [set] matches a **single** character in the specified set of characters (in this case either a, b, or c) | /namea.file /nameb.file  **no match:** /names.file |
| name[!abc].file | [!set] matches a **single** character, **except** the ones spesified in the set of characters (in this case a, b, or c) | /names.file /namex.file  **no match:** /namesb.file |
| \*.file | All files withe .file extention | /name.file /lib/name.file |
| name/ !name/secret.log | ! specifies a negation or exception. Matches all files and folders in any name folder, except name/secret.log | /name/file.txt /name/log/name.log  **no match:** /name/secret.log |
| \*.file!name.file | ! specifies a negation or exception. All files withe .file extention, except name.file | /log.file /lastname.file  **no match:** /name.file |
| \*.file!name/\*.file junk.\* | Adding new patterns after a negation will re-ignore a previous negated file All files withe .file extention, except the ones in name folder. Unless the file name is junk | /log.file /name/log.file  **no match:** /name/junk.file |

## Local and Personal Git Ignore Rules

It is also possible to ignore files or folders but not show it in the distributed .gitignore file.

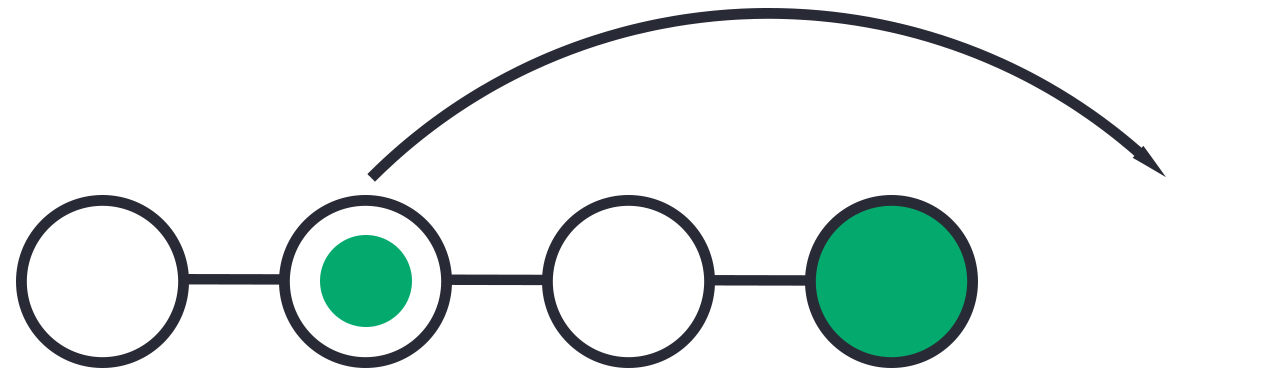
These kinds of ignores are specified in the .git/info/exclude file. It works the same way as .gitignore but are not shown to anyone else.

# **Git Revert**

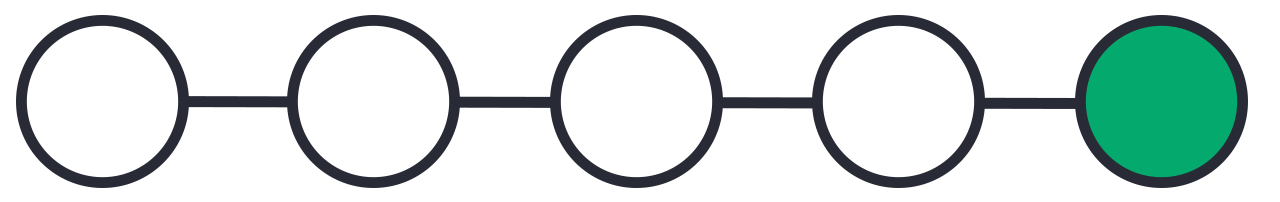
## Git Revert

revert is the command we use when we want to take a previous commit and add it as a new commit, keeping the log intact.

Step 1: Find the previous commit:



Step 2: Use it to make a new commit:



Let's make a new commit, where we have "accidentally" deleted a file:

### Example

git commit -m "Just a regular update, definitely no accidents here..."

[master 16a6f19] Just a regular update, definitely no accidents here...

1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100644 img\_hello\_git.jpg

Now we have a part in our commit history we want to go back to. Let's try and do that with revert.

## Git Revert Find Commit in Log

First thing, we need to find the point we want to return to. To do that, we need to go through the log.

To avoid the very long log list, we are going to use the --oneline option, which gives just one line per commit showing:

* The first seven characters of the commit hash
* the commit message

So let's find the point we want to revert:

### Example

git log --oneline

52418f7 (HEAD -> master) Just a regular update, definitely no accidents here...

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

We want to revert to the previous commit: 52418f7 (HEAD -> master) Just a regular update, definitely no accidents here..., and we see that it is the latest commit.

## Git Revert HEAD

We revert the latest commit using git revert HEAD (revert the latest change,  and then commit), adding the option --no-edit to skip the commit message editor (getting the default revert message):

### Example

git revert HEAD --no-edit

[master e56ba1f] Revert "Just a regular update, definitely no accidents here..."

Date: Thu Apr 22 10:50:13 2021 +0200

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 img\_hello\_git.jpg

Now let's check the log again:

### Example

git log --oneline

e56ba1f (HEAD -> master) Revert "Just a regular update, definitely no accidents here..."

52418f7 Just a regular update, definitely no accidents here...

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

**Note:** To revert to earlier commits, use git revert HEAD~x (*x* being a number. 1 going back one more, 2 going back two more, etc.)

On the next page, we'll go over git reset, which brings the repository back to an earlier state in the commits without making a new commit.

# **Git Reset**

[❮ Previous](https://www.w3schools.com/git/git_revert.asp?remote=github)[Next ❯](https://www.w3schools.com/git/git_amend.asp?remote=github)

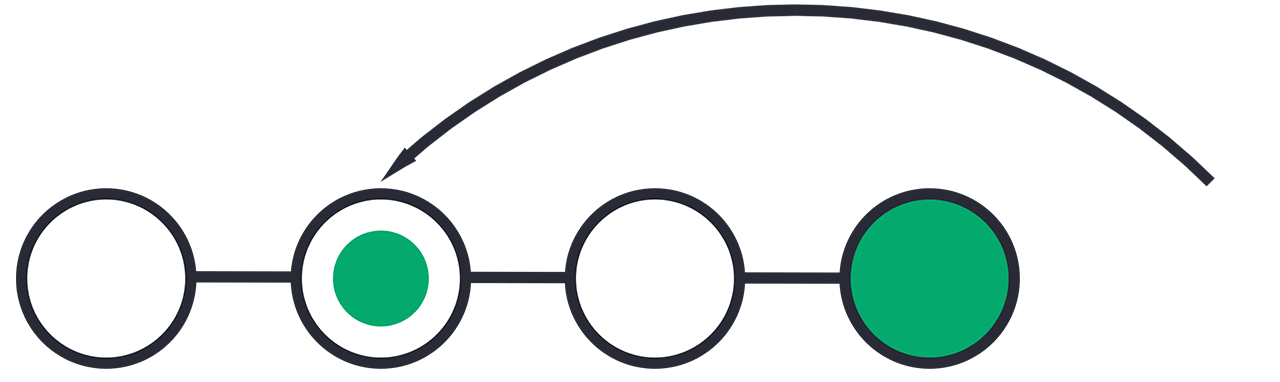
## Change Platform:

[[](https://www.w3schools.com/git/git_reset.asp?remote=github)GitHub](https://www.w3schools.com/git/git_reset.asp?remote=github)[[](https://www.w3schools.com/git/git_reset.asp?remote=bitbucket)Bitbucket](https://www.w3schools.com/git/git_reset.asp?remote=bitbucket)[[](https://www.w3schools.com/git/git_reset.asp?remote=gitlab)GitLab](https://www.w3schools.com/git/git_reset.asp?remote=gitlab)

## Git Reset

reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

Step 1: Find the previous commit:



Step 2: Move the repository back to that step:



After the previous chapter, we have a part in our commit history we could go back to. Let's try and do that with reset.

## Git Reset Find Commit in Log

First thing, we need to find the point we want to return to. To do that, we need to go through the log.

To avoid the very long log list, we are going to use the --oneline option, which gives just one line per commit showing:

* The first seven characters of the commit hash - this is what we need to refer to in our reset command.
* the commit message

So let's find the point we want to reset to:

### Example

git log --oneline

e56ba1f (HEAD -> master) Revert "Just a regular update, definitely no accidents here..."

52418f7 Just a regular update, definitely no accidents here...

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

We want to return to the commit: 9a9add8 (origin/master) Added .gitignore, the last one before we started to mess with things.

## Git Reset

We reset our repository back to the specific commit using git reset commithash (commithash being the first 7 characters of the commit hash we found in the log):

### Example

git reset 9a9add8

Now let's check the log again:

### Example

git log --oneline

9a9add8 (HEAD -> master, origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

**Warning:** Messing with the commit history of a repository can be dangerous. It is usually ok to make these kinds of changes to your own local repository. However, you should avoid making changes that rewrite history to remote repositories, especially if others are working with them.

## Git Undo Reset

Even though the commits are no longer showing up in the log, it is not removed from Git.

If you know the commit hash you can reset to it:

### Example

git reset e56ba1f

Now let's check the log again:

### Example

git log --oneline

e56ba1f (HEAD -> master) Revert "Just a regular update, definitely no accidents here..."

52418f7 Just a regular update, definitely no accidents here...

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

**Git Amend**

## Git commit --amend

commit --amend is used to modify the most recent commit.

It combines changes in the staging environment with the latest commit, and creates a new commit.

This new commit replaces the latest commit entirely.

## Git Amend Commit Message

One of the simplest things you can do with --amend is to change a commit message.

Let's update the README.md and commit:

### Example

git commit -m "Adding plines to reddme"

[master 07c5bc5] Adding plines to reddme

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

Now let's check the log:

### Example

git log --oneline

07c5bc5 (HEAD -> master) Adding plines to reddme

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

Oh no! the commit message is full of spelling errors. Embarrassing. Let's amend that:

### Example

git commit --amend -m "Added lines to README.md"

[master eaa69ce] Added lines to README.md

Date: Thu Apr 22 12:18:52 2021 +0200

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

And re-check the log:

### Example

git log --oneline

eaa69ce (HEAD -> master) Added lines to README.md

9a9add8 (origin/master) Added .gitignore

81912ba Corrected spelling error

3fdaa5b Merge pull request #1 from w3schools-test/update-readme

836e5bf (origin/update-readme, update-readme) Updated readme for GitHub Branches

daf4f7c (origin/html-skeleton, html-skeleton) Updated index.html with basic meta

facaeae (gh-page/master) Merge branch 'master' of https://github.com/w3schools-test/hello-world

e7de78f Updated index.html. Resized image

5a04b6f Updated README.md with a line about focus

d29d69f Updated README.md with a line about GitHub

e0b6038 merged with hello-world-images after fixing conflicts

1f1584e added new image

dfa79db updated index.html with emergency fix

0312c55 Added image to Hello World

09f4acd Updated index.html with a new line

221ec6e First release of Hello World!

We see the previous commit is replaced with our amended one!

**Warning:** Messing with the commit history of a repository can be dangerous. It is usually ok to make these kinds of changes to your own local repository. However, you should avoid making changes that rewrite history to remote repositories, especially if others are working with them.

## Git Amend Files

Adding files with --amend works the same way as above. Just add them to the staging environment before committing.