Git Some Help

By: The SRT team

Sign In: (We need your GitHub Username)

Agenda

1

Git Commands

2

Creating Repositories

3

GitHub Features

4

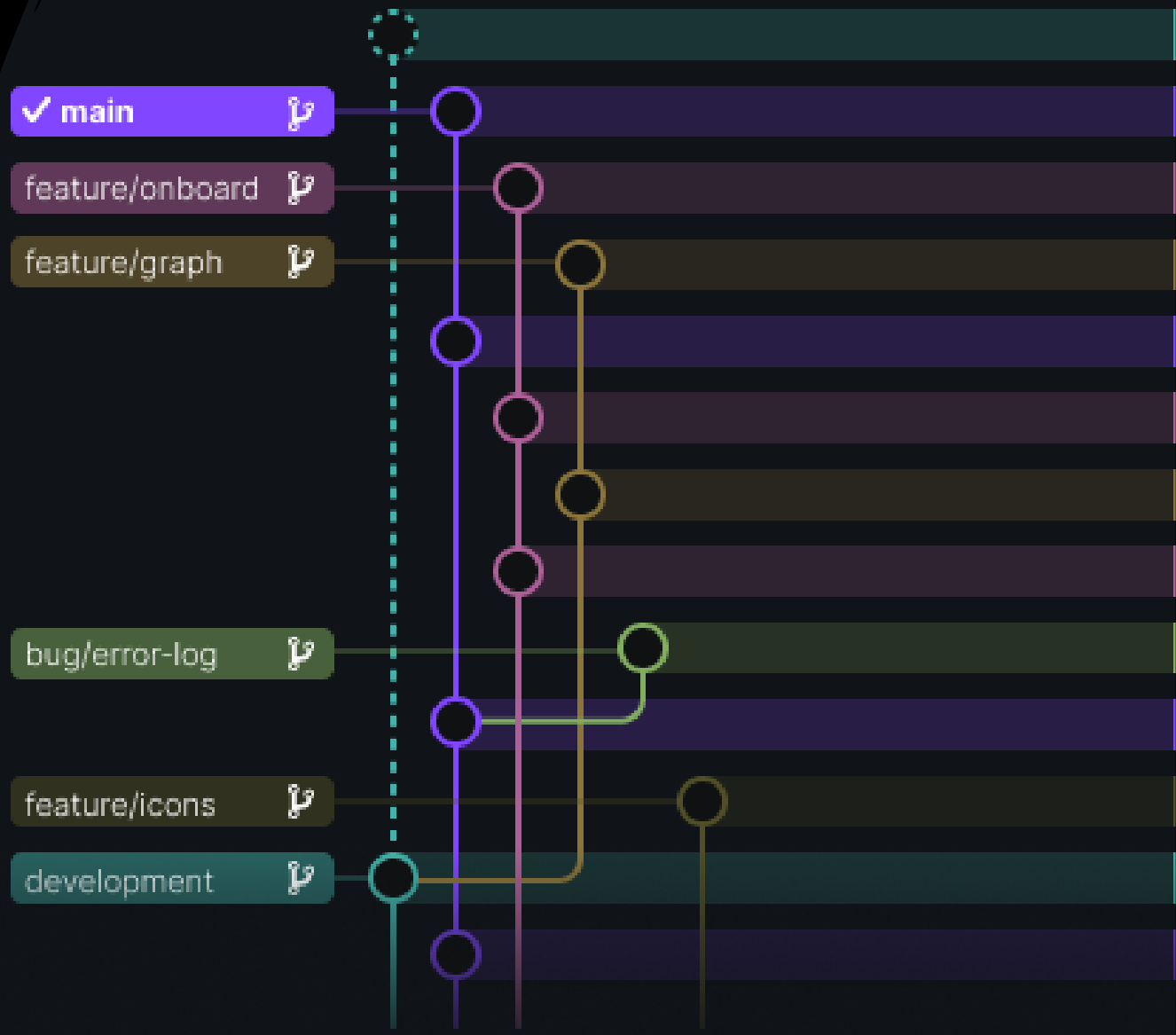
Using Git Effectively

What is git?

- “Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.”
- Namely, a fancy file tracker that allows you to send changes to some outside source. (GitHub, SVN, etc…)

How does it work?

- Git runs on “commits”.
- Each commit is a linked list node. It stores the changes you’ve made, and what the previous commit was.
- Branches are ways to make “multiple” commits at a particular point in your code’s development.



The “Core” Commands

- git add <filename> OR git add –A
  - Tells git to take your current changes to <filename> and add (stage) them onto the next commit. The –A argument says to add ALL files with changes.
- git commit –m “<commit message>”
  - Creates a new commit with your *staged changes* and links the current commit to be the “next” one in the linked list.
    - The linked lists run “backwards” for git branches!
- git push
  - Takes all of your commits you’ve made *locally* and pushes them to whatever remote repository you’ve specified.
    - This will usually mean your GitHub page, but it can be other sources (like Gitter from CSSE132).
    - You can specify a *remote source* to push to with this command, so you can have multiple different repositories you use!

Other “Core” Commands

- git pull
  - Pulls all changes from the remote branch(es) your local branch is set to track
- git clone <repository source>
  - Brings a copy of the remote repository to your local machine, complete with git commit history and branches.
- git status
  - Shows your currently staged changes (what will end up ON the commit), unstaged changes (changes to files made but haven’t been added), and untracked files (newly created files that git doesn’t know about).
  - This is a VERY handy command!
- git restore --staged <filename>
  - Remove a currently staged file from a commit, i.e., do not track those changes with my next commit.
    - Doing this without the --staged flag will restore the file to the current commit’s state

Other Basic Commands

- git log --oneline
  - Gives a log of the entire commit history of your current branch
- git diff <filename>
  - Shows the changes you have made to the given file as compared to the current commit you are on.
- git stash
  - Allows you to put your staged files into a stash which acts like a stack
  - Saves your changes and reverts your local repo back to the latest commit
  - Things can be gotten off the stack by doing **git stash pop**
  - You can see everything in your stash with **git stash list**
- git fetch <remote> <branch>
  - Fetches branches from the remote specified and brings them to your local repo
  - The branch flag specifies a branch in the remote but is not necessary
  - Can also take the --all flag which will fetch all branches from all linked remotes.

Some Advanced Commands

- git rebase -i <commit target>
  - Modifies your git commit history, interactively. If you want one commit to be “applied” before another, you can do it here.
  - Unlike merge, this brings over all of the commits you choose.
  - Only do this if you know what you’re doing!
- git merge <branch name>
  - Apply the changes from commits from *the specified branch* on top of the commits on your current branch. It will show up as ONE big commit afterwards.
  - Note: ***THIS IS WHERE MERGE CONFLICTS APPEAR!***
  - This is safer to use if you know what changes you want to apply.
- git remote
  - Has sub commands of git remote add <name> <repository>, git remote remove <name>
  - This is how you can specify OTHER remote repositories you’d like to interact with. i.e. you can run git push <remote name>. Pushes to the ORIGIN remote by default.
  - Base command simply lists your remotes you have setup.

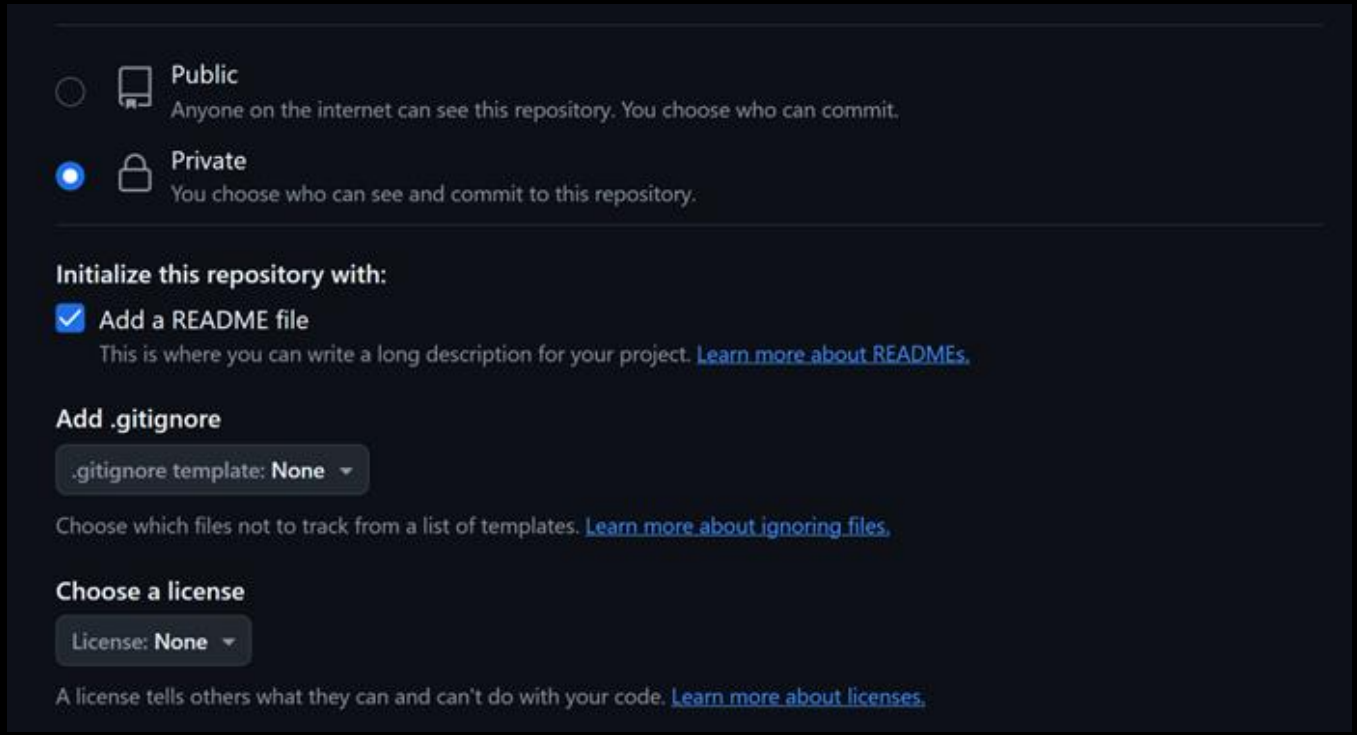
Other Advanced Commands

- git reset --soft <commit-hash> or git reset --soft HEAD~1
  - Allows you to rollback your local repo to the state of a previous commit
  - The soft option preserves changes, also hard option but this does not preserve changes
  - use git log to find the commit-hash
  - HEAD~1 goes back one commit in your working branch
- git revert <commit-hash> or git revert HEAD --no-edit
  - Used to rollback a previous commit without moving the HEAD
  - Basically creates a new commit that undoes the changes of a previous commit
- git commit --amend
  - Allows you to add more files/changes to the most recent commit and update the commit message
  - Completely replaces previous commit

Creating Repositories

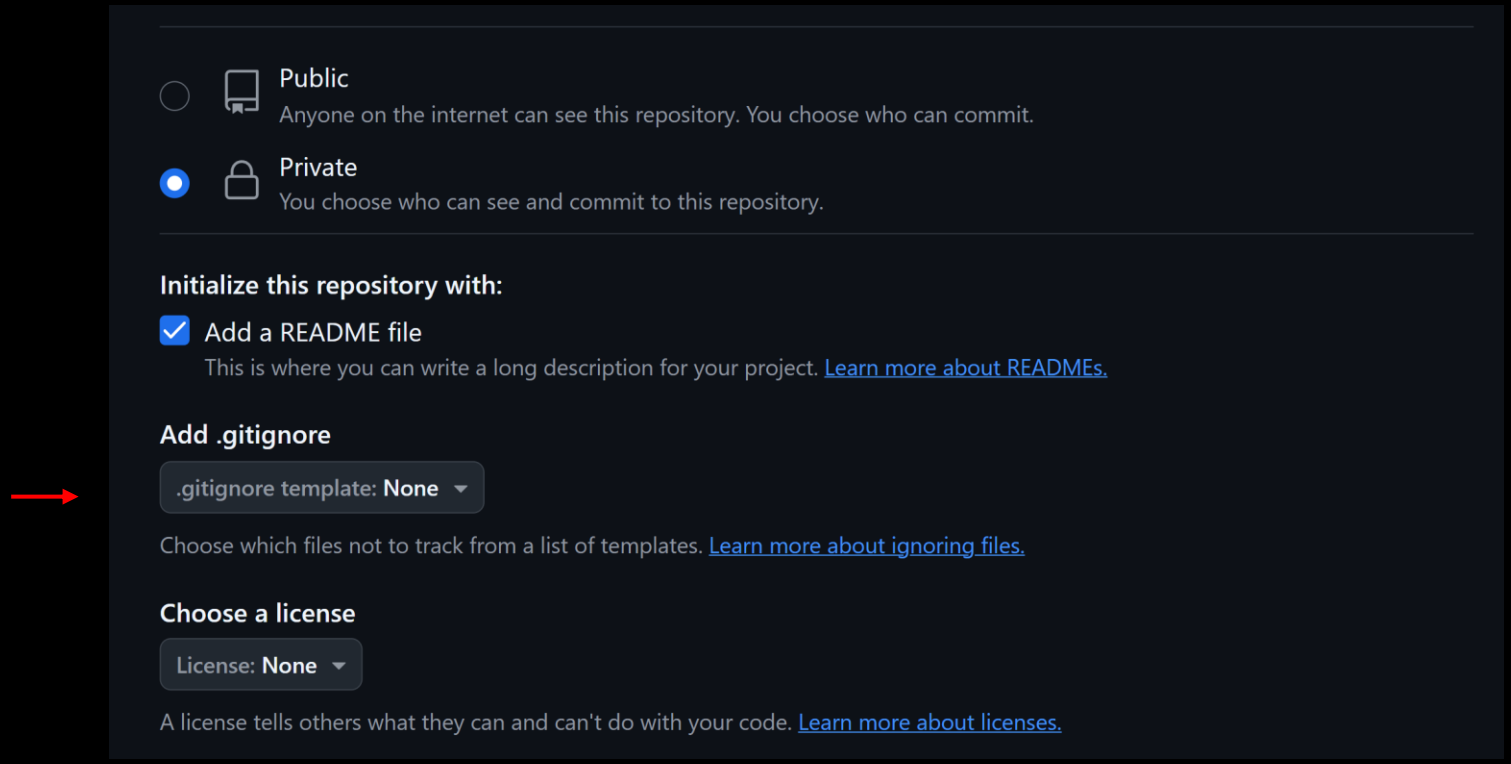
Two main ways to create a repository:

- GitHub (preferred)
- Command line
  - Make sure to follow the hints!
  - This repository will be empty! No initial commit -- you can't make branches or anything without one.




```
wynessgp@RHIT-R912GH3C:~/git-some-help-example$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/wynessgp/git-some-help-example/.git/
```

Using GitHub



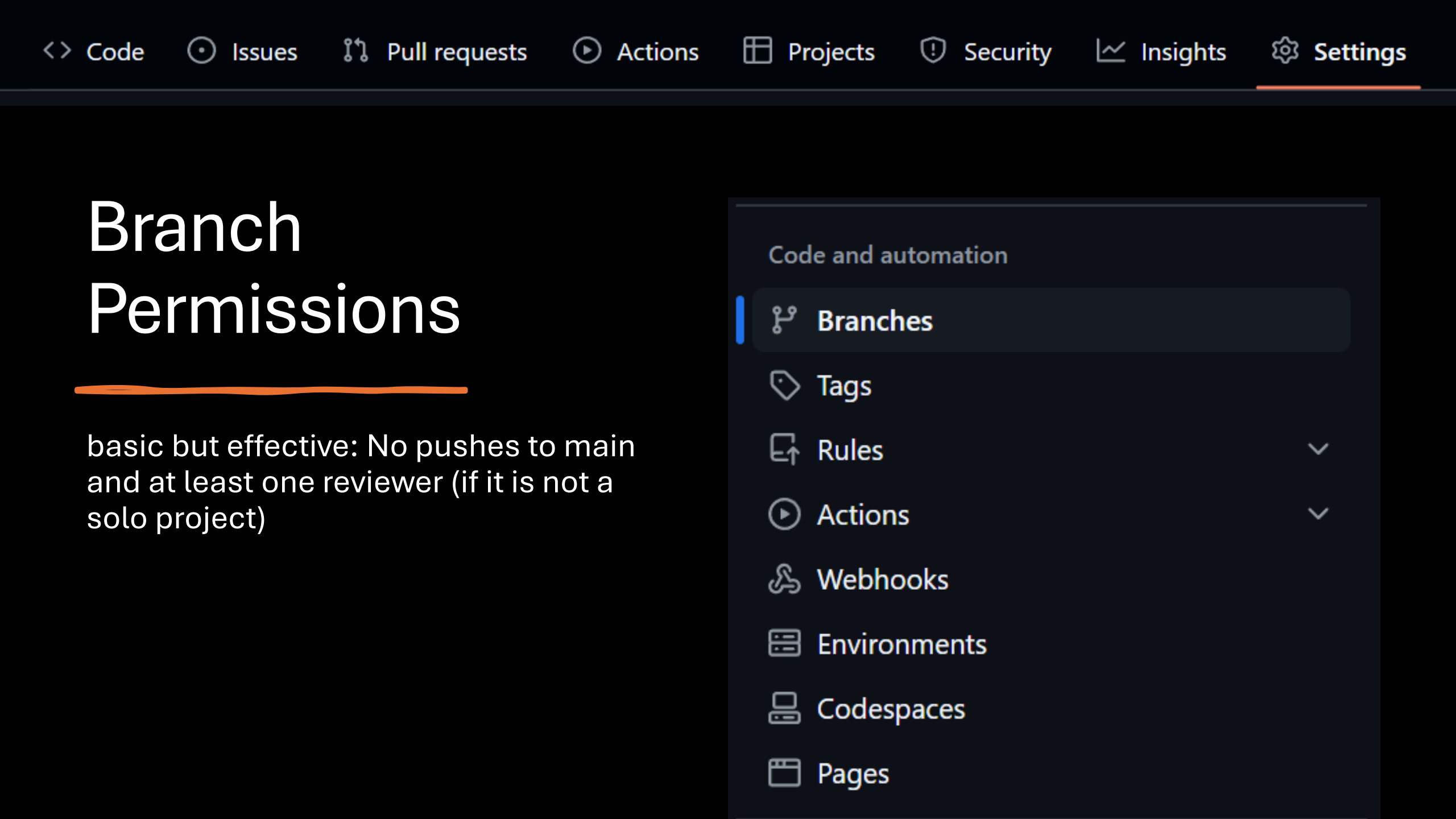
.gitignore Patterns

- GitHub Provides lots of documentation along with premade .gitignore files here: .gitignore documentation
- The following are the basic rules of .gitignores (there are many more)

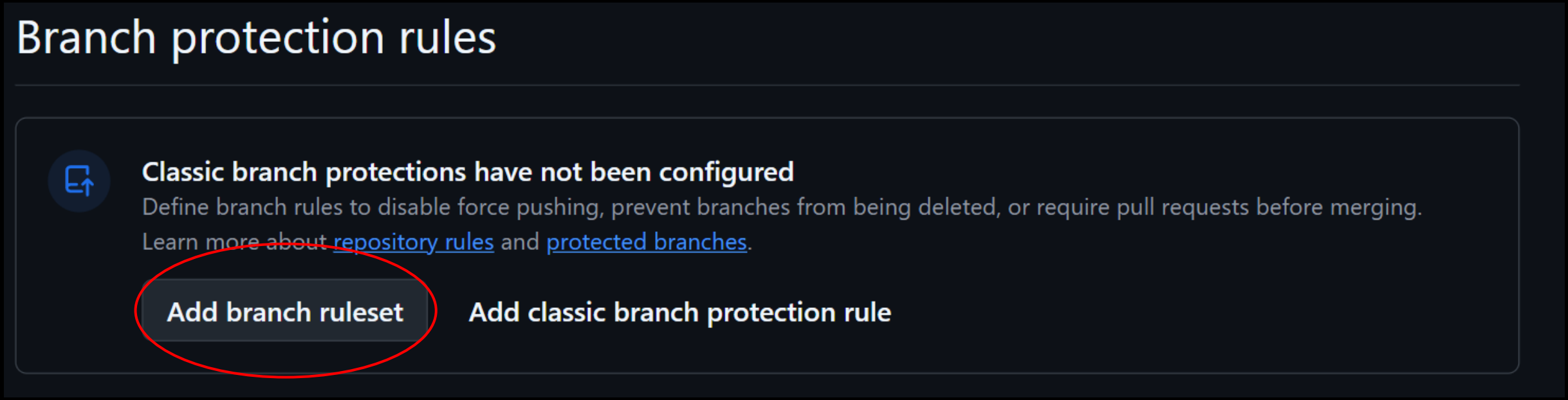
Pattern	Example(s)	Explanation
logs	logs logs/log1.txt build/logs/errors.txt Important/logs	a keyword will exclude **all** files **and** directories with that name
*.bin	main.bin graph/surfing.bin	An asterisk before a file extension excludes all files with the matching file extension within the repo.
!special.bin	**will not exclude:** special.bin	An exclamation mark before a specific file will include the specified file. This only works if the file is excluded from a pattern match that isn't a directory exclusion. (E.g. *.bin)
logs/	logs/log1.txt logs/runtime/input.txt build/logs/errors.txt	A directory excludes all directories with that name

Example .gitignore: Java

```
# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
replay_pid*
```


Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
Branch Permissions
basic but effective: No pushes to main and at least one reviewer (if it is not a solo project)
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages

Branch Protection Rules



Main Protection

Ruleset Name *

Main Protections

Enforcement status

Active

Branch targeting criteria

Add target

Branch targeting has not been configured

Add target

Include default branch

Include all branches

Target by inclusion or exclusion pattern

Include by pattern

Exclude by pattern

Include by pattern

Branches that match the matching pattern will be targeted by this ruleset.

Branch naming pattern

main

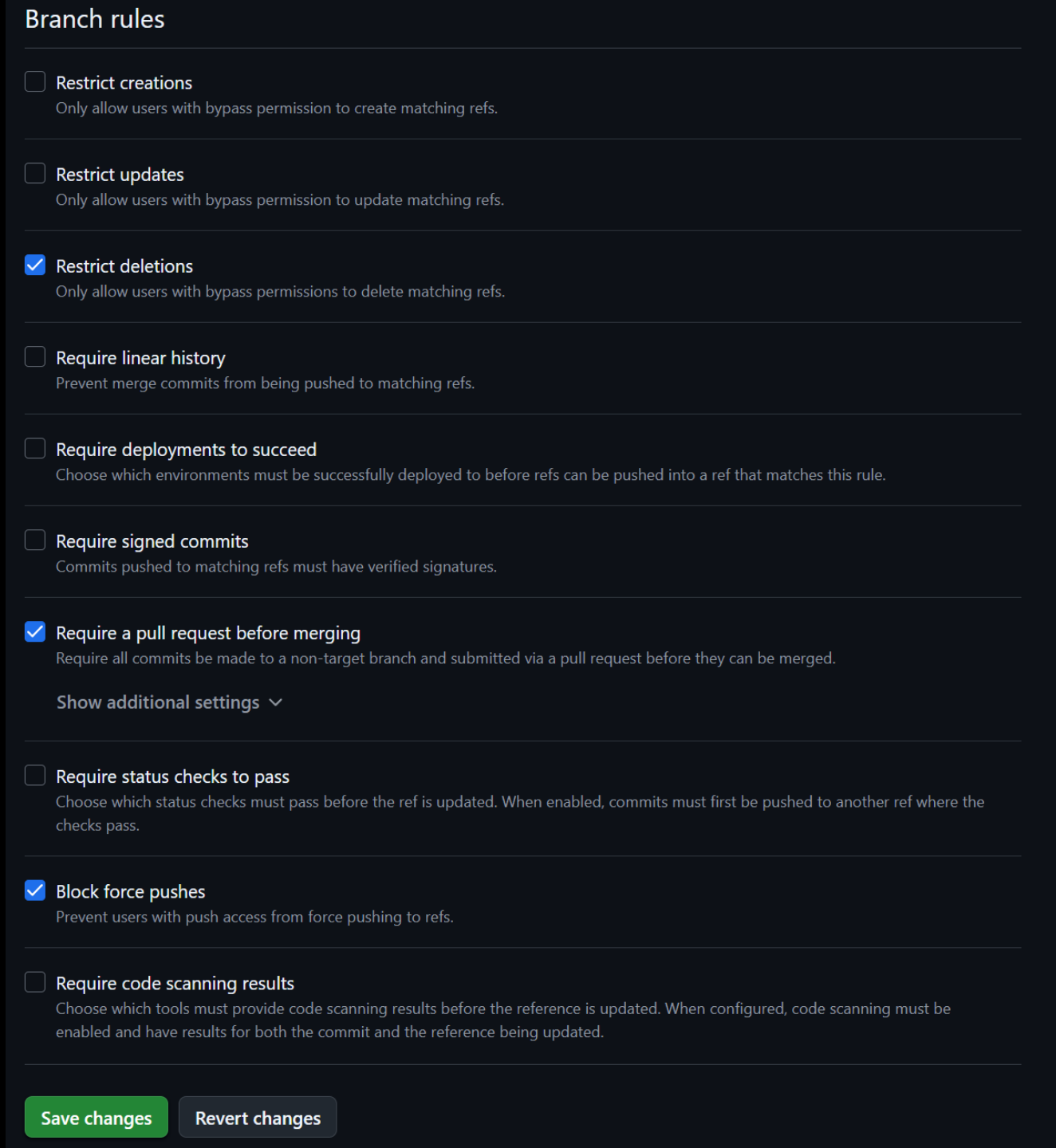
Example patterns: "main", "releases/**/*", "users/**/*". Learn more about fnmatch.

Cancel

Add Inclusion pattern

Branch Rules

- Restrict Deletions:
  - Makes it so that only people with bypass permissions can delete the main branch
  - Additional setting has option for required reviewers
- Require a pull request:
  - Restricts committing to the main branch directly
- Block force pushes:
  - Removes the ability to force a push



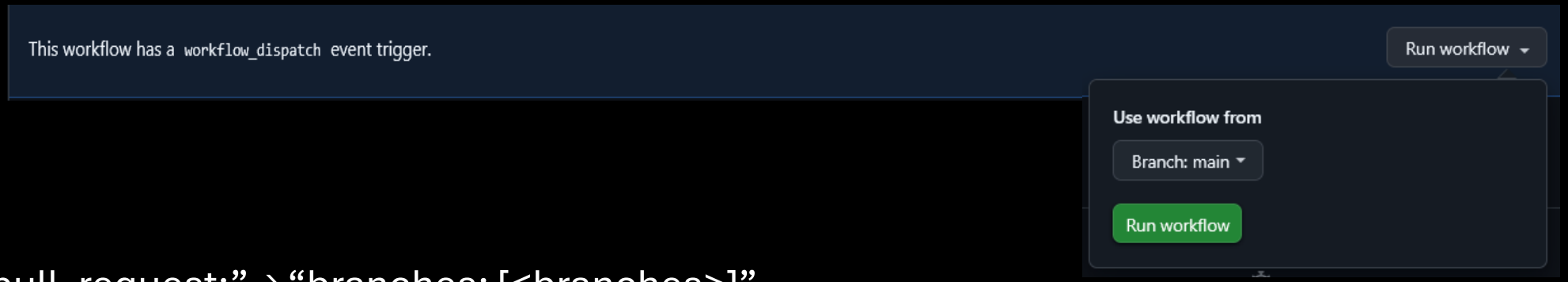
GitHub Actions

- Very powerful tools that allow you to specify scripts (usually bash) that should be run upon certain Git actions being done on your repository.
- Runs a workflow file that you specify, under .github/workflows
- These must be .yaml files to be recognized by GitHub.

```yaml
name: Run Gradle on PRs
on:
  pull_request:
    branches: [ "main" ]
jobs:
  gradle:
    strategy:
      matrix:
        os: [macos-latest, windows-latest]
    runs-on: ${{ matrix.os }}
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 11
      uses: actions/setup-java@v3
      with:
        distribution: temurin
        java-version: 11

    - name: Run chmod to make gradlew executable
      run: chmod +x ./gradlew

    - name: Setup Gradle
      uses: gradle/gradle-build-action@v2
      with:
        gradle-version: 7.4

    - name: Execute Gradle build
      run: ./gradlew build
```

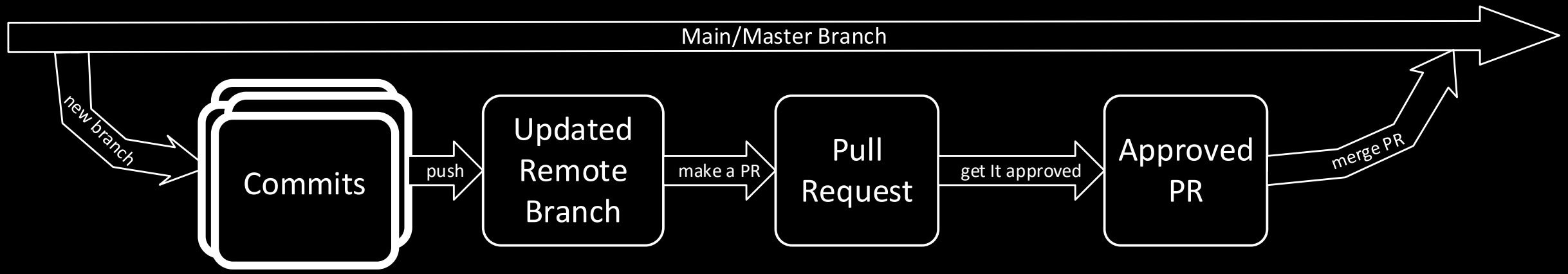
Basic Actions Rules

- These will go under the “on:” section of your workflow
- “push:” → “branches:” → **“-<branch name>$^{+}$”**
  - Whenever you push to the branches that are specified, the workflow will be run on the LATEST commit.
- “workflow_dispatch:”
  - This enables you to trigger a workflow run MANUALLY from the GitHub actions interface.



- “pull_request:” → “branches: [<branches>]”
  - Run the workflow whenever a pull request is opened with a target branch specified.
- And many more… GitHub documentation

Git Workflow



Making a New Branch

- git branch <name>
  - Creates a new local branch
  - Then use git checkout to get onto the branch
- git checkout –b <name>
  - Creates a new local branch and moves you to that branch
- git branch
  - Lists all local branches of your repo
  - -a → lists all the remote branches you have fetched as well

Committing and Pushing

1. **git status** → check that all the correct files were changed
2. **git add <files/paths>$^{+}$** → can be multiple different files or paths at once
3. **git status** → check that all the files you wanted were added
4. **git commit –m "<message>"** → directly add your message with –m so it doesn't prompt you
5. **git pull** → check for remote changes
6. **git push** → after creating a new branch it may say that a remote is not setup, copy the command it gives you
7. ↺

testing-b had recent pushes 1 minute ago

Compare & pull request

Filters | is:pr is:open | Labels 9 | Milestones 0 | New pull request

Making a Pull Request (PR)

Add a title

Testing Branch 2

Add a description

Write | Preview

Add your description here...

Markdown is supported | Paste, drop, or click to add files

Create pull request

Reviewers

No reviews—at least 1 approving review is required.

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

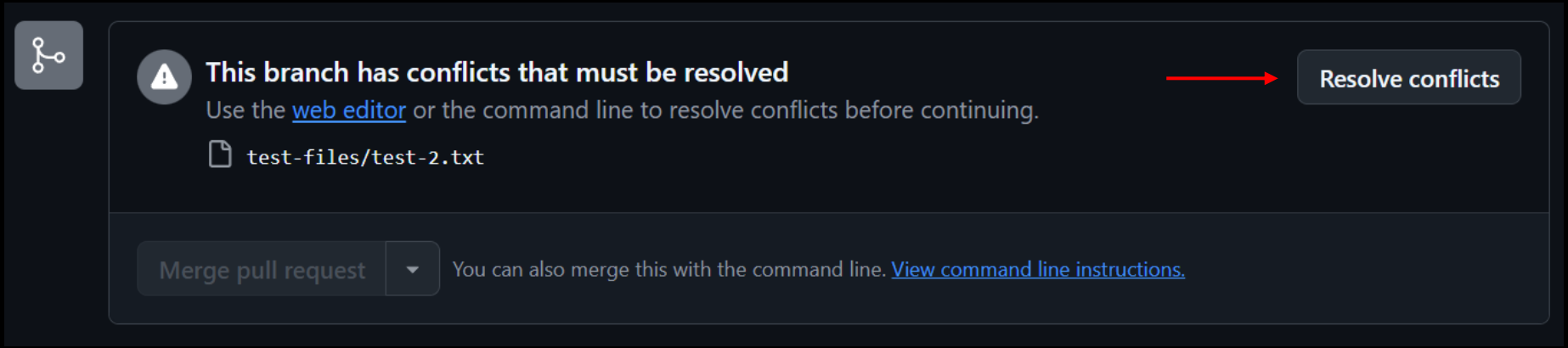
Development

Use Closing keywords in the description to automatically close issues

Helpful resources

Handling A Merge Conflict On Your PR

Using the GitHub conflict resolver:



- <<<<< is the start of your changes
- >>>>> is the end of the remote changes
- ===== is the split between the two

```
Here is a second test file. It is less important. Changing this file.
<<<<<<< merger-confl-3
I wanted to (more error creation) add a few more lines to this file. This is to make the merge conflict
=======
I wanted to add a (also trying to make error) few more lines to this file. This file is getting modified for an error.
>>>>>>> main
These should do.
```

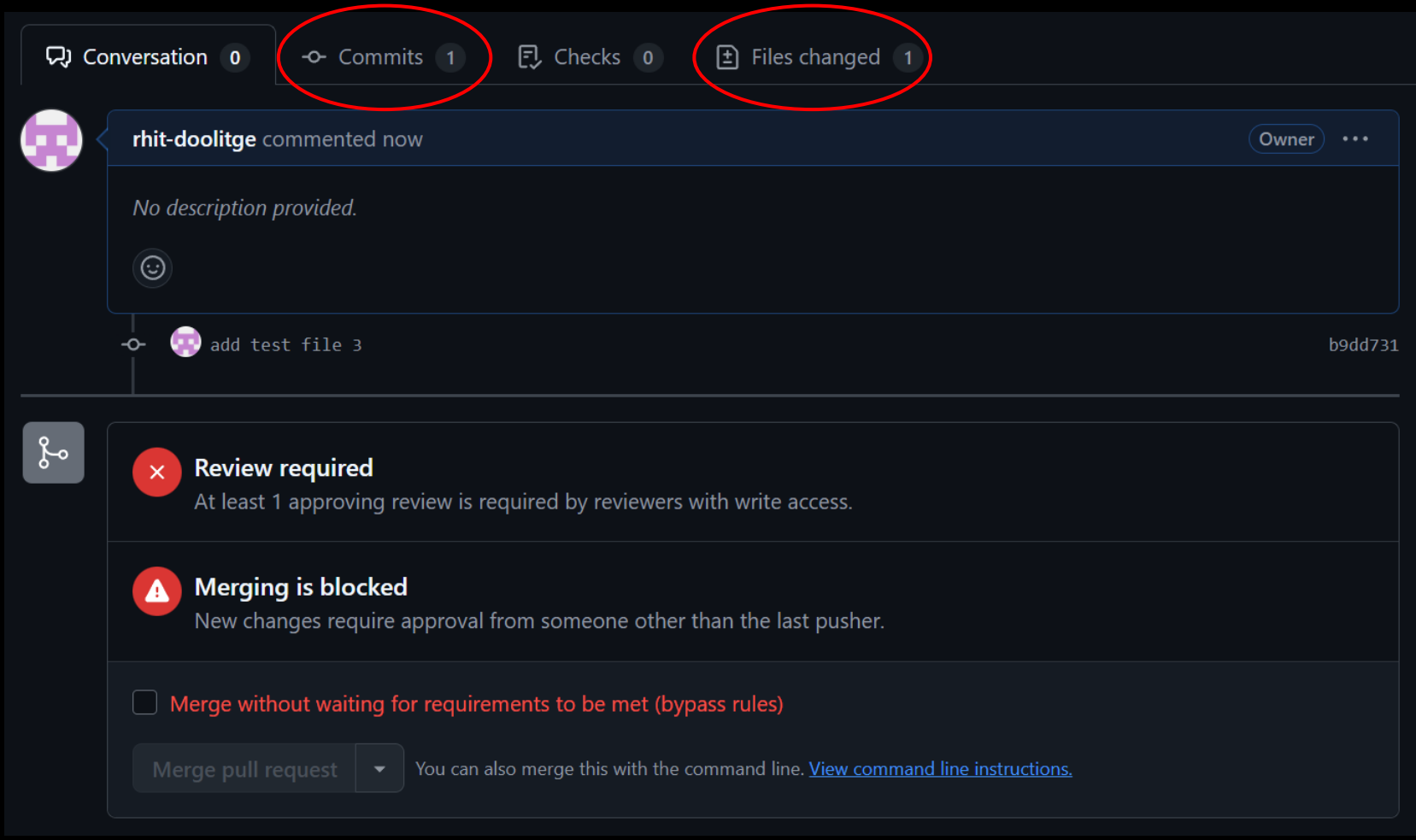
Handling A Merge Conflict On Your PR

- Merge main into your branch and resolve the conflict in your IDE
  - This allows you to ensure that everything is working before merging everything into main
  - The better option if the conflicts are very complex

1. git checkout <branch-with-conflict>
2. git merge origin/main
3. Handle conflict either in IDE or terminal
4. Check everything works by running all the tests in your project
5. Commit and push merge to branch

Code Review

- Commits
  - allows you to see all commits on the branch.
- Files changed
  - Gives you an overview of all the changes to all the files in the branch
  - This is where you go to review someone else's code



Reviewing a Team Members Code

You are reviewing the code not the person who wrote it.

Filter changed files

- test-files
  - test-1.txt
  - test-2.txt
  - test-3.txt

4 test-files/test-1.txt

```
@@ -1 +1,3 @@
- Here is a test file. This is a very important file.
+ Here is a test file. This is a very important file.
+ Maybe some more in here as well.
+ Hopefully this goes well.
```

4 test-files/test-2.txt

```
@@ -1 +1,3 @@
- Here is a second test file. It is less important. Changing this file.
+ Here is a second test file. It is less important. Changing this file.
+ I wanted to add a few more lines to this file.
+ These should do.
```

1 test-files/test-3.txt

```
@@ -0,0 +1 @@
+ here is another random test file.
```

Conversations

Diff view

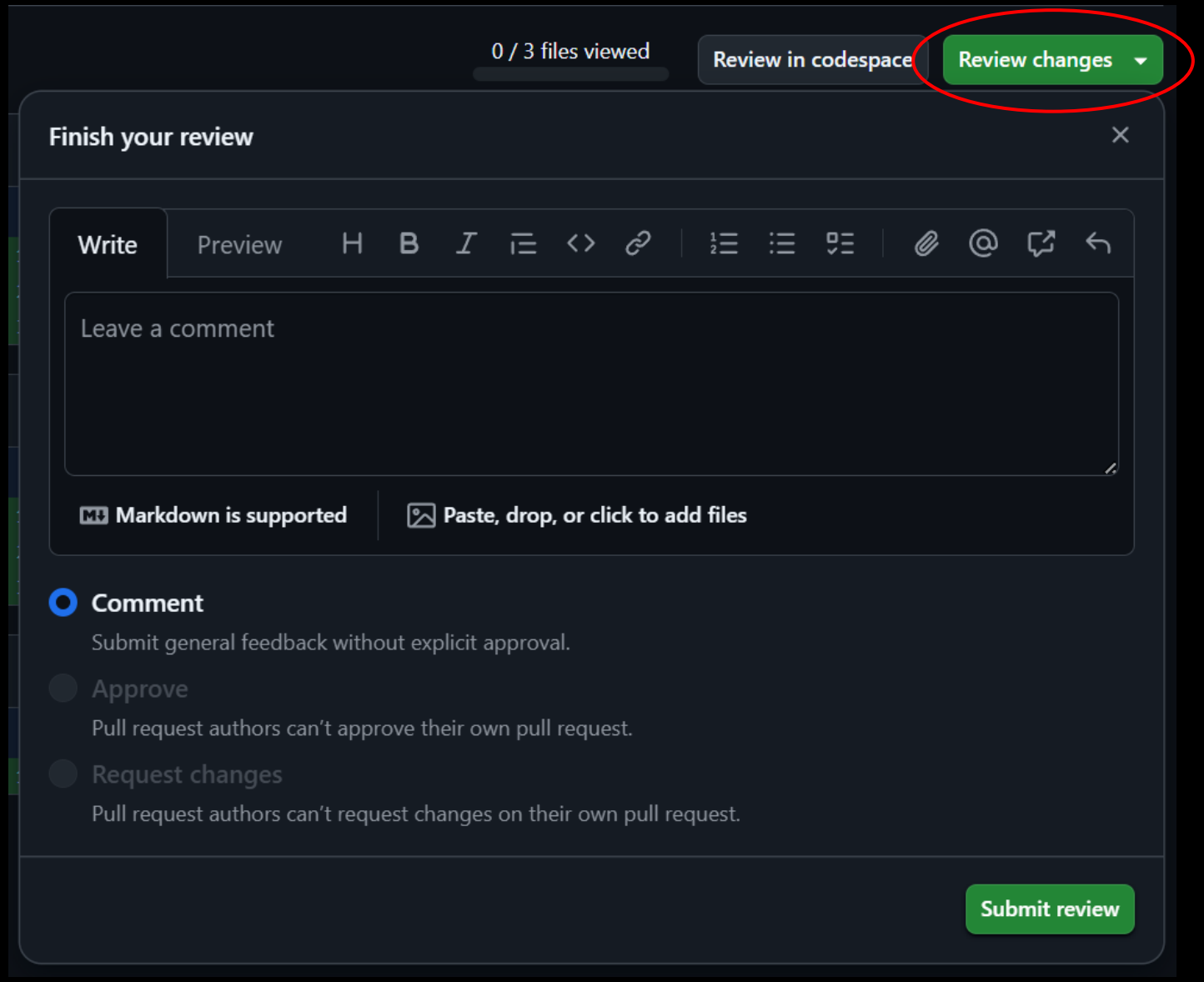
Unified

Split

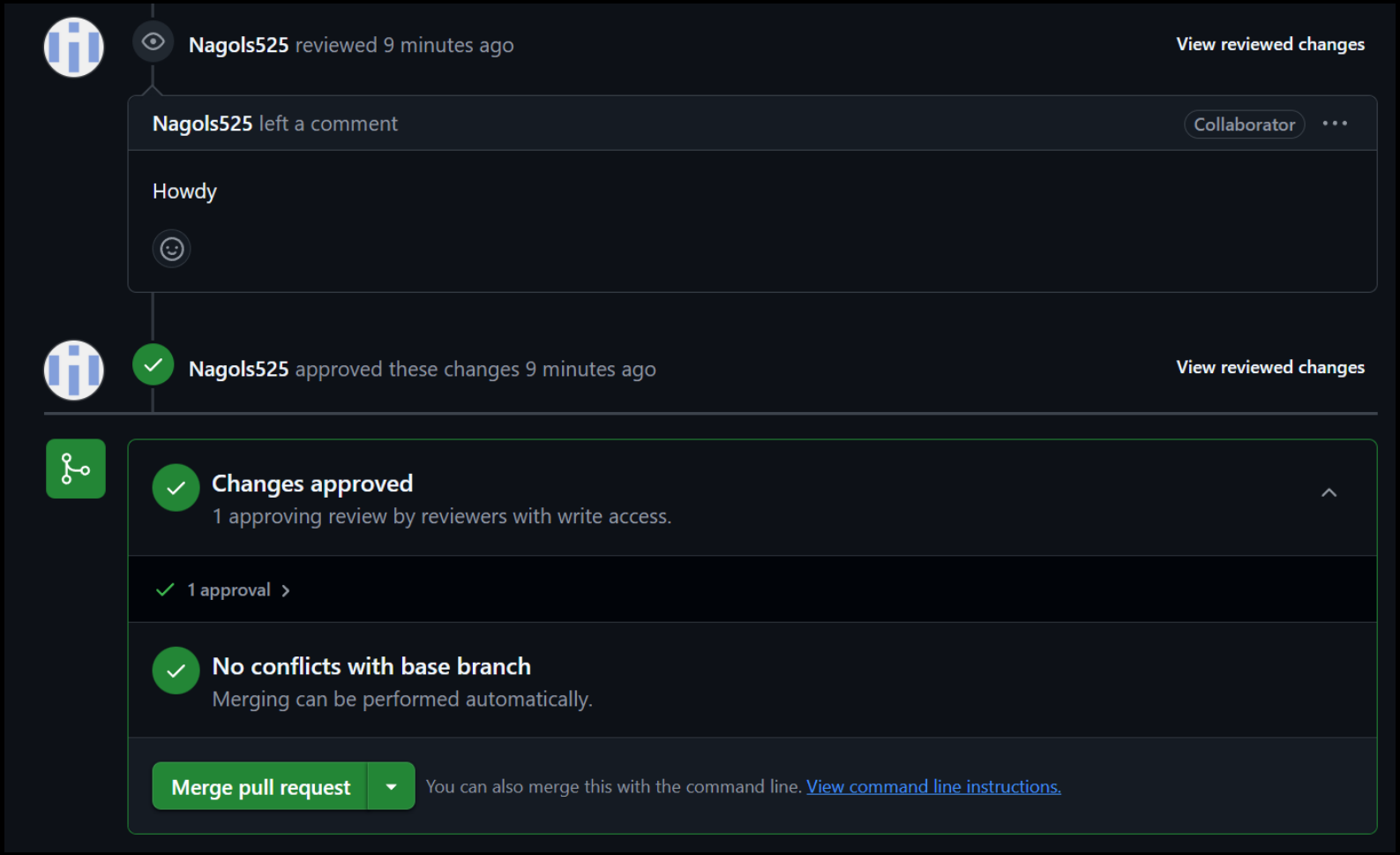
Hide whitespace

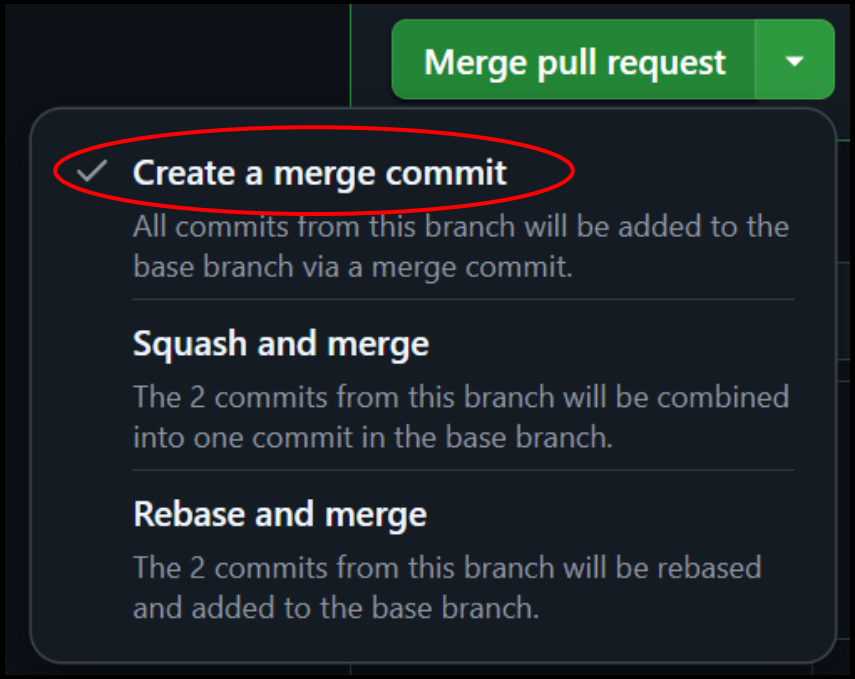
Apply and reload

Writing Your Review

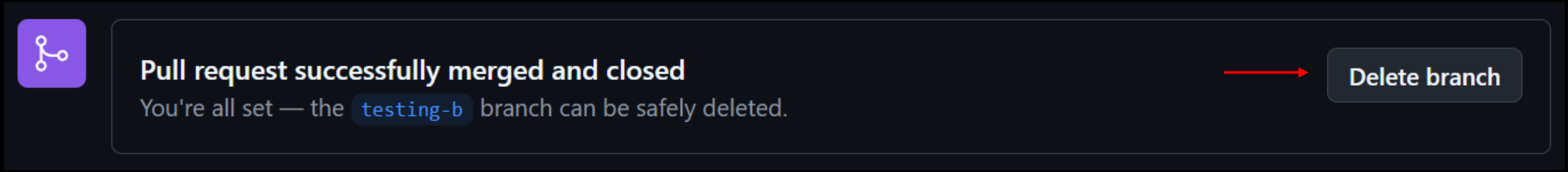


Cleaning up the Branch



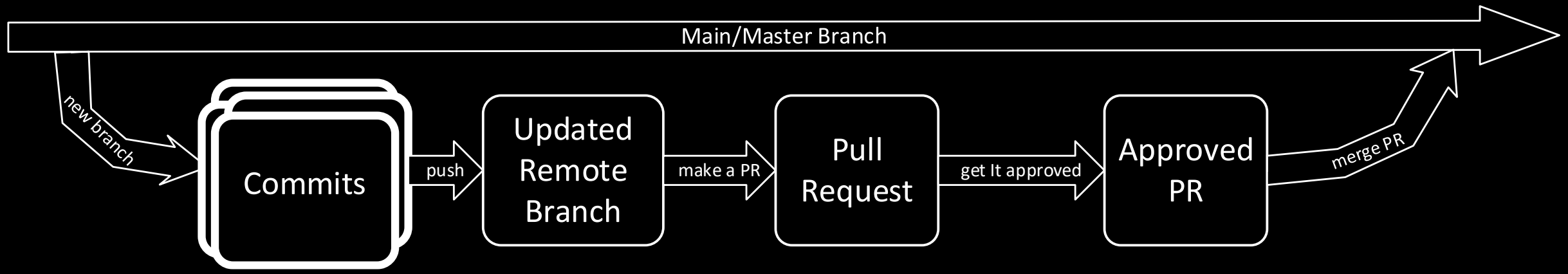


Delete the Branch



Now You Do It

https://github.com/rhit-doolitge/Git-and-GitHub-basics



Other Tips

- Don’t use powerpoint