



Git for Version Control

Rujuan Xing

Version Control System Features

- Collaborative Development
- Data backup
 - Keeps records of folders and files and their history
- Version Management
 - Keep records of your changes without duplication.
- Access Control
- History
 - Allows you to know who made what changes and when
 - Allows you to revert any changes and go back to a previous state
- Branch Management

Git

- A version control system for tracking changes in computer files
- Coordinate work on those files among multiple people
- primarily used for source code management in software development
- Initially created by Linus Torvalds for development of the Linux Kernel
- Pros
 - Most of the operations can be done locally, no need to connect to internet
 - Integrity
 - Excellent support for parallel development, support for hundreds of parallel branches
 - Fully compatible with Linux commands
- Install Git
 - <https://git-scm.com/downloads>
 - Change Default VS Code As The Git Editor (Globally)
 - `git config --global core.editor "code --wait"`

Git Configuration

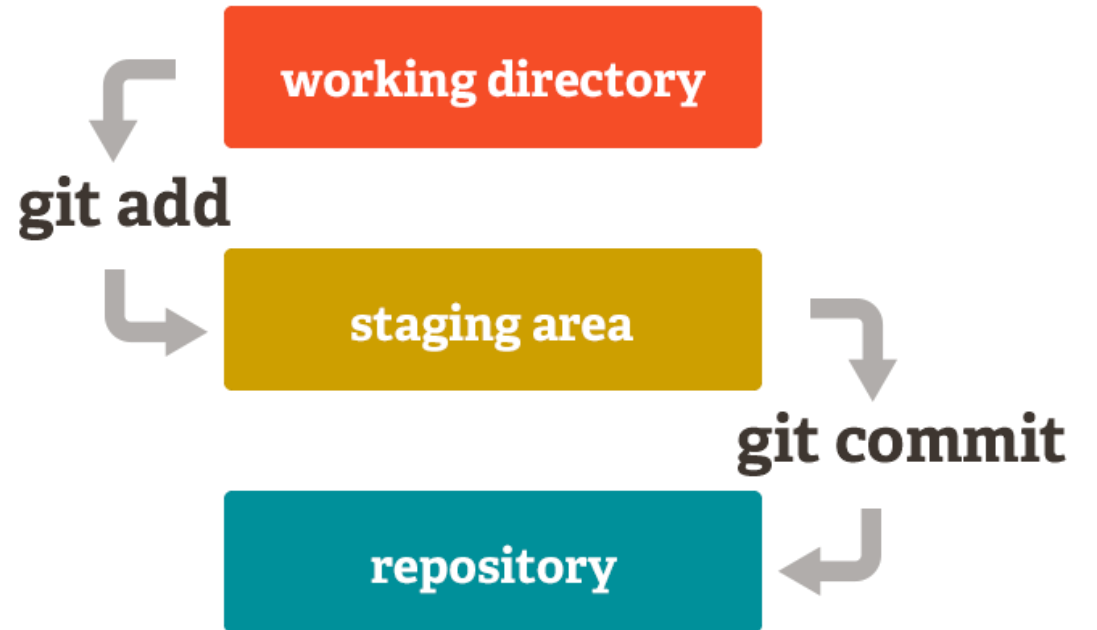
- Designed to configure or read the corresponding environment variable
- Can be stored in three different places
 - `/etc/gitconfig` file: The system for all users of general application configuration
 - `C:/Program Files/Git/etc`
 - `git config --system`
 - `~/.gitconfig` file: user profile directory apply only to that user.
 - `C:/Users/rxing`
 - `git config --global`
 - `.git/config` file: Git directory of the current project configuration file – use `git init` to add
 - `H:/courses/MSD/445/Demos/.git`
 - `git config`

Git Configuration

- User Info
 - `git config user.name tina`
 - `git config user.email rxing@miu.edu`
- View configuration information. If you don't specify which of the configs you would like to see, **you will get all 3 configs merged** into the output in your console.
 - `git config --list`
 - `git config --list --system`
 - `git config --list --global`
 - `git config --list --local`

Git Main Components

- Working Tree
 - Where you're currently working, where your files live
 - "untracked" area of git
 - Git is not aware of the files or changes in the working tree until you tell it to pay attention to them.
- Staging Area
 - Git starts tracking and saving changes that occurs in files
 - The saved changes reflect in the `.git` directory
- Local Repo
 - Everything in your `.git` directory
 - All of your checkpoints or commits are in Local Repository
 - Don't delete it.



Branch

- A branch is when a new line of development is created that splits the main line of development. This alternative line of development can continue without altering the main line.
- By default, any repo will have one **master** branch
- The special **HEAD** pointer has a reference to currently **Active Branch**

.gitignore

- Git sees every file in your working copy as one of three things:
 - 1.tracked - a file which has been previously staged or committed;
 - 2.untracked - a file which *has not* been staged or committed;
 - 3.ignored - a file which Git has been explicitly told to ignore.
- Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are:
 - dependency caches, such as the contents of /node_modules or /packages
 - compiled code, such as .o, .pyc, and .class files
 - build output directories, such as /bin, /out, or /target
 - files generated at runtime, such as .log, .lock, or .tmp
 - hidden system files, such as .DS_Store or Thumbs.db
 - personal IDE config files, such as .idea/workspace.xml

Basic Operations

git init

- Create a new, empty repository in the current directory.
- It sets up all the necessary files and directories that Git will use to keep track of everything. All these files are stored in a directory called `.git`
- This `.git` directory is the repo, it is where Git records all of the commits and keeps track of everything.

courses > MSD > 445 > cs445 > .git

Name ^

- hooks
- info
- logs
- objects
- refs
- config
- description
- FETCH_HEAD
- HEAD
- index
- packed-refs

git add

- Move files from the Working Directory to the Staging Area.
- The period `.` can be used in place of a list of files to tell Git to add the current directory (and all nested files).

```
H:\courses\MSD\cs445\project>git add hello.js
```

git status

- Display the current status of the repository
 - The current Git **Branch** in your local repository
 - Tell us about new files that have been created in the **Working Directory** that Git hasn't started tracking, yet
 - Files that Git is tracking in the **Staging area** that have been modified

```
H:\courses\MSD\445\cs445>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   test.js
```

git commit -m "message"

- Takes files from the Staging Area and saves them in the repository
- A commit should explain **what** the commit does (not how or why).
- The goal is that each commit has a single focus. Each commit should record a single-unit change.
- Do not use the word "and", if you have to use "and", your commit message is probably doing too many changes - break the changes into separate commits.

```
H:\courses\MSD\cs445\project>git commit -m "init commit"
[master (root-commit) 9767144] init commit
1 file changed, 1 insertion(+)
create mode 100644 hello.js

H:\courses\MSD\cs445\project>git status
On branch master
nothing to commit, working tree clean
```

git diff

- To see changes that have been made to files in the **working area** but haven't been added into staging area yet.

```
H:\courses\MSD\cs445\project>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.js

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

H:\courses\MSD\cs445\project>git diff hello.js
diff --git a/hello.js b/hello.js
index 71d4a77..7d5dc0a 100644
--- a/hello.js
+++ b/hello.js
@@ -1,2 +1,3 @@
  let x = 10;
  console.log(x);
+let y = 3;
```

git log

- Displays all the **commits of a repository**
- `git log` view history
 - space – page down
 - b – page up
 - q – exit
- `git log --oneline` lists one commit per line
- `git log --all --pretty=format:"%h %cd %s (%an)" --since="7 days ago"` review changes made in the last week
- `git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short`
 - `--pretty="..."` defines the format of the output.
 - `%h` is the abbreviated hash of the commit
 - `%d` are any decorations on that commit (e.g. branch heads or tags)
 - `%ad` is the author date
 - `%s` is the comment
 - `%an` is the author name
 - `--graph` informs git to display the commit tree in an ASCII graph layout
 - `--date=short` keeps the date format nice and short

Aliases

- Shortcuts for git commands
- Change .gitconfig in global or project level as you want.
- Make sure the value of property is single quote in config file.
 - `git config --global alias.st status`
 - `git config alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"`

```
[alias]
co = checkout
ci = commit
st = status
br = branch
hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
```

```
H:\courses\MSD\445\git-demo>git hist
* 610844d 2021-05-16 | init commit (HEAD -> master) [rxing]
```


Get Old Versions

- Checkout any previous snapshot into the working directory
- Use the hashes for previous versions
 - `git checkout <hash>`
- Return the latest version in the `master` branch
 - `git checkout master`
 - `master` is the name of the default branch

```
* 326c0dc 2021-05-03 | add function bar [rxing]
* 45d58dd 2021-05-03 | add function f [rxing]
* c2b0e37 2021-05-03 | print y [rxing]
* 1765063 2021-05-03 | add y=3 [rxing]
* 5b99be8 2021-05-03 | removed console.log(), added let x=10 [rxing]
* 9767144 2021-05-03 | init commit [rxing]
```

```
H:\courses\MSD\cs445\project>git checkout 326c0dc
Note: switching to '326c0dc'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false

HEAD is now at 326c0dc add function bar

git tag -a <tagname>

- Add a marker on a specific commit. The tag does not move around as new commits are added.
- Tags can be viewed in log
- `git tag` will display all tags that are in the repository.
- `git tag -a <tagname> -m <message> <SHA>`
- `git tag -d <tagname>` will delete a tag

```
H:\courses\MSD\cs445\project>git tag
```

```
H:\courses\MSD\cs445\project>git tag v1
```

```
H:\courses\MSD\cs445\project>git checkout v1
```

```
Note: switching to 'v1'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.
```

```
If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:
```

```
git switch -c <new-branch-name>
```

```
Or undo this operation with:
```

```
git switch -
```

```
Turn off this advice by setting config variable advice.detachedHead to false
```

```
HEAD is now at c5fcc3a made a change in master to create a conflict with order branch
```

Undoing Local Changes (before staging)

- Revert changes in the working directory
- After making changes in working area, check status

```
H:\courses\MSD\cs445\project>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.js

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

- Revert the changes in the working directory

```
H:\courses\MSD\cs445\project>git restore hello.js
```

Undoing Staged Changes (before committing)

- Revert changes that have been staged

```
H:\courses\MSD\cs445\project>notepad hello.js  
  
H:\courses\MSD\cs445\project>git add hello.js  
  
H:\courses\MSD\cs445\project>git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
        modified:   hello.js
```

- Reset the Staging Area

```
H:\courses\MSD\cs445\project>git restore --staged hello.js  
  
H:\courses\MSD\cs445\project>git status  
On branch master  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
        modified:   hello.js  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

Undoing Committed Changes

- Revert changes that have been committed to a local repository.
- To undo a committed change, we need to generate a commit that removes the changes introduced by our unwanted commit.
 - `git revert HEAD`
- Since we were undoing the very last commit we made, we were able to use `HEAD` as the argument to `revert`. We can revert any arbitrary commit earlier in history by simply specifying its `hash` value.
- The `revert` command of the previous section is a powerful command that lets us undo the effects of any commit in the repository. However, both the original commit and the “undoing” commit are visible in the branch history (using the `git log` command).

Removing Commits from a Branch

- When given a commit reference (i.e. a hash, branch or tag name), the `reset` command will ...

1. Rewrite the current branch to point to the specified commit
2. Optionally reset the staging area to match the specified commit
3. Optionally reset the working directory to match the specified commit

```
H:\courses\MSD\cs445\project>git reset --hard v1  
HEAD is now at c5fcc3a made a change in master to create a conflict with order branch
```

- Our **master** branch now points to the `v1` commit and other commits after `v1` commit are no longer in the branch. The `--hard` parameter indicates that the working directory should be updated to be consistent with the new branch head.

Updating The Last Commit

- To **update the most-recent commit** instead of creating a new commit

```
git commit --amend -m "Your new commit message"
```

- The `--no-edit` flag will allow you to make the amendment to your commit without changing its commit message.
- Amended commits are actually entirely new commits and the previous commit will no longer be on your current branch.

```
H:\courses\MSD\cs445\project>git commit -m "wrong comment" hello.js
[master 5248ecd] wrong comment
1 file changed, 1 insertion(+)

H:\courses\MSD\cs445\project>git log --oneline
5248ecd (HEAD -> master) wrong comment
5b99be8 removed console.log(), added let x=10
9767144 init commit
```

```
H:\courses\MSD\cs445\project>git commit --amend -m "add y=3" hello.js
[master 1765063] add y=3
Date: Mon May 3 16:06:31 2021 -0500
1 file changed, 1 insertion(+)

H:\courses\MSD\cs445\project>git log --oneline
1765063 (HEAD -> master) add y=3
5b99be8 removed console.log(), added let x=10
9767144 init commit
```

Branch Operations

git branch

- List all branch names in the repository, A branch is used to do development or make a fix to the project that won't affect the project.
- Once you make the change on the branch, you can combine that branch into the master branch (merging).
 - `git branch <branchname>` will create a new branch
 - `git branch -d <branchname>` will delete a branch
 - `git checkout/switch <branchname>` will switch to a branch
- You can't delete a branch that you're currently on. To delete a branch, you'd have to switch to either the master branch or create and switch to a new branch.

```
H:\courses\MSD\cs445\project>git branch order
H:\courses\MSD\cs445\project>git branch -v
* master 222eb59 add function foo
  order  222eb59 add function foo
H:\courses\MSD\cs445\project>git branch
* master
  order
H:\courses\MSD\cs445\project>git checkout order
Switched to branch 'order'
H:\courses\MSD\cs445\project>
```

Use Case: deal with multiple branches with different (and possibly conflicting) changes

1. In order branch

- i. Add a new file named `order.js` with `placeOrder()`
- ii. Call `placeOrder()` in `hello.js`
- iii. Add and commit

2. Switch to master branch

- i. `git checkout master`
- ii. `hello.js` remains no change

3. Create and commit README to master

- i. `git add README`
- ii. `git commit -m "Added README"`

```
H:\courses\MSD\cs445\project>git hist --all
* 67c1c3d 2021-05-03 | Added README (HEAD -> master) [rxing]
* b8685a0 2021-05-03 | use placeOrder() in hello.js (order) [rxing]
* 9445858 2021-05-03 | add placeOrder() [rxing]
* d10c5ca 2021-05-03 | Order branch init commit [rxing]
/
* 222eb59 2021-05-03 | add function foo [rxing]
* 326c0dc 2021-05-03 | add function bar [rxing]
* 45d58dd 2021-05-03 | add function f [rxing]
* c2b0e37 2021-05-03 | print y [rxing]
* 1765063 2021-05-03 | add y=3 [rxing]
* 5b99be8 2021-05-03 | removed console.log(), added let x=10 [rxing]
* 9767144 2021-05-03 | init commit [rxing]
```

git merge <name-of-branch-to-merge-in>

- Merging brings the changes in two branches together.
- Steps:
 - `git checkout <branchname>` - switch to the branch you want to merge other changes
 - `git merge <name-of-branch-to-merge-in>` - merge

Merge master to order branch

```
H:\courses\MSD\cs445\project>git checkout order
Switched to branch 'order'

H:\courses\MSD\cs445\project>git merge master
Merge made by the 'recursive' strategy.
 README | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README

H:\courses\MSD\cs445\project>git hist --all
*   fabcf88 2021-05-03 | Merge branch 'master' into order (HEAD -> order) [rxing]
|
| * 67c1c3d 2021-05-03 | Added README (master) [rxing]
| * | b8685a0 2021-05-03 | use placeOrder() in hello.js [rxing]
| * | 9445858 2021-05-03 | add placeOrder() [rxing]
| * | d10c5ca 2021-05-03 | Order branch init commit [rxing]
|/
* 222eb59 2021-05-03 | add function foo [rxing]
* 326c0dc 2021-05-03 | add function bar [rxing]
* 45d58dd 2021-05-03 | add function f [rxing]
* c2b0e37 2021-05-03 | print y [rxing]
* 1765063 2021-05-03 | add y=3 [rxing]
* 5b99be8 2021-05-03 | removed console.log(), added let x=10 [rxing]
* 9767144 2021-05-03 | init commit [rxing]
```

Use Case: Merge Conflict

1. Switch to master, edit hello.js, then commit to master

i. git add hello.js

ii. git commit -m "Changes in master"

2. Merge master to order branch

i. git merge master

3. Conflict happens

i. Open/view hello.js

```
<<<<<<< HEAD
placeOrder();
=====
let x = 10;
console.log(x);
let y = 3;
console.log(y);

function f(){
    console.log('inside f');
}

function bar(){
    console.log('inside bar');
}

function foo(){
    console.log('inside foo');
}

console.log('this is added in master to create conflict with order branch');
>>>>>> master
```

Fix Merge Conflict

- A merge conflict will happen when the exact same line is changed in separate branches.

`<<<< HEAD: code in current checked-out active branch.`

`|||| merged common ancestors: original code.`

`==== until >>>> <branchname>: code in the branch we want to merge.`

- **To solve a merge conflict**

1. Edit the file, remove all lines with indicators (special characters)
2. Edit the file till you're satisfied, save and exit
3. `git add <filename>` - add to staging
4. `git commit -m "commit message"` - commit

```
H:\courses\MSD\cs445\project>git add hello.js
```

```
H:\courses\MSD\cs445\project>git commit -m "Merged master fixed conflict in order branch"  
[order 5229a1b] Merged master fixed conflict in order branch
```

Remote Repository

Git vs Github

- Git is a distributed version control system which means there is not one main repository of information. Each developer has a copy of the repository.
- A remote repository is the same Git local repository but it exists somewhere else.
- Github is a hosting service for version control repositories can be managed from your browser. This is where we will host our remote repositories to be shared with other developers.
- Starting October 1, 2020 all `master` branches will be called `main` branches to avoid any unnecessary references to slavery.

README.md

A **README** is a text file that introduces and explains a project. It contains information that is commonly required to understand what the project is about.

This is where someone who is new to your project will start out.

While READMEs can be written in any text file format, the most common format is Markdown.

Code hosting services such as GitHub will also look for your README and display it along with the list of files and directories in your project.

1. Create the repository, clone it to your PC, and work on it. (Recommended)

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *



TinaXing2012 ▾

/

Repository name *

My-Git-Project



Great repository names are short and memorable. Need inspiration? How about [shiny-engine](#)?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file


This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  `main` as the default branch. Change the default name in your [settings](#).

Create repository

git clone <path-to-repository-to-clone>

Create an identical copy of an existing repository:



- Takes the path to an existing repository
- By default will create a directory with the same name as the repository that's being cloned (can be given a second argument that will be used as the name of the directory)
- Create the new repository inside of the current working directory

```
H:\courses\MSD\cs445\project>cd ..  
H:\courses\MSD\cs445>git clone https://github.com/TinaXing2012/My-Git-Project.git  
Cloning into 'My-Git-Project'...  
remote: Enumerating objects: 3, done.  
remote: Counting objects: 100% (3/3), done.  
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (3/3), done.
```

Go to file

Add file ▾

↓ Code ▾

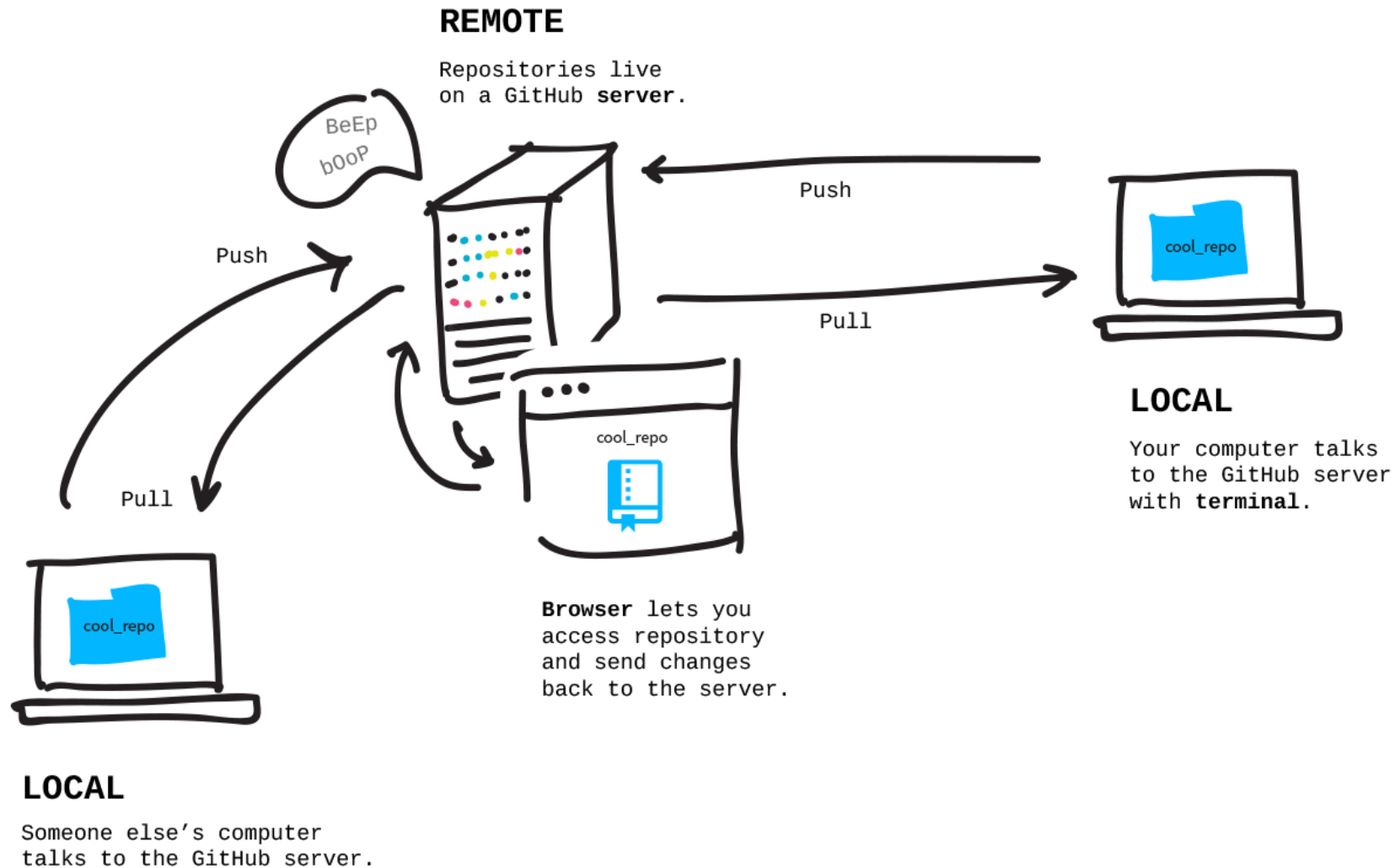
 Clone 

HTTPS SSH GitHub CLI



Use Git or checkout with SVN using the web URL.

Describes how pull & push work



git push <remote-shortname> <branch>

The `git push` command is used to send commits from a local repository to a remote repository.

- A new marker `origin/main` is called a **tracking branch**.
- The tracking branch is telling us that the remote origin has a master branch that points to a certain commit (and includes all of the commits before it).
- Figure out the remote's name (mostly is named origin)
 - `git remote`
- `git push origin main`

```
H:\courses\MSD\cs445\My-Git-Project>git remote
origin

H:\courses\MSD\cs445\My-Git-Project>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

H:\courses\MSD\cs445\My-Git-Project>git push origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/TinaXing2012/My-Git-Project.git
  9a9764e..7704e74  main -> main
```

The screenshot shows the GitHub interface for a repository named 'My-Git-Project' by user 'TinaXing2012'. At the top, there are buttons for 'Go to file', 'Add file', and 'Code'. Below this, a commit history table is displayed:

Commit Hash	Commit Message	Time Ago	Commits
7704e74	Initial commit	10 minutes ago	2
9a9764e	init commit	5 minutes ago	1

Below the table, the 'README.md' file is shown with the content 'My-Git-Project'.

git pull <remote-shortname> <branch>

If there are changes in a remote repository that we would like to include in your local repository, then we want to pull in those changes.

- Adds the commits from the remote branch to the local repository
- Moves the tracking branch
- The local tracking branch (origin/main) is merged into the local branch (main)
- ⚠ Although, changes can be manually added on GitHub, it is not recommended, so don't do it.

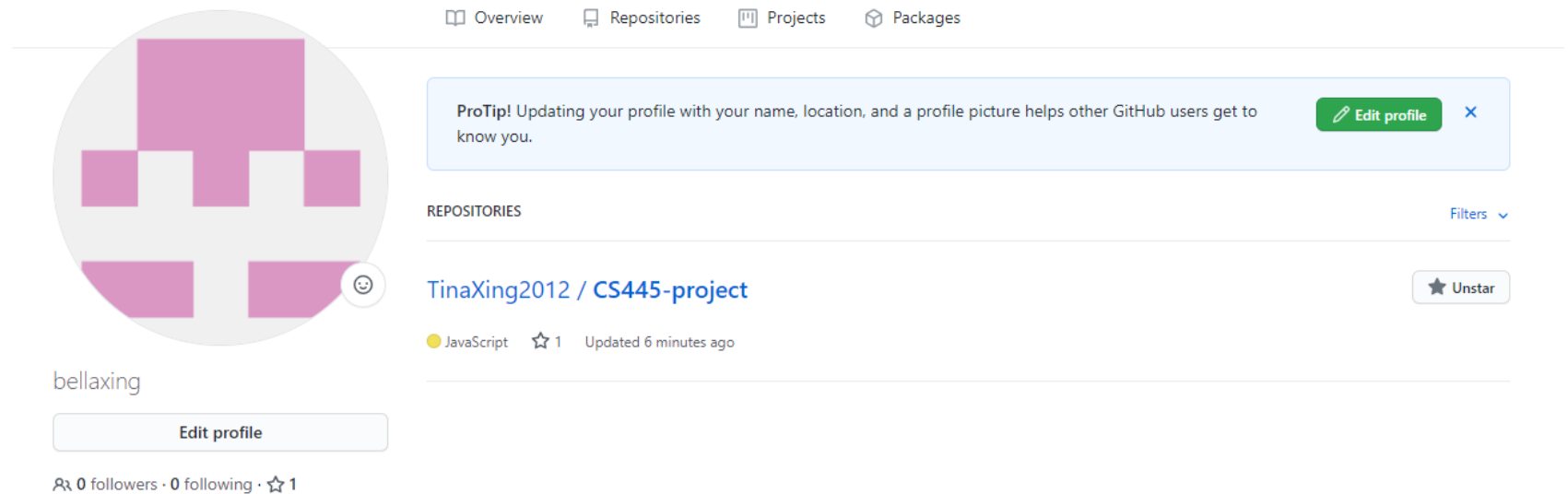
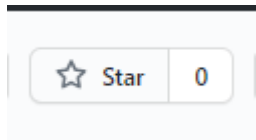
```
H:\courses\MSD\cs445\My-Git-Project>git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 741 bytes | 6.00 KiB/s, done.
From https://github.com/TinaXing2012/My-Git-Project
* branch                main          -> FETCH_HEAD
  b3b51f0..dbb4810      main          -> origin/main
Updating b3b51f0..dbb4810
Fast-forward
 hello.js | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```



**Collaborate with pull
request**

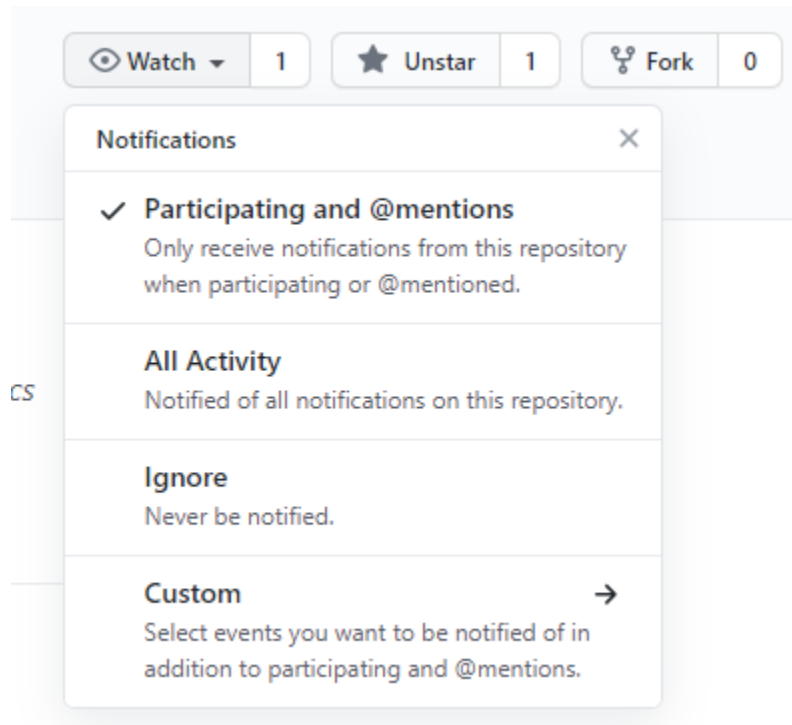
Star a repository

- Users on the GitHub website are able to "star" other people's repositories, thereby saving them in their list of Starred Repos. Some people use "stars" to indicate that they like a project, other people use them as bookmarks so they can follow what's going on with the repo later.

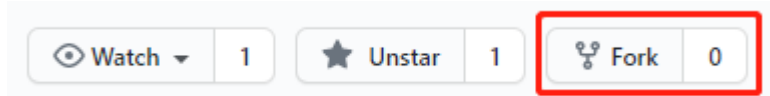


Watch a repository

- If you are **watching** a repository, you will receive notifications for all discussions — project issues, pull requests, comments on commits and any other comments. If you're not watching a repo you'll just receive notification for the discussions you participate in.



Fork a Repository



- A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository.
- You can fetch updates from or submit changes to the original repository with pull requests.
- Forking a repository is similar to copying a repository, with two major differences:
 - You can use a pull request to suggest changes from your user-owned fork to the original repository, also known as the **upstream** repository.
 - You can bring changes from the upstream repository to your local fork by synchronizing your fork with the upstream repository.
- ⚠ **Forking is not done on the command line**

Overview Repositories 1

Popular repositories

[CS445-project](#)

Forked from TinaXing2012/CS445-project

JavaScript

Configuring a remote for a fork

- You must configure a remote that points to the upstream repository in Git to sync changes you make in a fork with the original repository.

1. List the current configured remote repository for your fork.

```
$ git remote -v
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

2. Specify a new remote **upstream** repository that will be synced with the fork.

```
$ git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
```

3. Verify the new upstream repository you've specified for your fork.

```
$ git remote -v
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
> upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
> upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

Syncing a fork

- Sync a fork of a repository to keep it up-to-date with the upstream repository.
1. Fetch the branches and their respective commits from the upstream repository. Commits to `BRANCHNAME` will be stored in the local branch `upstream/BRANCHNAME`.

```
$ git fetch upstream
> remote: Counting objects: 75, done.
> remote: Compressing objects: 100% (53/53), done.
> remote: Total 62 (delta 27), reused 44 (delta 9)
> Unpacking objects: 100% (62/62), done.
> From https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY
> * [new branch]      main      -> upstream/main
```

2. Check out your fork's local default branch - in this case, we use `main`.

```
$ git checkout main
> Switched to branch 'main'
```

3. Merge the changes from the upstream default branch - in this case, `upstream/main` - into your local default branch. This brings your fork's default branch into sync with the upstream repository, without losing your local changes.

```
$ git merge upstream/main
> Updating a422352..5fdff0f
```

Working with your Fork

- Because forking a repository gives you a copy of it in your account, you can clone it down to your computer, make changes to it, and then push those changes back to your forked repository.
- Keep in mind that you push the changes back to your remote repository not to the original remote repository that you forked from.

```
H:\courses\MSD\445\test\CS445-project>git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bellaxing/CS445-project.git
89cd8c0..fad223b  main -> main
```

```
H:\courses\MSD\445\test\CS445-project>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

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

H:\courses\MSD\445\test\CS445-project>git add .

H:\courses\MSD\445\test\CS445-project>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

```
H:\courses\MSD\445\test\CS445-project>git commit -m "added forked message"
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Pull Request

- Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.
- There are 2 main workflows when dealing with pull requests:
 1. Pull Request from a forked repository –study by yourself
 2. Pull Request from a branch within a repository

1. Create a Feature/Topic branch

- When you start working on a new feature/bug fix, you should create a feature/topic branch. A feature/topic branch is normally created off a develop/integration branch. This feature/topic branch can reside in your local machine throughout the entire development lifecycle of the feature.
- You will push this branch to the remote repository whenever you are ready to merge the change set with the develop/integration branch.
- Examples: bugfix-login-form, bugfix-signup

1. Make sure your repository is up to date

- `git pull origin main == git fetch + git merge`

2. Create a branch


- `git checkout -b feature-login`

3. Push to github

- `git push origin feature-login`

2. Create a Pull Request

1. To create a pull request, you must have changes committed to the your new branch
2. Go to the repository page on Github. And click on "**Pull Request**" link in the repo header.
3. Pick the branch you wish to compare and pull request

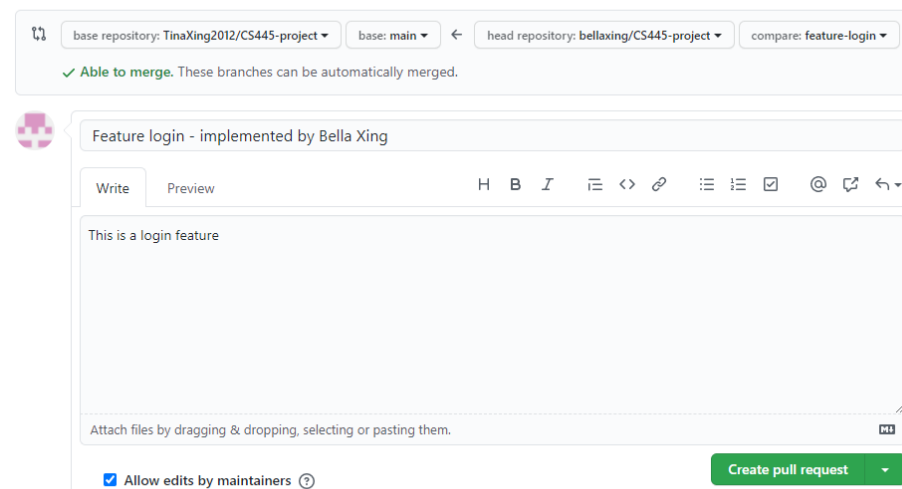
 feature-login had recent pushes 2 minutes ago

Compare & pull request


4. Create a pull request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

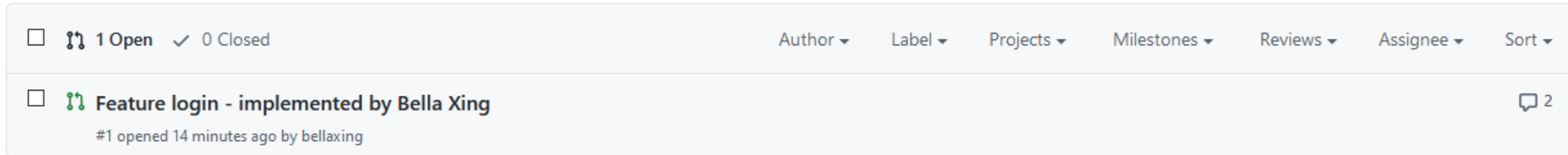


The screenshot shows the GitHub interface for creating a pull request. At the top, there are dropdown menus for 'base repository: TinaXing2012/CS445-project', 'base: main', 'head repository: bellaxing/CS445-project', and 'compare: feature-login'. Below these, a green checkmark indicates 'Able to merge. These branches can be automatically merged.' The main title of the pull request is 'Feature login - implemented by Bella Xing'. There are tabs for 'Write' and 'Preview'. The 'Write' tab is active, showing a text area with the placeholder text 'This is a login feature'. Below the text area, there is a note 'Attach files by dragging & dropping, selecting or pasting them.' and a checkbox labeled 'Allow edits by maintainers' which is checked. A green 'Create pull request' button is at the bottom right.

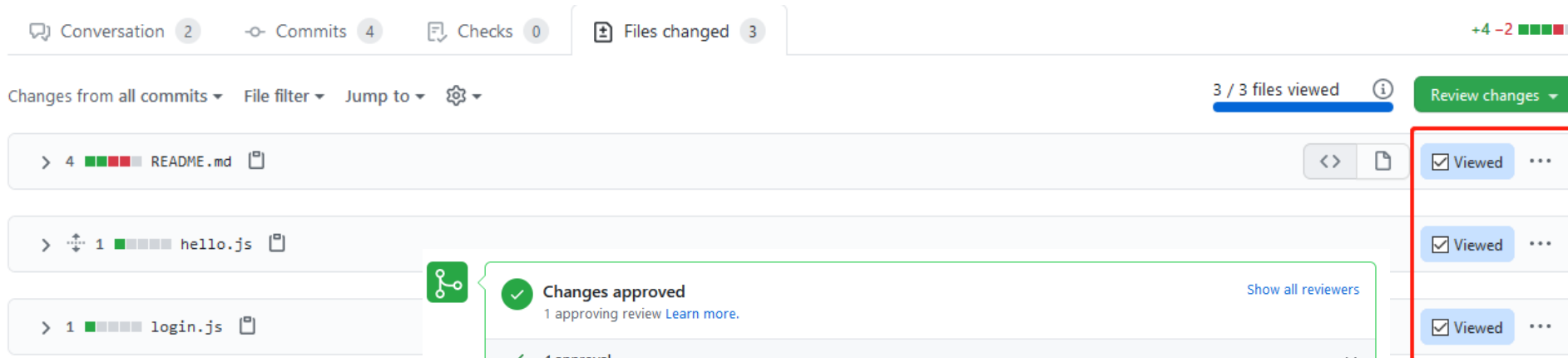
 Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

3. Merge changes in upstream repository

1. As maintainer of the upstream repository, you're able to view all pull requests



2. Go to "File Changed" tabs, view and approve changes



3. Merge pull request