



Git - 0 to Pro Reference

By: supersimple.dev

Tutorial link: <https://www.youtube.com/watch?v=hrTQipWp6co>

Command Line (Terminal / PowerShell)

ls

cd ~/Desktop/folder

List the files and folders in the **current** folder
Change the folder that the command line is running in

Note: git commands must be run inside the folder that contains all the code.

Creating Commits



In Git, version = commit

Version history = commit history

git init
git status

Git will start tracking all changes in the current folder
Show all changes since the previous commit

git add <file|folder>
git add file
git add folder/
git add .

Pick changes to go into next commit
Pick individual file
Pick all files inside a folder (and subfolders)
Pick all files (in folder command line is running in)

git commit -m "message"
git commit -m "message" --amend

Creates a commit with a message attached
Update previous commit instead of creating new one

git log
git log --all
git log --all --graph

View the commit history
Show all commits (not just current branch)
Show branching visually in the command line

Configure Name & Email for Commits

git config --global user.name "Your Name"
git config --global user.email "email@example.com"

Working Area = contains changes start in the working area

Staging Area = contains changes that will go into the next commit

```
git add .                                working => staging  
git commit -m "message"                  staging => commit history
```

```
git reset file  
git reset folder/  
git reset .
```

`git checkout -- <file|folder>` working => remove the changes

```
git checkout -- file  
git checkout -- folder/  
git checkout -- .
```



[Viewing Previous Commits](#)

```
git checkout <commit_hash|branch_name>
```

View a previous commit

commit 81491250a2a940babba4a3f69bec7aa2c87b782a (master)

Author: Simon Bao <simon@supersimple.dev>
Date: Sat Feb 20 07:19:11 2021 +0800

Version 3

commit 4fb1b33d86a825c517b0376ebd950111f98d0ada

Author: Simon Bao <simon@supersimple.dev>
Date: Sat Feb 20 07:18:53 2021 +0800

Version 2

commit 400e1ba797f732c94e290774aacfd4738c864db8 (HEAD)

Author: supersimpledev <supersimpledev@Simons-MacBook-Pro.local>
Date: Sat Feb 20 05:49:00 2021 +0800

Version 1

master = branch name

1. You can `git checkout` branch
 2. Always points to latest commit on the branch.

HEAD = indicates which commit you are currently viewing

Restoring to a Previous Commit

`git checkout <hash|branch> <file|folder>` Restore the contents of files back to a previous commit

`git checkout <hash|branch> file`

```
git checkout <hash|branch> folder/
```

```
git checkout <hash|branch> .
```

Restore the contents of files back to a previous commit

Restore a file

Restore all files in folder (& subfolders)

Restore all files in project



Other Features of Git

```
git config --global alias.shortcut <command>    Creates an alias (a shortcut)
      git config --global alias.s "status"          git s = git status
```

.gitignore	Tell git which files/folders it SHOULD NOT track
rm -rf .git	Remove git from project

GitHub

Repository = a folder containing code where any changes to the code are tracked by git.
(To create a repository, we create a new folder on our computer, and then run `git init`)

GitHub = a service that lets us save our git repositories online. It also helps us:

- backup our code in case we delete it on our computer
- see the history of our code changes more easily
- alternatives include Bitbucket and GitLab

Local repository = a git repository saved on our computer

Remote repository = a git repository saved online (for example on GitHub)

Uploading Code to GitHub

<code>git remote add <remote_name> <url></code>	Link a local repository to a remote repository and give a name for this link
---	--

<code>git remote add origin https://github.com/SuperSimpleDev/repository1</code>	
--	--

^
The above command links a local repository to a GitHub repository (located at the url `https://github.com/SuperSimpleDev/repository1`) and gives it a name "origin"

<code>git remote</code>	List all remote repositories that are linked
<code>git remote -v</code>	List all remote repositories (but with more detail)

<code>git remote remove <remote_name></code>	Removes a link to a remote repository
<code>git remote remove origin</code>	Removes the link to the remote repository named "origin"

<code>git config --global credential.username <username></code>	Configure your GitHub username so you can get access to your Github repository
---	--

<code>git push <remote_name> <branch></code>	Upload a branch of your git version history to your remote repository
--	---

<code>git branch</code>	Shows a list of available branches
<code>git log --all --graph</code>	Shows the branches visually in the history

<code>git push origin main</code>	Upload the branch "main" to the remote repository named "origin"
<code>git push <remote_name> <branch> --set-upstream</code>	Sets up a shortcut for this branch and remote repository. Next time you are on the <code>main</code> branch and you run <code>git push</code> , it will automatically push the <code>main</code> branch to <code>origin</code>
<code>git push <remote_name> <branch> -f</code>	Force-push the branch to the remote repository (it will overwrite what's on the remote repository)

Downloading Code from GitHub

<code>git clone <url></code>	Download a remote repository from a url <code>git clone https://github.com/SuperSimpleDev/repository1</code>
<code>git clone <url> <folder_name></code>	Download the repository and give it a different folder name
<code>git fetch</code>	Updates all remote tracking branches. Remote tracking branches (like <code>origin/main</code>) show what the branch looks like in the remote repository
<code>git pull <remote_name> <branch></code>	Update the local branch with any updates from the remote repository (on GitHub)
<code>git pull origin main</code>	Downloads any new commits from the <code>main</code> branch on <code>origin</code> , and updates the local <code>main</code> branch with those new commits
<code>git pull origin main --set-upstream</code>	Sets up a shortcut so that the next time you are on the <code>main</code> branch and run <code>git pull</code> , it will automatically <code>git pull origin main</code>

Branching

Branching = create a copy of the version history that we can work on without affecting the original version history. This lets us work on multiple things (features + fixes) at the same time.

<code>git branch <branch_name></code>	Creates a new branch
<code>git branch feature1</code>	Create a new branch named <code>feature1</code>
<code>git checkout <branch_name></code>	Switch to a different branch and start working on that branch
<code>git checkout feature1</code>	Switch to the <code>feature1</code> branch. New commits will now be added to the <code>feature1</code> branch

```

* commit 9bb22ff9063a3e1134e5cea3fb289df492868cef (HEAD -> feature1, master)
| Author: Simon Bao <simon@supersimple.dev>
| Date:   Sat Jun 5 09:27:25 2021 +0800
|
|     version3
|
* commit 8464f5b7dc7d0271f8a00f9dc0b707b4ecc64301
| Author: Simon Bao <simon@supersimple.dev>
| Date:   Sat Jun 5 09:27:16 2021 +0800
|
|     version2
|
* commit 285addbf98ee4d450c226a410acf38ab16ba7696
  Author: Simon Bao <simon@supersimple.dev>
  Date:   Sat Jun 5 09:27:01 2021 +0800

    version1

```

HEAD = points to which branch we are currently working on

HEAD -> feature1 = we are currently working on the **feature1** branch. Any new commits will be added to the **feature1** branch

`git branch -D <branch_name>`
`git branch -D feature1`

Deletes a branch
 Deletes the **feature1** branch

Merging

`git merge <branch_name> -m "message"`

Merge the current branch (indicated by **HEAD ->**) with another branch (**<branch_name>**). Saves the result of the merge as a commit on the current branch

`git checkout main`
`git merge feature1 -m "message"`

1. First switch to the **main** branch
 2. Then merge the **main** branch with the **feature1** branch. The result of the merge will be added to **main** as a commit (a "merge commit")

Merge Conflicts

```

<<<<< HEAD
code1
=====
code2
>>>>> branch

```

If there is a merge conflict (git doesn't know what the final code should be), it will add this in your code.

(This is just for your convenience, the <<<<< and >>>>> don't have special meaning)

```
<<<<< HEAD
...
      <-- Code in the current branch (indicated by HEAD ->)
=====
...
      <-- Code in the branch that is being merged into HEAD
>>>>> branch
```

To resolve a merge conflict:

1. Delete all the extra code and just leave the final code that you want.

```
<<<<< HEAD
code1
=====
code2
=> code2
>>>>> branch
```

2. If there are conflicts in multiple places in your code, repeat step 1 for all those places.

3. Create a commit.

```
git add .
git commit -m "message"
```

Feature Branch Workflow

A popular process that companies use when adding new features to their software.

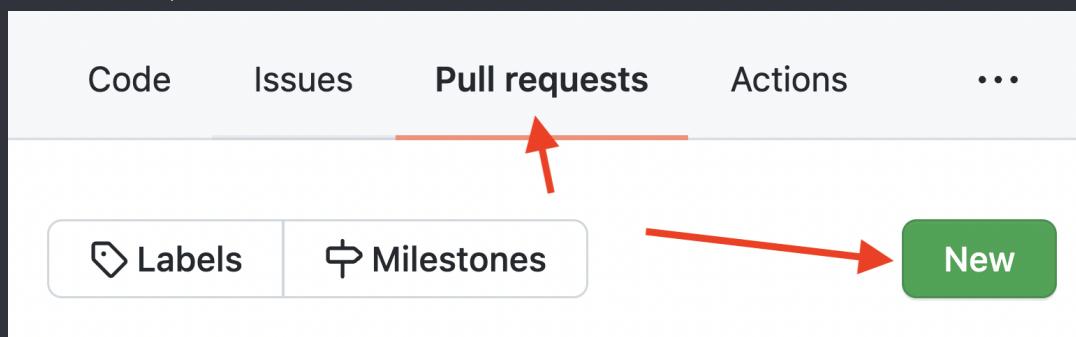
1. Create a branch for the new feature (called a "feature branch").

```
git branch new-feature
git checkout new-feature
Make some changes to the code...
git add .
git commit -m "new feature message"
```

2. Upload the feature branch to GitHub.

```
git push origin new-feature
```

3. Create a pull request on GitHub (a pull request lets teammates do code reviews and add comments).



4. Merge the feature branch into the main branch (by opening the pull request in the browser and clicking "Merge pull request")

A screenshot of a GitHub pull request interface. At the top, there's a green button labeled 'Merge pull request'. A red arrow points to this button. Below the button, there's a message: 'This branch has no conflicts with the base branch' followed by 'Merging can be performed automatically.' To the right of the button, it says 'You can also open this in GitHub Desktop or view command line instructions.'

5. After merging, update the local repository (so that it stays in sync with the remote repository on GitHub).

```
git checkout main  
git pull origin main
```

Merge Conflicts in the Feature Branch Workflow

A merge conflict can happen if 2 or more pull requests change the same file and the same line.

We can either:

1. Resolve the merge conflict on GitHub.

A screenshot of a GitHub merge conflict resolution page. It shows a warning: 'This branch has conflicts that must be resolved' with a link to 'the web editor or the command line'. Below this, it lists 'Conflicting files' and 'feature'. At the bottom, there's a 'Merge pull request' button and a 'Resolve conflicts' button, with a red arrow pointing to the 'Resolve conflicts' button. There's also a note: 'You can also open this in GitHub Desktop or view command line instructions.'

2. Resolve the merge conflict on our computer.

- 1) Get the latest updates from `main`

```
git checkout main  
git pull origin main
```

- 2) Get the latest updates from the feature branch.

```
git checkout feature4  
git pull origin feature4
```

- 3) Merge `main` into the feature branch (`feature4`). Notice the direction of the merge: we want the merge commit to stay on the feature branch so our teammates can review it.

```
git checkout feature4  
git merge master
```

4) Push the resolved feature branch to GitHub.

```
git push origin feature4
```

Now the pull request should be ready to merge again.

