Introduction to git and github

Contents

- Version Control Systems
- Git
- Github
- Quiz
- Configuration
- Walkthrough
- Practical 1
- Practical 2
- Practical 3

Version control

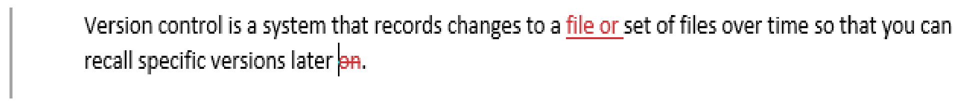
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later .
- Not just for code, can use it for any documents
- Allows you to keep track of changes (what was changed, when it was changed, who changed it)
- You may already practice something similar e.g.

- Final_1.txt

- Final_Final.txt

- Final_Final_definitely_the_last_one.txt

- Word - tracked changes
- Dropbox – keeping previous file versions



Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later ~~on~~.

Online book:https://git-scm.com

Photo by Kiwihug on Unsplash

Advantages of version control

Revert selected files back to a previous state	Revert an entire project back to a previous state	Compare changes over time
See who last modified something	When and who introduced a change and what that change was	Can easily recover files if you make a mistake
Branching and merging	Work with multiple people on a project at the same time and be alerted to conflicts (multiple people change same lines of code)	Track issues and commits that address them

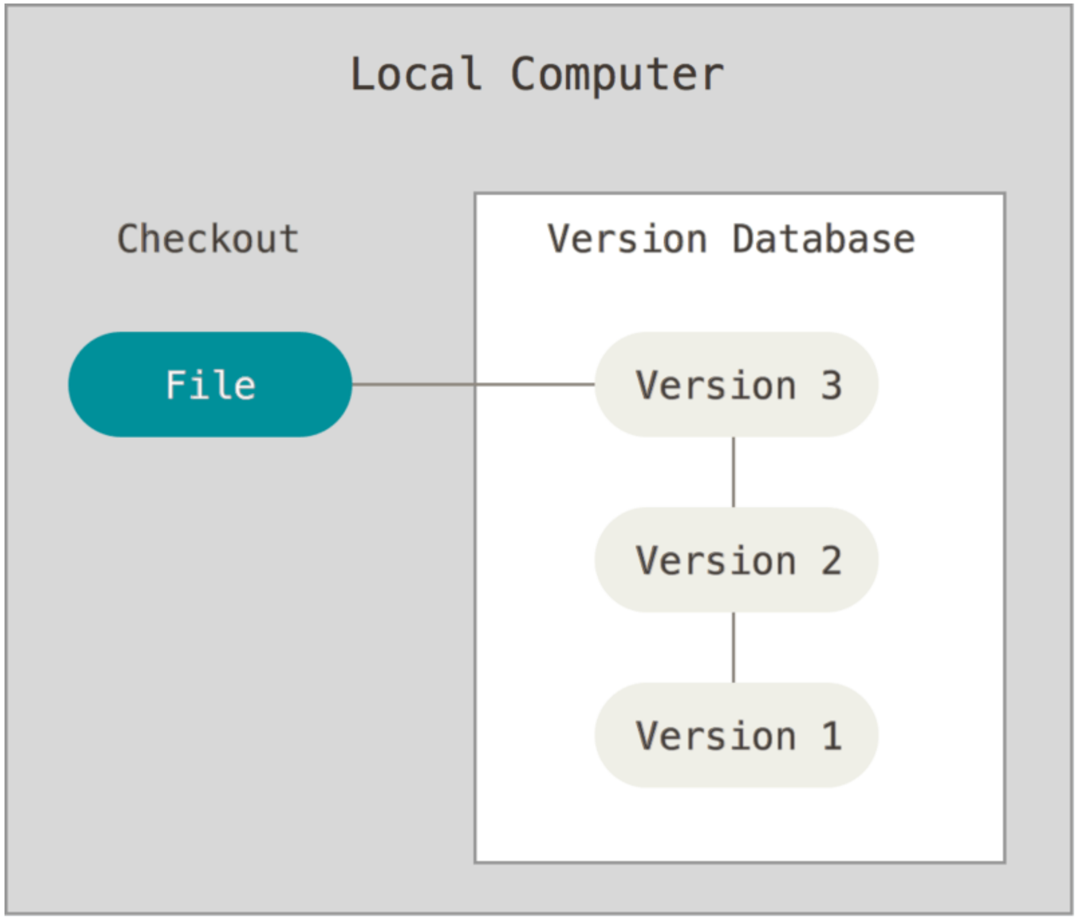
Version Control systems (VCS)

- Local
- Centralised
- Decentralised

Local VCS

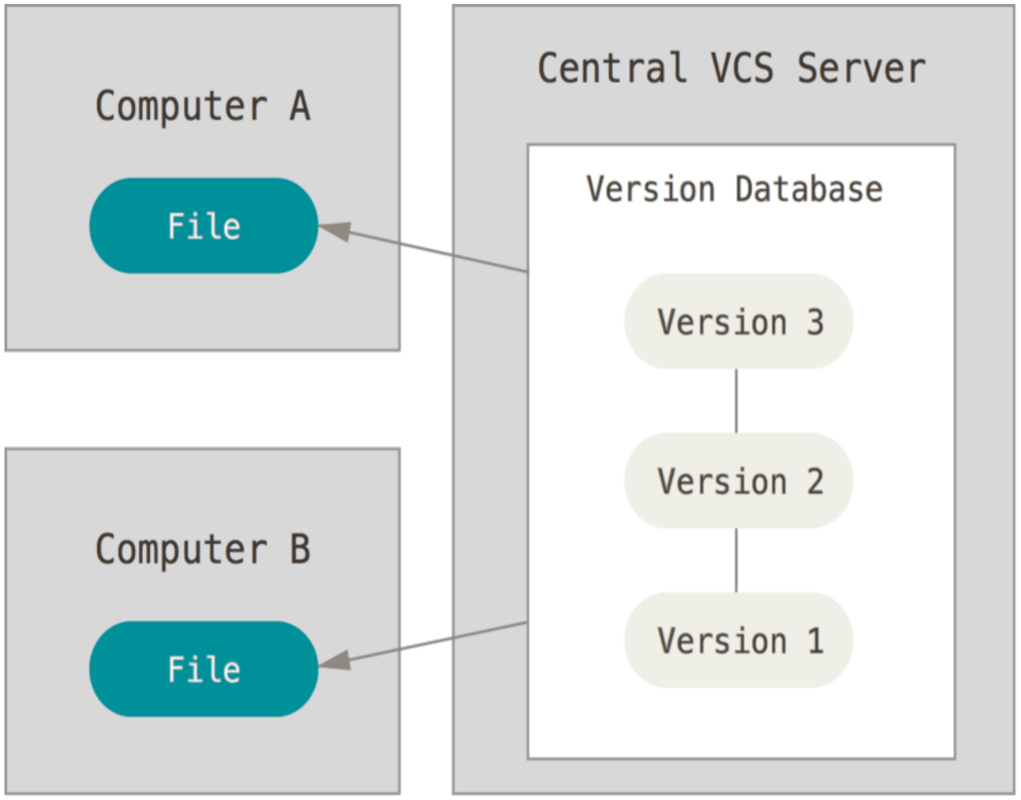
Most popular was RCS:
Revision Control System

- Works by keeping the differences between files (called patch sets) on disk
- Can then re-create what any file looked like at any point in time by adding up all the patches.



Centralised VCS (CVCS)

- You need to collaborate with others -> CVCS
- Standard for years
- Central server, people checkout files to work on
- Subversion
- Perforce
- CVS



Centralised VCS versus local VCS

Advantages of CVCS

- Know what others on a project are doing
- Administrators have fine-grained control over user permissions

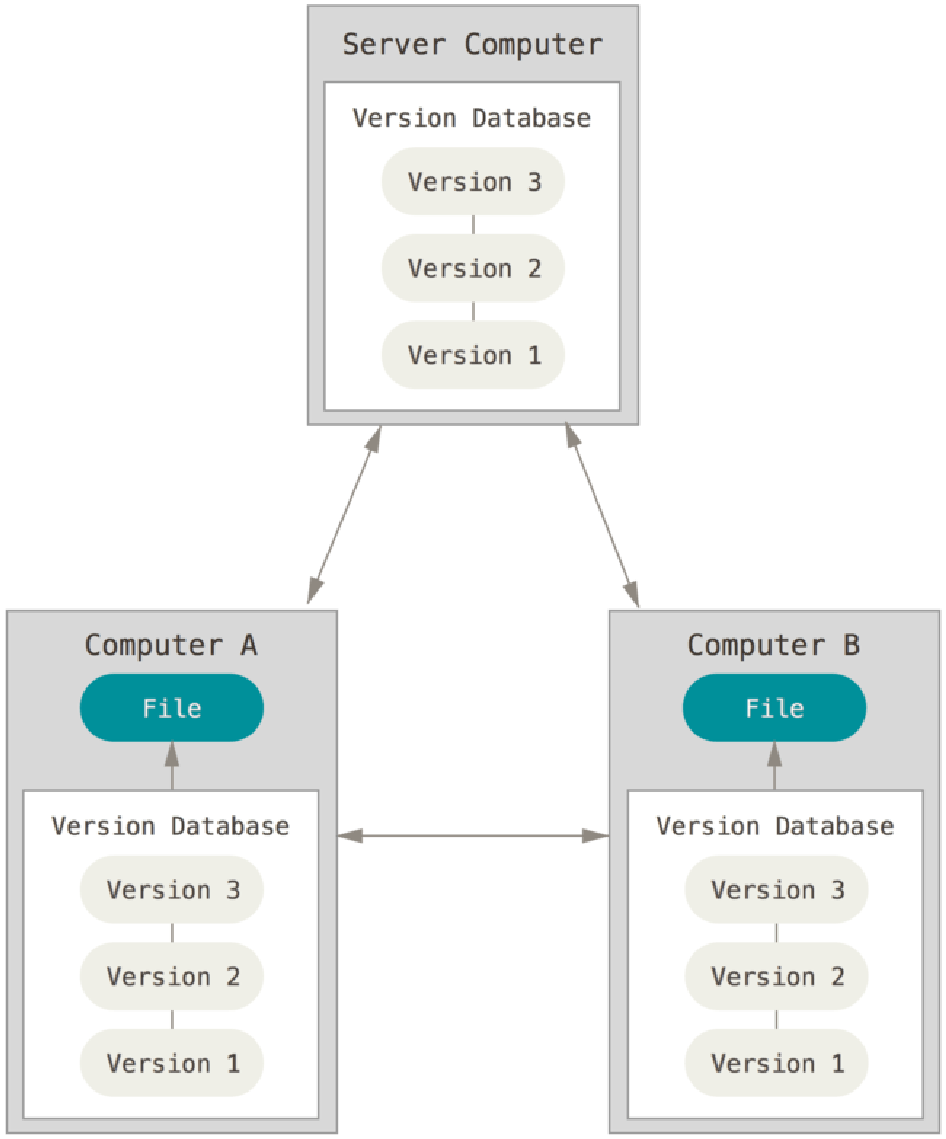
Disadvantages of Centralised VCS and local VCS

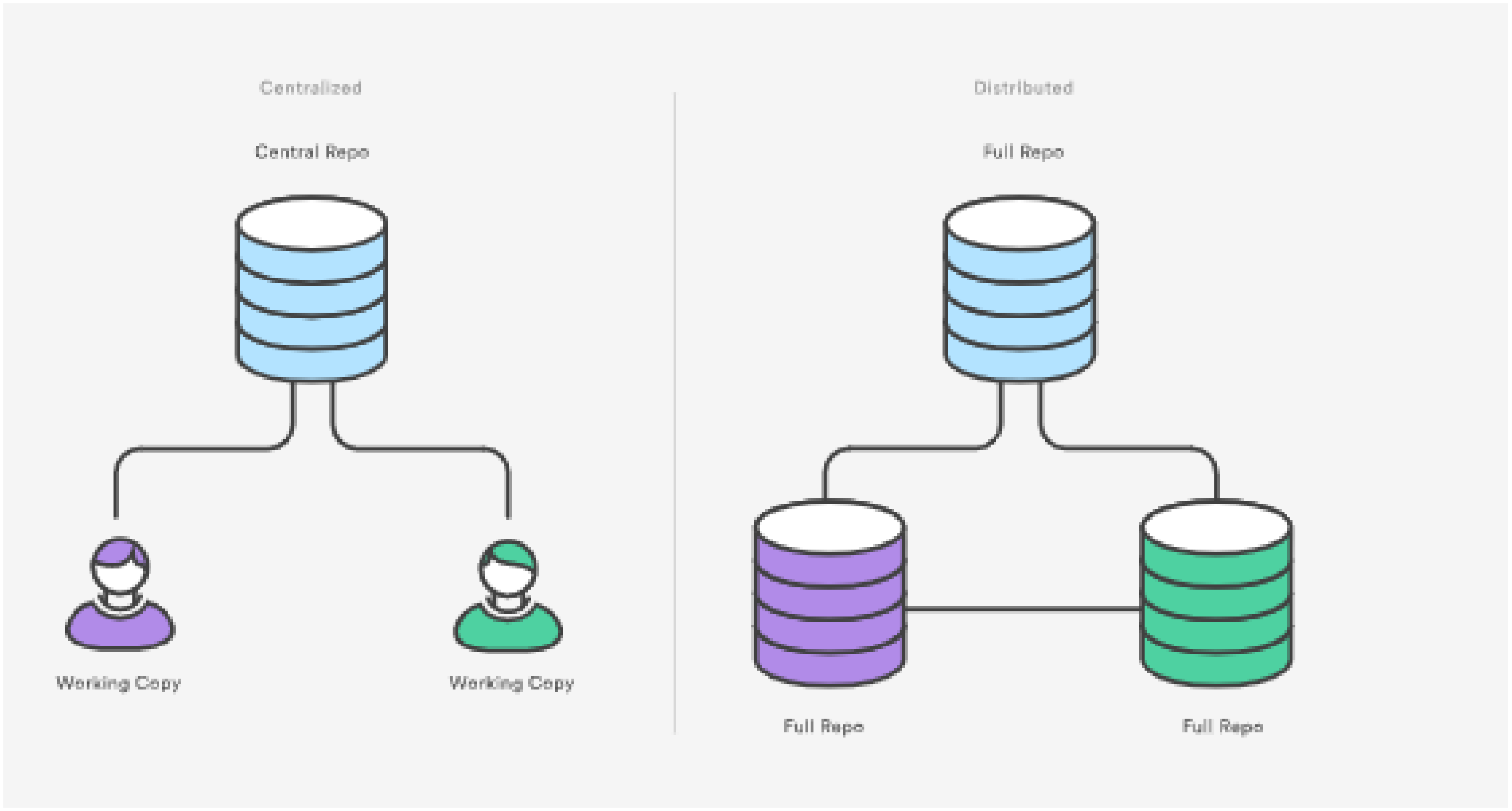
- Centralised server in CVCS is a single point of failure
- If it goes down for a bit, cant save changes, collaborate etc
- If hard-disk gets corrupted and is not backed up, lose everything etc whatever ‘snapshots’ people gace in their computer.
- Same issue with local VCS

Decentralised VCS (DVCS)

- Client computers fully mirror the repository, including its full history instead of only checking out a local snapshot
- If the server dies, any the repositories on the client computers can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.

- Git
- Mercurial
- Bazaar
- Darcs

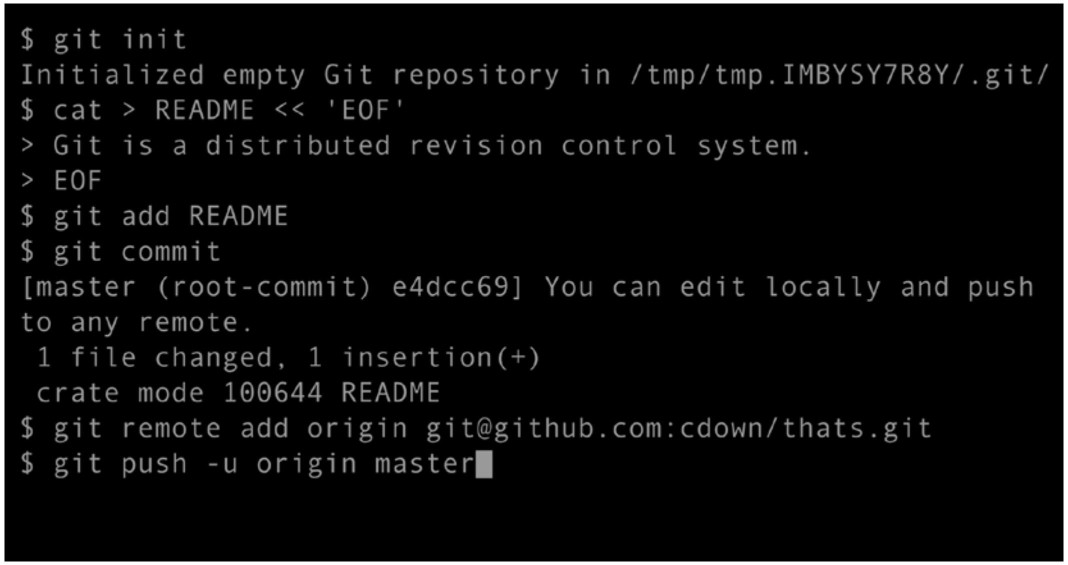




https://www.atlassian.com/git/tutorials/why-git

Git

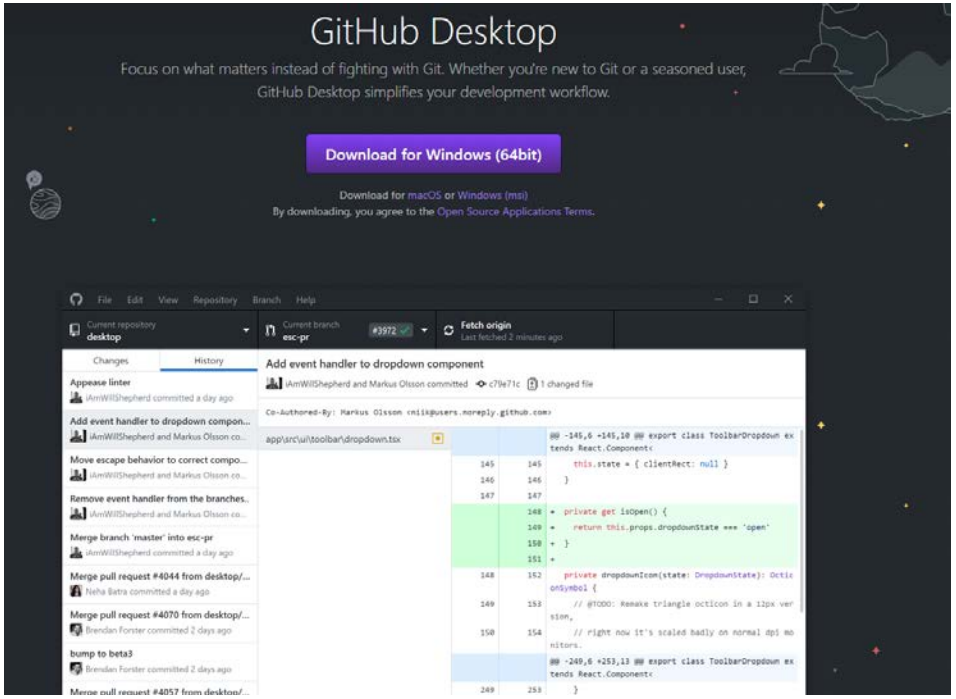
- **Git** is a distributed version-control system for tracking changes in source code during software development.
- Designed for coordinating work among programmers, but it can be used to track changes in any set of files
- Free and open source
- Open source (OSS): software in which source code is released under a license in which the copyright holder grants users the rights to use, study, change, and distribute the software to anyone and for any purpose.
- May be developed in a collaborative public manner (https://en.wikipedia.org/wiki/Open-source_software)



```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 crate mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

git command line

https://en.wikipedia.org/wiki/Git#/media/File:Git_session.svg



GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Download for Windows (64bit)
Download for macOS or Windows (msi)
By downloading, you agree to the Open Source Applications Terms.
File
Edit
View
Repository
Branch
Help
Current repository
desktop
Current branch
esc-pr
Fetch origin
Last fetched 2 minutes ago
Changes
History

Appease linter
iAmWillShepherd committed a day ago

Add event handler to dropdown compon...
iAmWillShepherd and Markus Olsson co...

Move escape behavior to correct compo...
iAmWillShepherd and Markus Olsson co...

Remove event handler from the branches...
iAmWillShepherd and Markus Olsson co...

Merge branch 'master' into esc-pr
iAmWillShepherd committed a day ago

Merge pull request #4044 from desktop/...
Neha Batra committed a day ago

Merge pull request #4070 from desktop/...
Brendan Forster committed 2 days ago

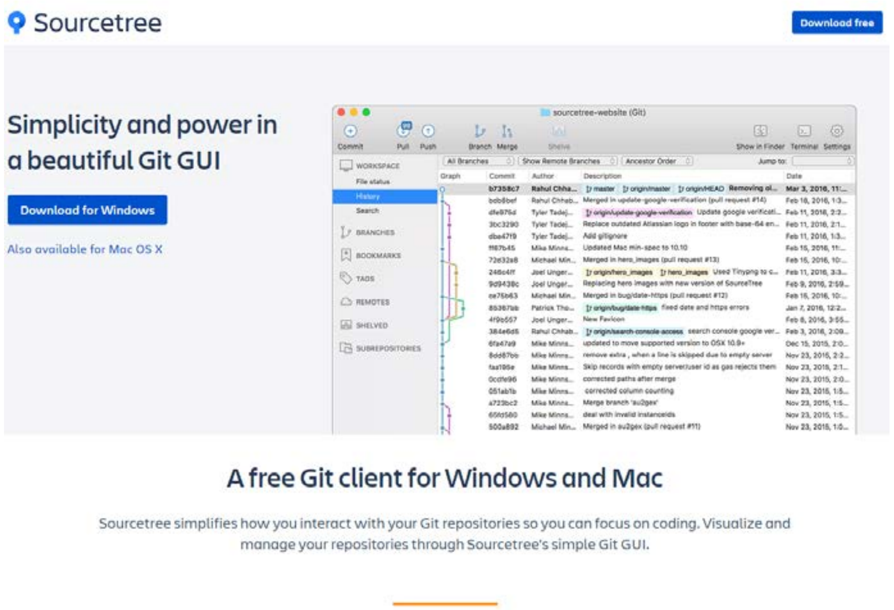
bump to beta3
Brendan Forster committed 2 days ago

Add event handler to dropdown component
iAmWillShepherd and Markus Olsson committed c79e71c 1 changed file

```
Co-Authored-By: Markus Olsson <niik@users.noreply.github.com>

app\src\ui\toolbar\dropdown.tsx

@@ -145,6 +145,10 @@ export class ToolbarDropdown extends React.Component<
      this.state = { clientRect: null }
    }

+   private get isOpen() {
+     return this.props.dropdownState === 'open'
+   }
+
    private dropdownIcon(state: DropdownState): OcticonSymbol {
      // @TODO: Remake triangle octicon in a 12px version,
      // right now it's scaled badly on normal dpi monitors.
@@ -249,6 +253,13 @@ export class ToolbarDropdown extends React.Component<
    }
```

GitKraken

Legendary Git client for Windows, Mac & Linux. Free for open source



Sourcetree

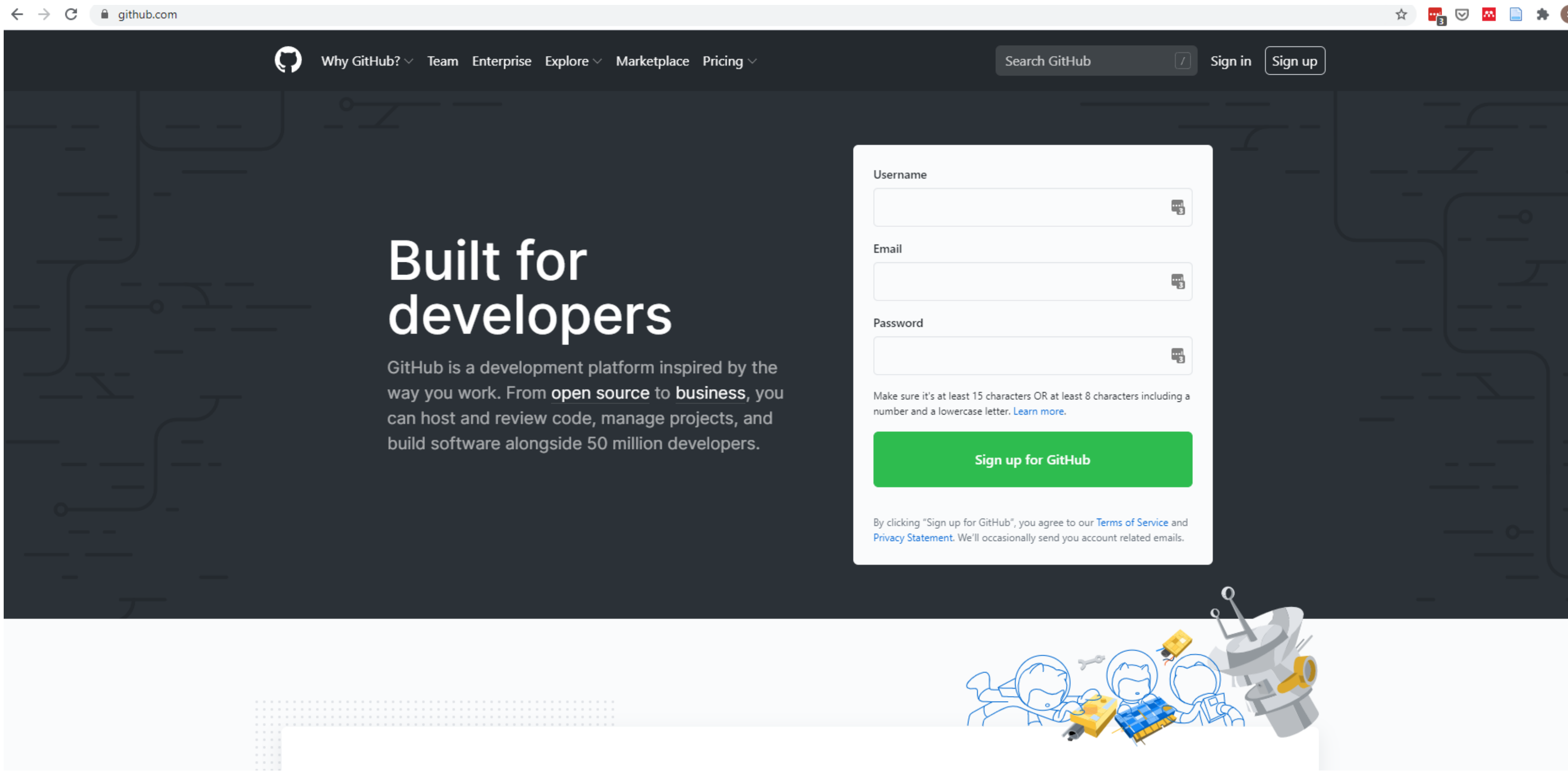
Simplicity and power in a beautiful Git GUI

A free Git client for Windows and Mac

Sourcetree simplifies how you interact with your Git repositories so you can focus on coding. Visualize and manage your repositories through Sourcetree's simple Git GUI.

Some Git GUI's

Github: Our remote repository



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

Username

Email

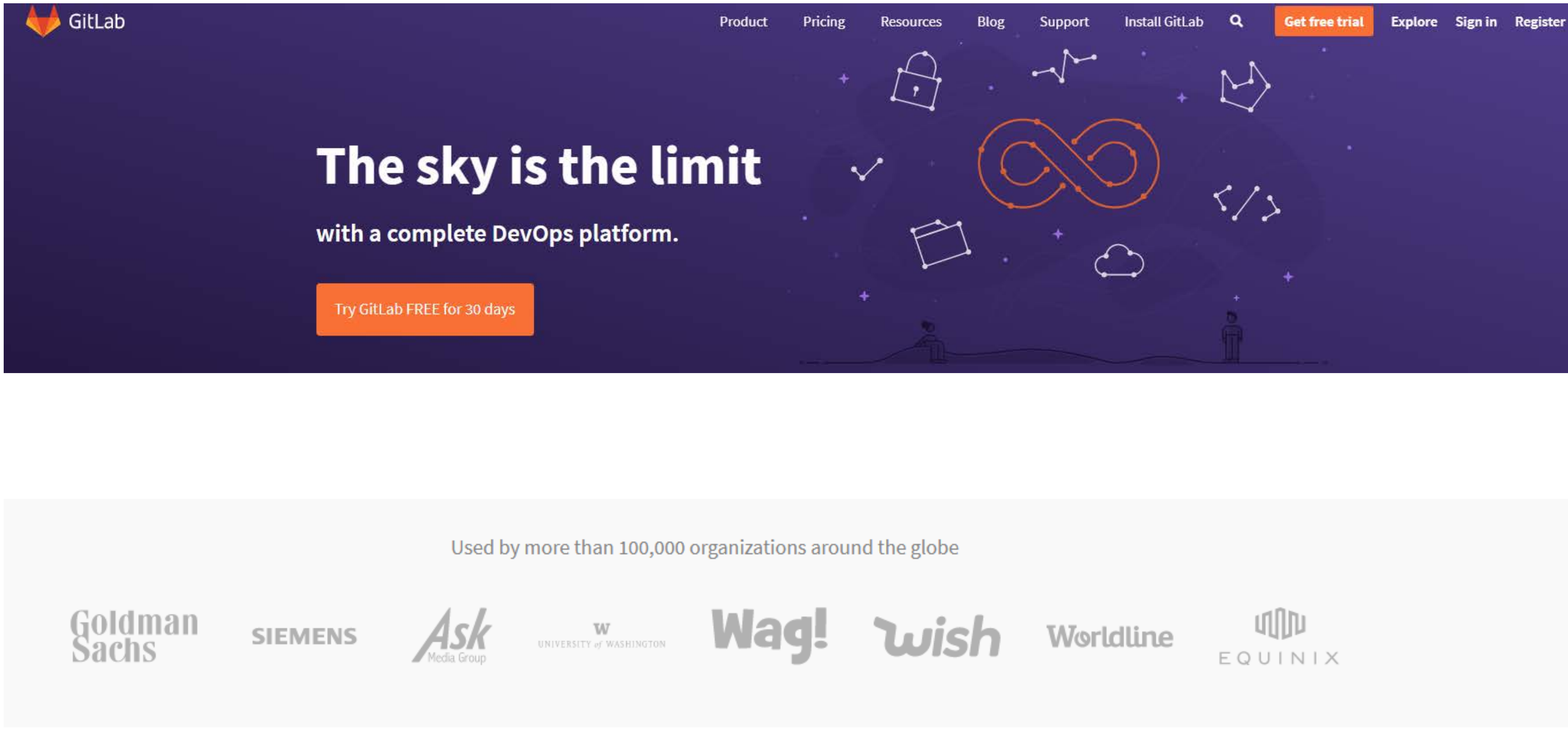
Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

GitLab



The sky is the limit

with a complete DevOps platform.

Used by more than 100,000 organizations around the globe

https://about.gitlab.com/

Git History

- The Linux kernel is an open source software project.
- For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files.
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.
- In 2005, relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This lead to linux team designing git.

Aims of git design:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)
- Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities
- Very fast and efficient system

How Git doesn’t work

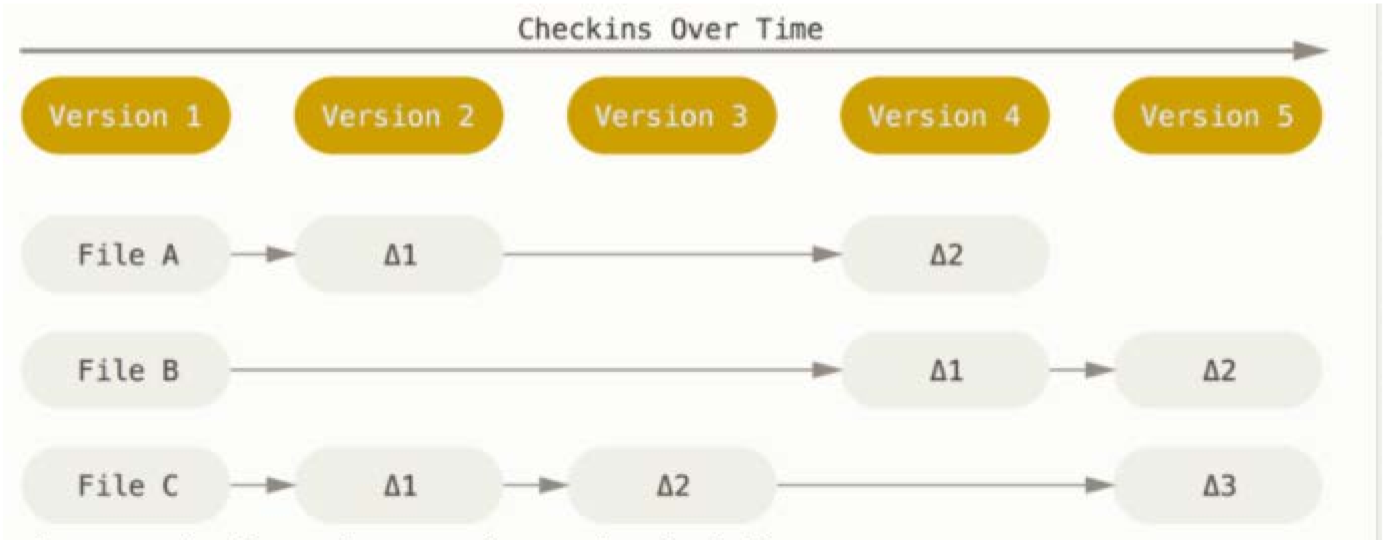


Figure 4. Storing data as changes to a base version of each file

- Most other VCS store information as a list of file-based changes e.g. CVS, Subversion, Perforce

- These think of the information they store as a set of files and the changes made to each file over time (this is commonly described as **delta-based** version control).

How Git works

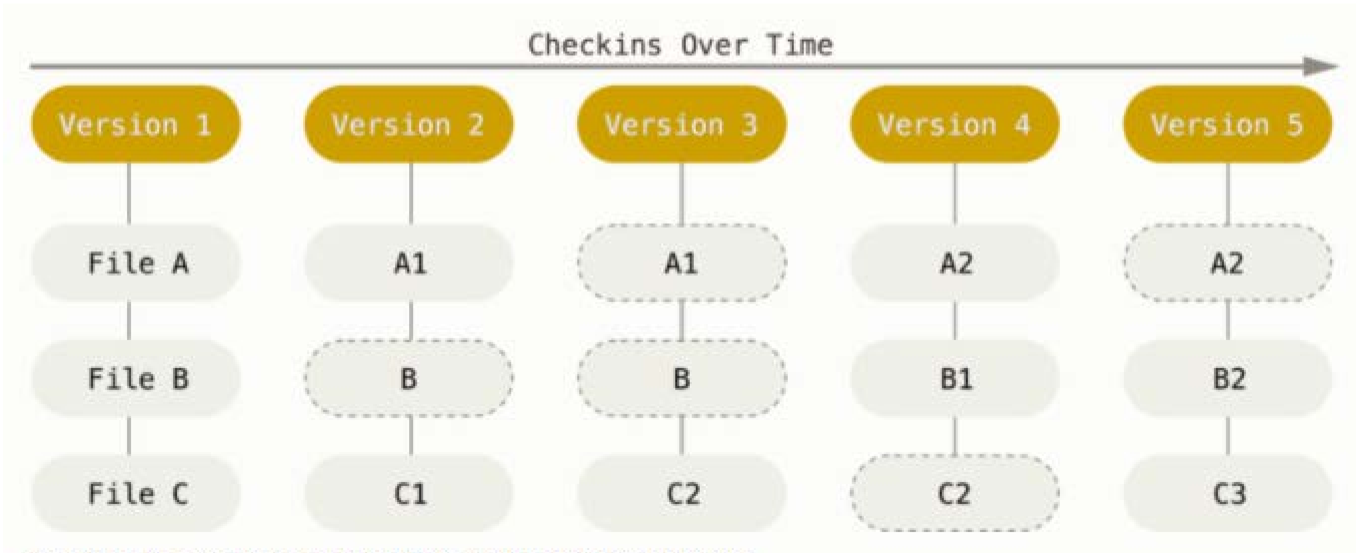
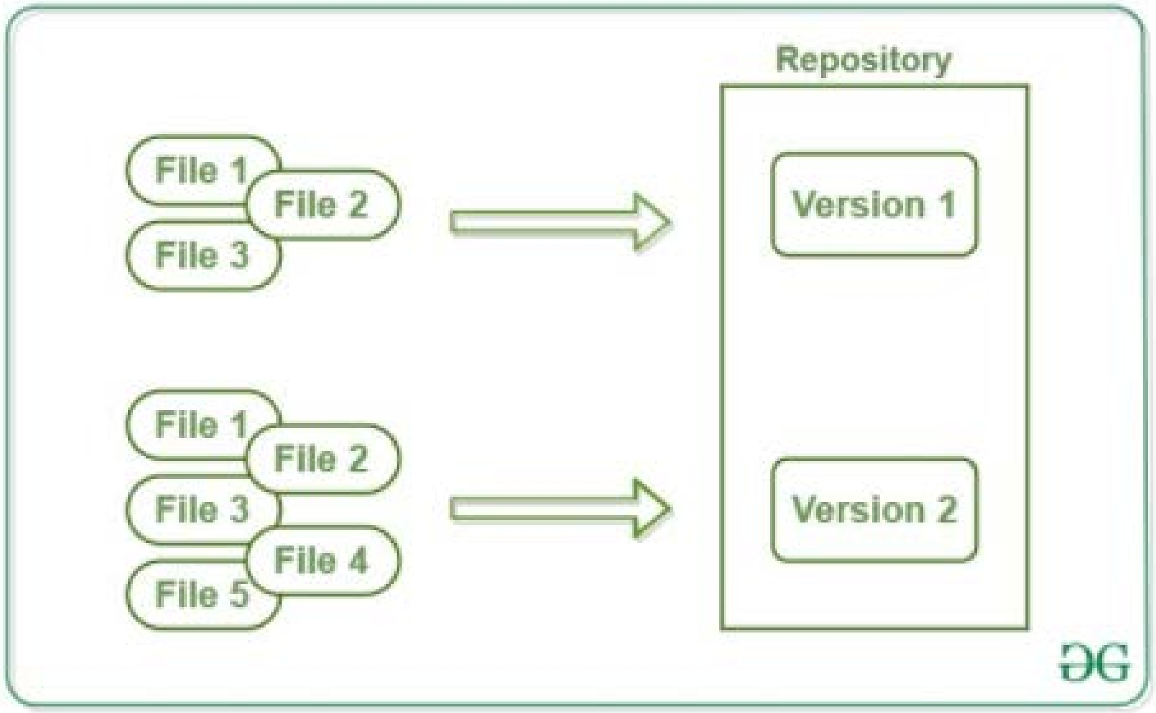


Figure 5. Storing data as snapshots of the project over time

- Git thinks of its data more like a series of snapshots of a miniature filesystem.
- With Git, every time you commit (save the state of your project) git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- To be efficient, if files have not changed, Git doesn’t store the file again, just a link to the previous identical file it has already stored.
- Git thinks about its data more like a **stream of snapshots**.

- Repositories in Git contain a collection of files of various different versions of a Project.
- Contains both the files and folders that you create and the extra info that git records about their history.
- Git info is stored in the root of the repository in .git



https://www.geeksforgeeks.org/what-is-a-git-repository/

Git Integrity

- Everything in Git is checksummed before it is stored and is then referred to by that checksum
- Impossible to change the contents of any file or directory without Git knowing about it.
- Can’t lose info in transit or get corrupted files without that being detetcted
- The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. A SHA-1 hash looks something like this:
- `24b9da6552252987aa493b52f8696cd6d3b00373`
- Git stores everything in its database by the hash value of its contents (not the filename) - > compares hashs to see if something changed

Git Generally Only Adds Data

- Nearly all actions in git only **add** data to the Git database.
- Hard to do something undoable or to make it erase data in any way.
- But you can lose or mess up changes you haven't committed yet,

-> commit and push to remote regularly

- Hard to severely screwing things up!
- See undoing things if you get stuck: https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things#_undoing

Git only needs local resources

- Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network
- For example, to browse the history of the project, Git doesn’t need to go out to the remote server to get the history and display it for you — it simply reads it directly from your local database.
- Enables offline working as can commit to local copy until you get a network connection to upload changes to remote version
- In many other systems, working offline is either impossible or painful. In Subversion and CVS, you can edit files, but you can’t commit changes to your database (because your database is offline).

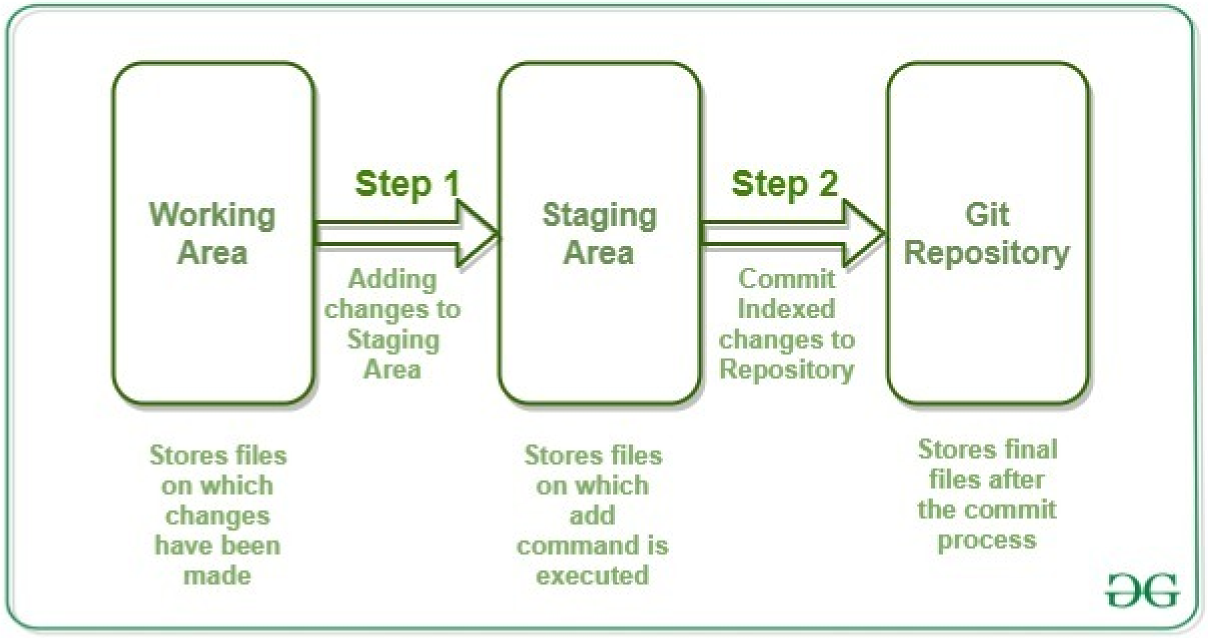
Git File Stages

Git has three main states that your files can reside in: **modified**, **staged**, and **committed**:

•**Modified** means that you have changed the file **but have not committed it** to your database yet.

•**Staged** means that you have marked a modified file in its current version to go into your **next commit** snapshot.

•**Committed** means that the data is safely stored in your local database (**saved** to it).


Working Area
Step 1
Adding changes to Staging Area
Staging Area
Step 2
Commit Indexed changes to Repository
Git Repository
Stores files on which changes have been made
Stores files on which add command is executed
Stores final files after the commit process

Some definitions

- **Repository**: a collection of files of various different versions of a project.
- **Cloning**: The process of copying the content from an existing Git Repository with the help of various Git Tools is termed as **cloning**. You get the complete repository on your local machine. (git clone …)
- **Staging**: specify what files with changes that you want to go in your next commit
- **Committing**: Store a snapshot of your repository with all file changes marked (record the changes to the repo)
- **Pushing:** update the remote repository with your changes

More info

- The **working tree** is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- The **staging area** is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name is the "index", but the phrase "staging area" works just as well.
- The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you **clone** a repository from another computer.

Basic Workflow: Stage, commit, push

- The basic Git workflow goes something like this:

1. You modify files in your working tree.

For example, you open a file named mywork.txt, add the date to it, and save it

2. You selectively stage just those changes you want to be part of your next commit, which adds **only** those changes to the staging area. This is called staging your changes. Can add multiple files to staging area also.

```
git add mywork.txt
```

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

```
git commit –m “added date to mywork.txt”
```

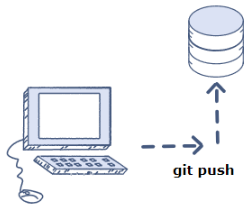
4. If a particular version of a file is in the Git directory, it’s considered **committed**. If it has been modified and was added to the staging area, it is **staged**. And if it was changed since it was checked out but has not been staged, it is **modified**..

5. Push changes to remote repository to update it

```
git push –u origin master
```

Synthax: git push 'remote_name' 'branch_name’

Origin = default remote repository name



Basic workflow: Pulling and status

6. You come back to work on repo after last use and others have added changes to the remote repo so you need to pull these to your repo

```
git pull
```

7. Can check what files staged etc using

```
git status
```

8. Check how a file/directory is different to its committed version

```
Git diff <filename>
Git diff <directoryname>
git diff
git diff -r HEAD <filename> (compare most recent commit with past ones)
```

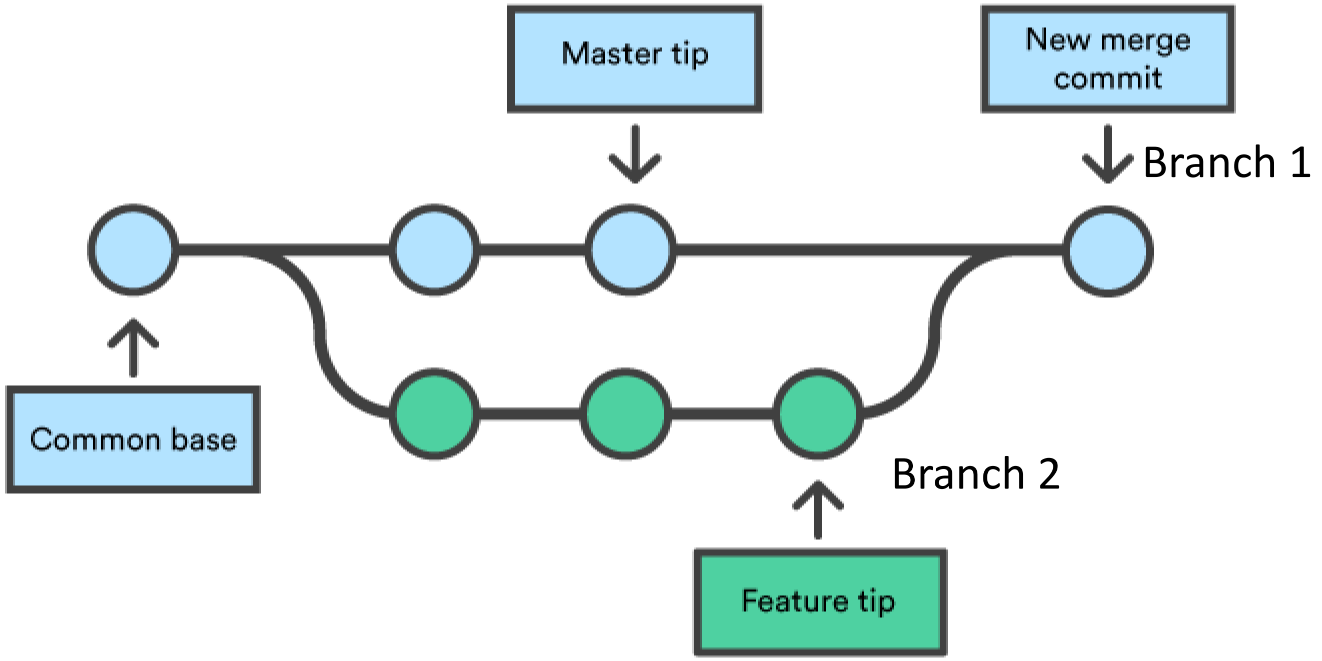
9. See git log of commits

```
git log
git log –oneline
```

10. Show a particular commit

```
git show <first six characters of hash>
 git show HEAD
git show HEAD~1
```

Branching: independent lines of development within a repository



A branch you create to build a feature is commonly referred to as a feature branch or topic branch

https://www.atlassian.com/git/tutorials/using-branches/git-merge

Git branch commands

- Create a branch called ‘dev’

```
git branch dev
```

- Switch to the dev branch

```
git checkout dev
```

Make changes to files… Then:

- Stage changes (command below adds all files; be careful not to add any sensitive files or log files etc; add their names or pattern to .gitignore file eg. *.log)

```
git add .
```

- Commit changes

```
git commit –m “my message”
```

…can make more changes and more commits using add and commit again

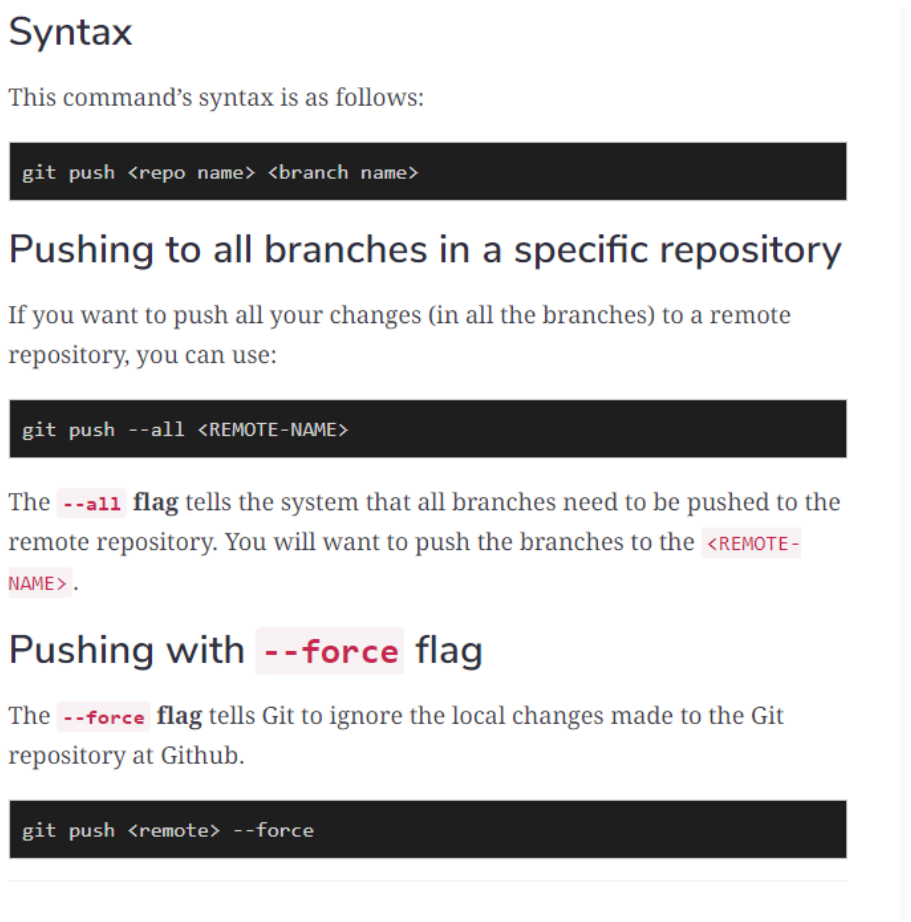
- Push to dev branch of the remote

```
git push origin dev
```

- Switch back to master branch

```
git checkout master
```

Pushing changes in many branches at the same time



Syntax

This command's syntax is as follows:

```
git push <repo name> <branch name>
```

Pushing to all branches in a specific repository

If you want to push all your changes (in all the branches) to a remote repository, you can use:

```
git push --all <REMOTE-NAME>
```

The `--all` **flag** tells the system that all branches need to be pushed to the remote repository. You will want to push the branches to the `<REMOTE-NAME>`.

Pushing with `--force` flag

The `--force` **flag** tells Git to ignore the local changes made to the Git repository at Github.

```
git push <remote> --force
```

https://www.educative.io/edpresso/push-command-in-git

A note on git Origin

- 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.

https://www.git-tower.com/learn/git/glossary/origin#:~:text=In%20Git%2C%20%22origin%22%20is,but%20just%20a%20standard%20convention.

Fetch vs pull vs sync

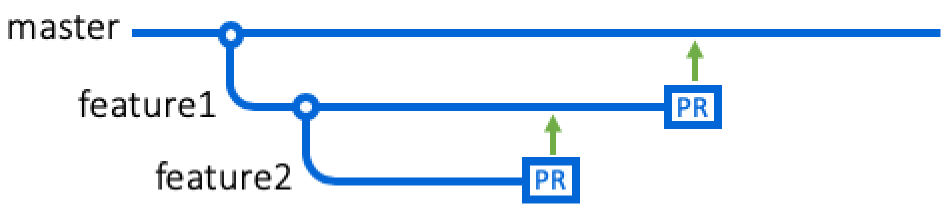
git fetch is the command that tells your local git to retrieve the latest meta-data info from the original (yet doesn't do any file transferring. It's more like just checking to see if there are any changes available). Useful if you want to see changes before you incorporate them.

git pull on the other hand does that (git fetch) AND brings (copy) those changes from the remote repository.

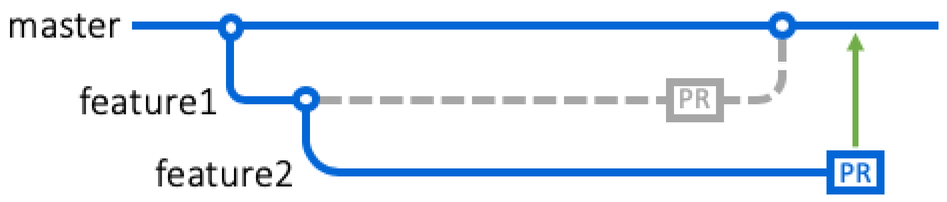
git sync pulls and pushes

https://www.freecodecamp.org/news/git-fetch-vs-pull/

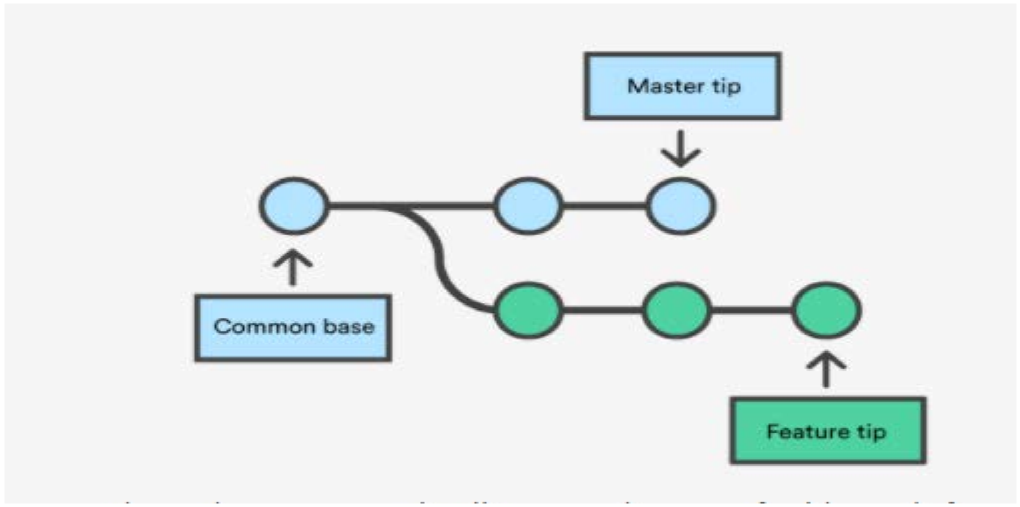
Branches of branches



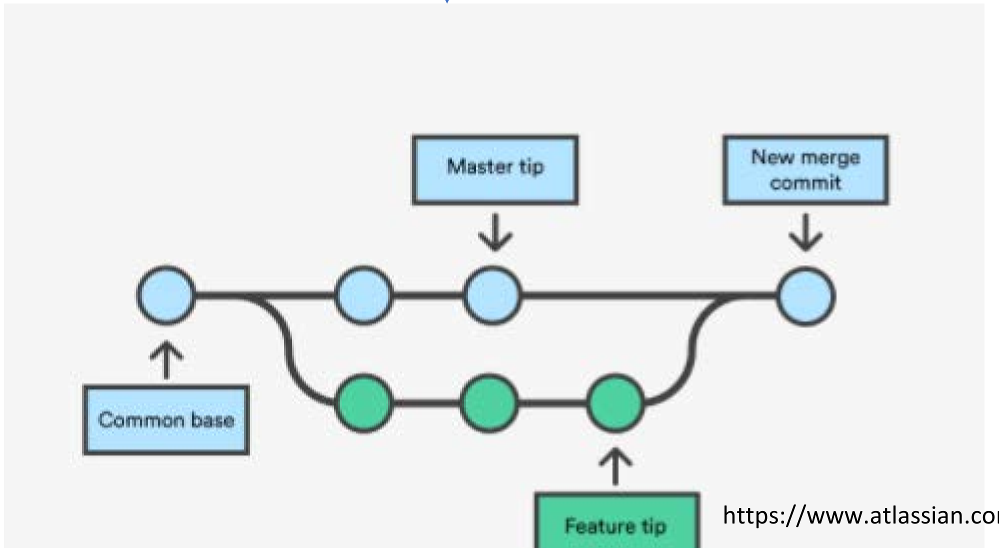
In the next diagram, someone has merged the pull request for `feature1` into the `master` branch, and they have deleted the `feature1` branch. As a result, GitHub has automatically retargeted the pull request for `feature2` so that its base branch is now `master`.



https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-branches



merge



https://www.atlassian.com/git/tutorials/using-branches/git-merge

Pull requests

- Once you're satisfied with your work, you can open a pull request to merge the changes in the current branch (the *head* branch) into another branch (the *base* branch).

In git bash

Move back to the master branch

```
git checkout master
```

Merge the dev branch into the master branch

```
git merge dev
```

Push back to remote

```
git push –u origin master
```

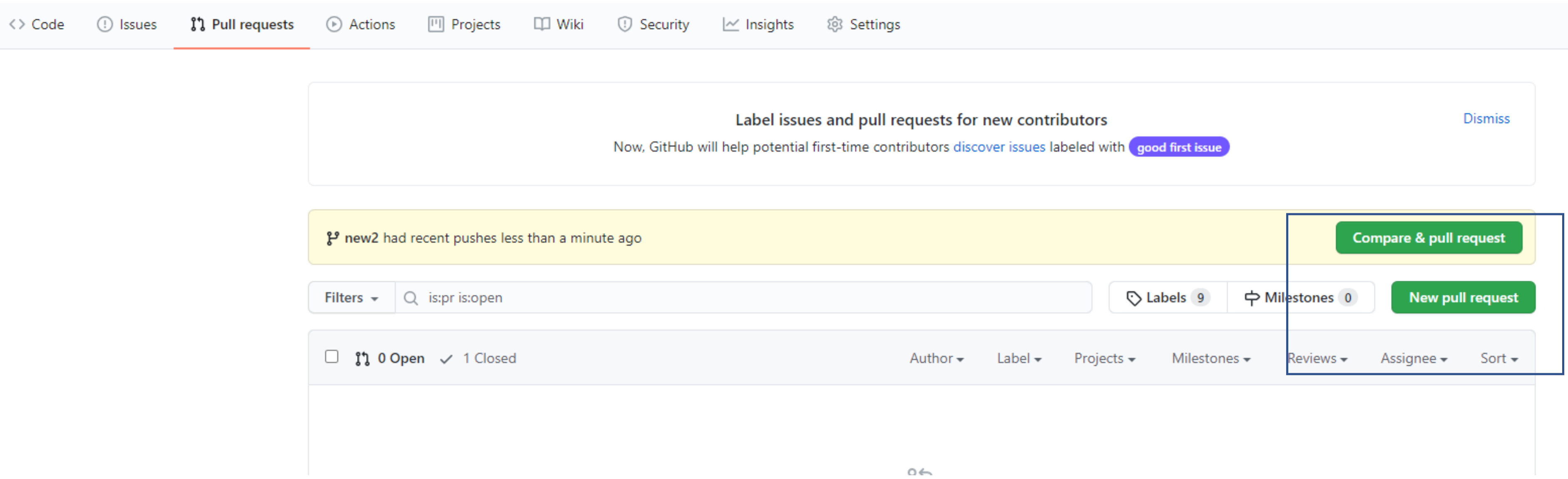
Could also do

```
git merge dev master
```

git merge <source> <destination>

https://www.atlassian.com/git/tutorials/using-branches/git-merge

Can also do on github (safer..)



Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

Dismiss

new2 had recent pushes less than a minute ago

Merge conflicts

If Git encounters the same piece of code changed in different ways on both branches, it will be unable to automatically combine them. This creates a merge conflict and git will ask you to intervene and choose what/how you want to merge.

Forking: git fork

- A fork is a copy of a repository. Forking a repository allows you to freely test and debug with changes *without affecting the original project.*

- Reasons to fork a repo:

1. Propose changes to someone else's project.
2. Use an existing project as a starting point.

Steps:

1. Fork the repository.
2. Make your changes
3. Issue a pull request to the project owner.

What we have covered so far

- Version control systems
- Git
- General Git workflow
- Branching and merging
- Forking

Next:

- Quiz
- Configuring git and github to talk to each other
- Walkthrough
- Practice

Quiz on blackboard (10 mins)

Configuring your git

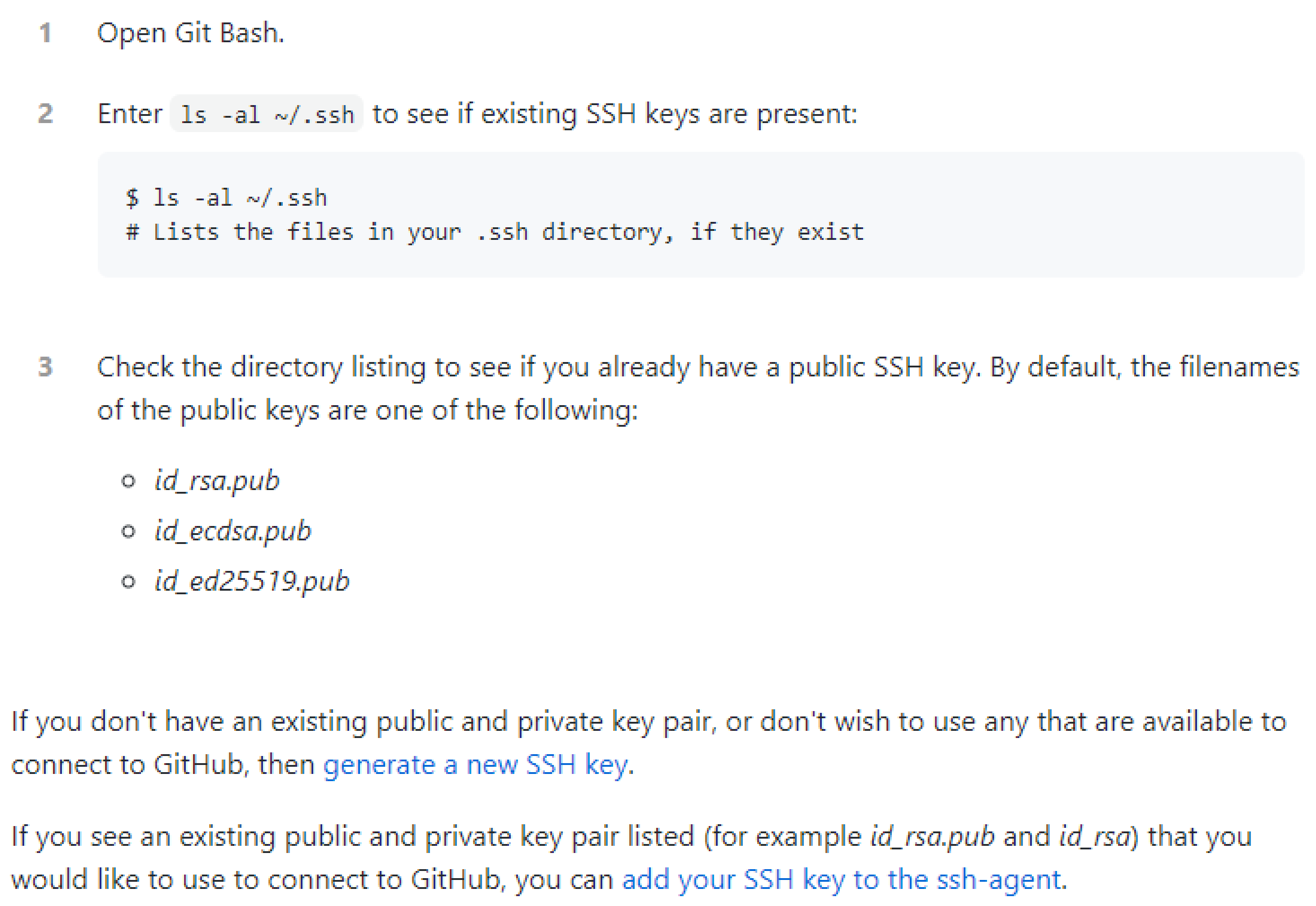
We need to use an ssh key to get git on your computer to talk to your Github account

Steps:

1. Open Git bash (in windows) or go to the command line in linux

https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

2. Check for existing ssh key



1. Open Git Bash.
2. Enter `ls -al ~/.ssh` to see if existing SSH keys are present:

   ```
   $ ls -al ~/.ssh
   # Lists the files in your .ssh directory, if they exist
   ```

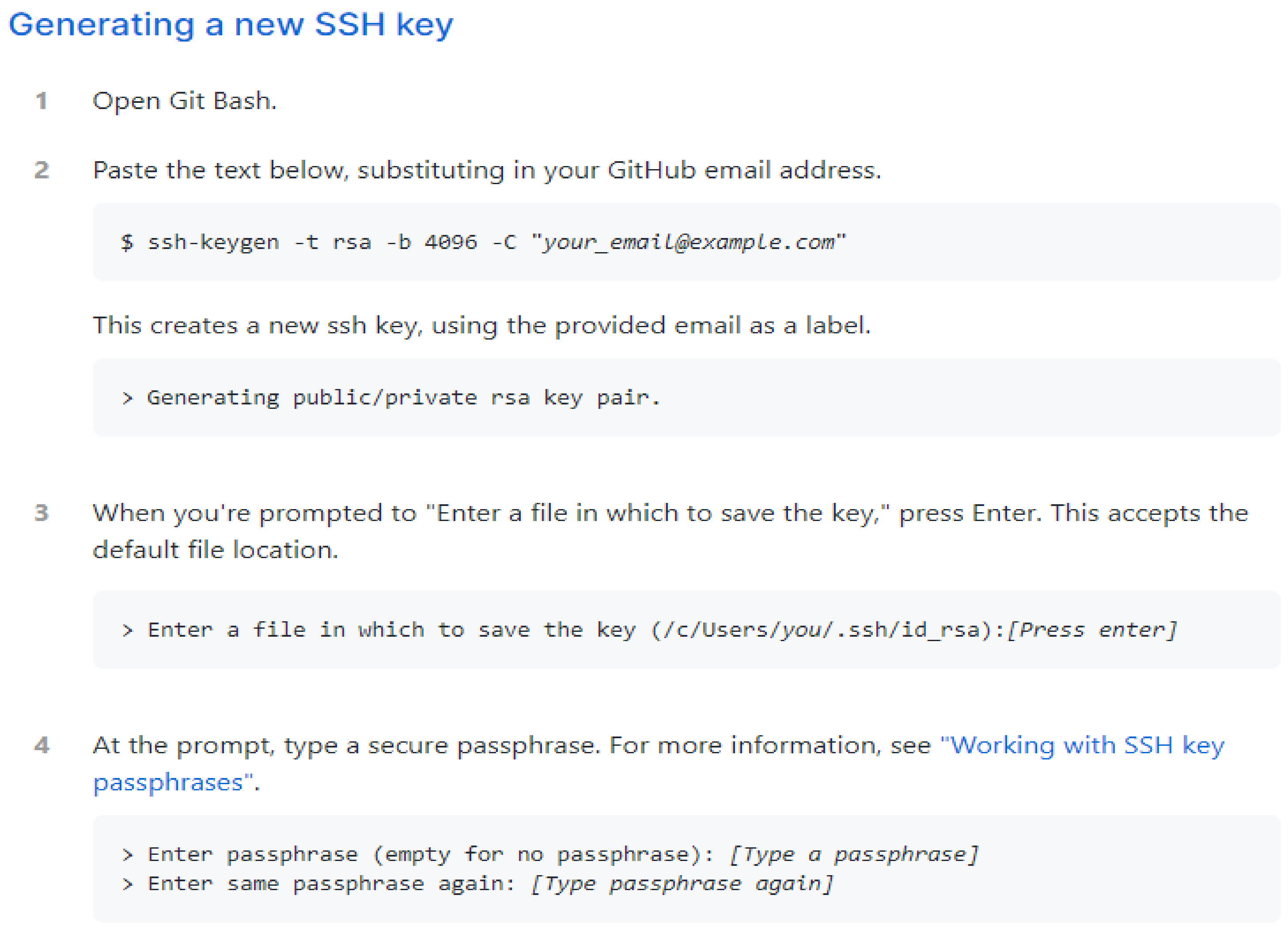
3. Check the directory listing to see if you already have a public SSH key. By default, the filenames of the public keys are one of the following:
   - *id_rsa.pub*
   - *id_ecdsa.pub*
   - *id_ed25519.pub*

If you don't have an existing public and private key pair, or don't wish to use any that are available to connect to GitHub, then generate a new SSH key.

If you see an existing public and private key pair listed (for example *id_rsa.pub* and *id_rsa*) that you would like to use to connect to GitHub, you can add your SSH key to the ssh-agent.

https://docs.github.com/en/github/authenticating-to-github/checking-for-existing-ssh-keys

3. If no key found – generate key



Generating a new SSH key

1 Open Git Bash.

2 Paste the text below, substituting in your GitHub email address.

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

3 When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

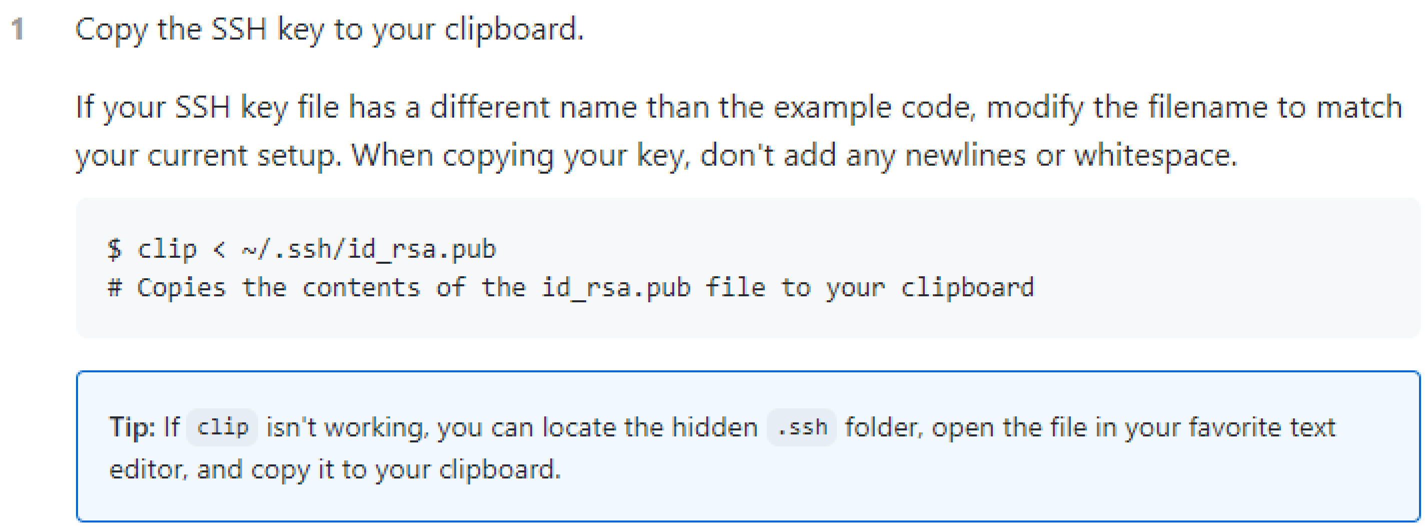
```
> Enter a file in which to save the key (/c/Users/you/.ssh/id_rsa):[Press enter]
```

4 At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases".

```
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

4. Adding your ssh key to github



1 Copy the SSH key to your clipboard.

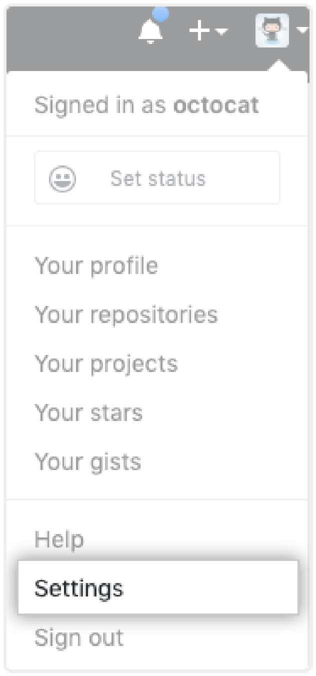
If your SSH key file has a different name than the example code, modify the filename to match your current setup. When copying your key, don't add any newlines or whitespace.

```
$ clip < ~/.ssh/id_rsa.pub
# Copies the contents of the id_rsa.pub file to your clipboard
```

Tip: If `clip` isn't working, you can locate the hidden `.ssh` folder, open the file in your favorite text editor, and copy it to your clipboard.

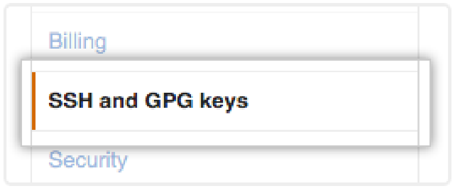
https://docs.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account

2 In the upper-right corner of any page, click your profile photo, then click **Settings**.

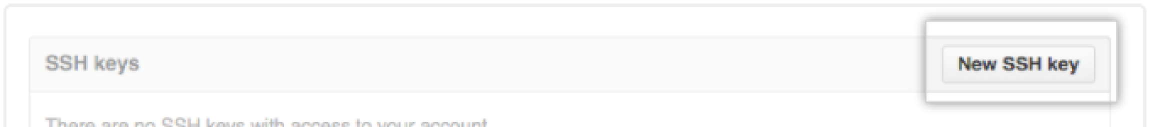


Go to Github and sign in

3 In the user settings sidebar, click **SSH and GPG keys**.



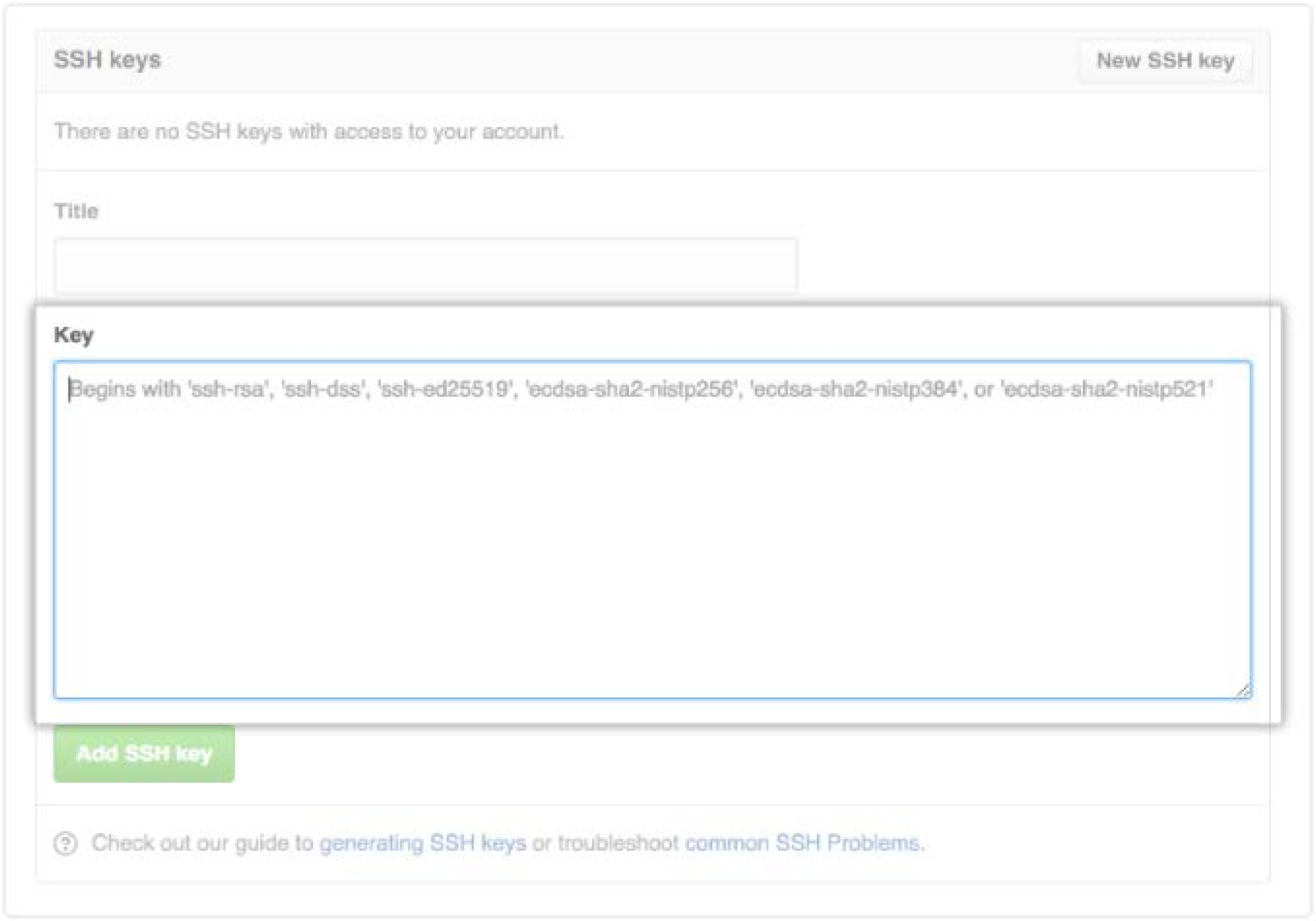
4 Click **New SSH key** or **Add SSH key**.



There are no SSH keys with access to your account.

5 In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Air".

6 Paste your key into the "Key" field.


SSH keys

New SSH key

There are no SSH keys with access to your account.

Title

Key

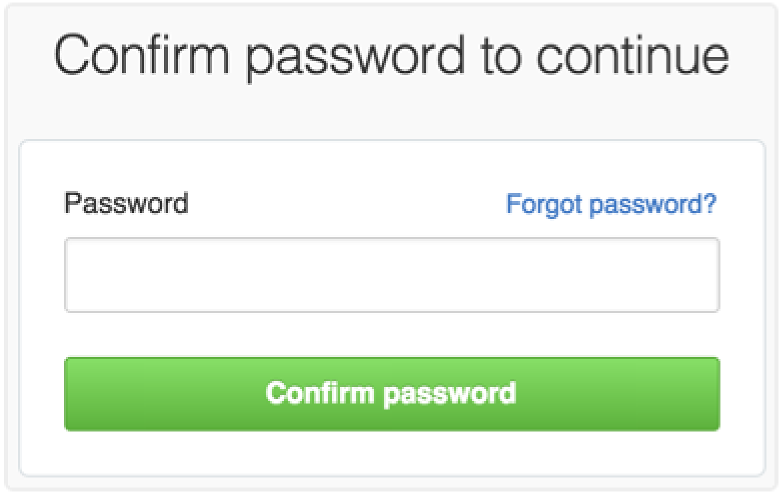
Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Add SSH key

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.

7 Click **Add SSH key**.

8 If prompted, confirm your GitHub password.



Confirm password to continue

Configuring git on terminal/bash

Open git bash

- git config --global user.email “your-github-email-address”
- git config --global user.name “your github username“

Confirming it works

- In git bash type:

```
ssh -T git@github.com
```

If it says something like the text below, then setup has worked:

```
"Hi username! You've successfully authenticated, but Github does not provide shell access"
```

Standard use walkthrough

- Make repo
- Clone repo to computer
- Make changes
- Stage,commits + push
- Pull changes
- Fork a repo
- Markdown – files, images , urls
- Danger zone: deleting repo etc
- Create branches , make changes
- Merge branches
- Protect branches
- Make an issue
- Reference it in a commit
- Send someone a link to a line in of your code
- Add members to your repo

Practical – Group work

- https://github.com/coughls/Git-training/blob/master/git-practical.md

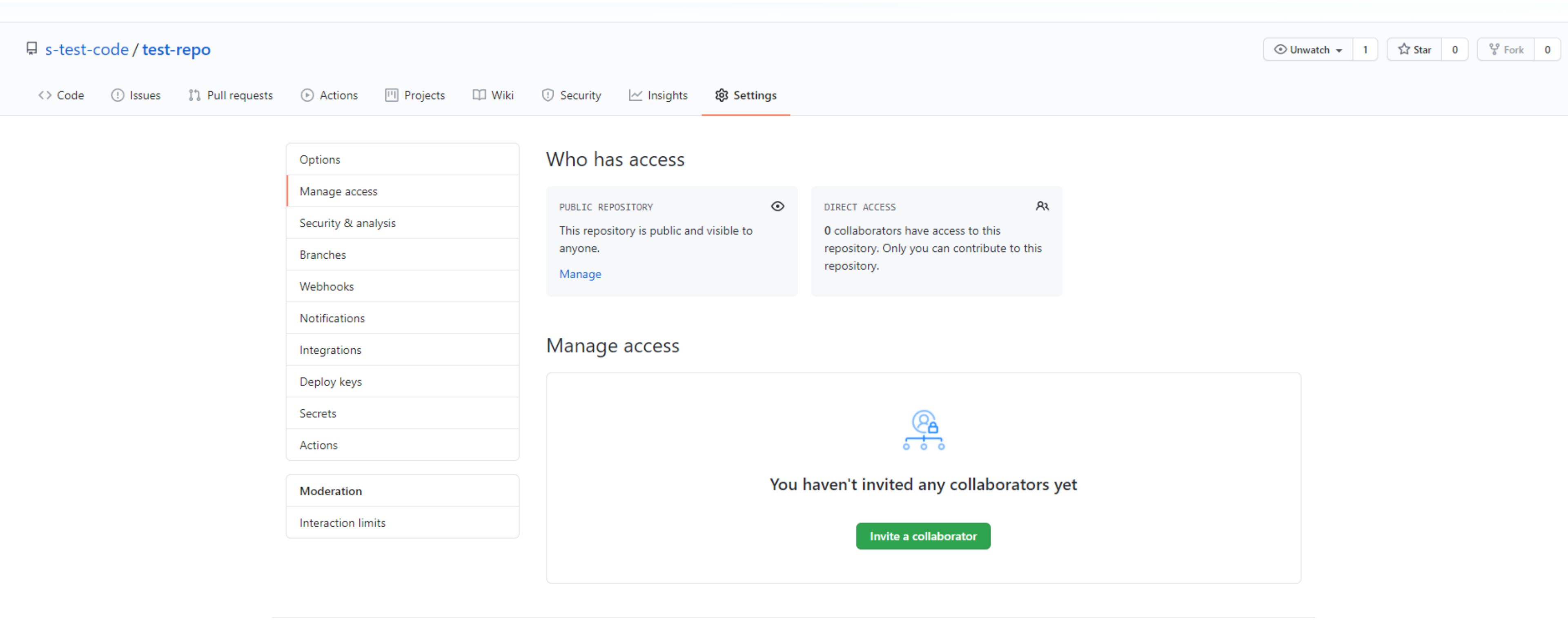
Practical – Adding a personal github page

- Make a repository the same name as your username
- Add some info
- See my one at https://github.com/coughls/coughls/blob/master/README.md (click the pen button on the righthand side of the box to see the code that underlies this)

Practical – Using GitHub & Github pages

https://lab.github.com/githubtraining/introduction-to-github

Adding collaborators



s-test-code / test-repo

Who has access

PUBLIC REPOSITORY

This repository is public and visible to anyone.

Manage

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.

Manage access

You haven't invited any collaborators yet

Useful References

- Git and github: https://kbroman.org/github_tutorial/pages/init.html
- https://www.datacamp.com/community/tutorials/git-push-pull?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=332602034352&utm_targetid=dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=1007835&gclid=EAIaIQobChMIxqLsn9rt6wIVVODtCh3Ngw16EAAYASAAEgJrk_D_BwE
- Markdown: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
- https://www.educative.io/edpresso/what-are-some-important-git-commands