Unit 1:
Version Control Tool

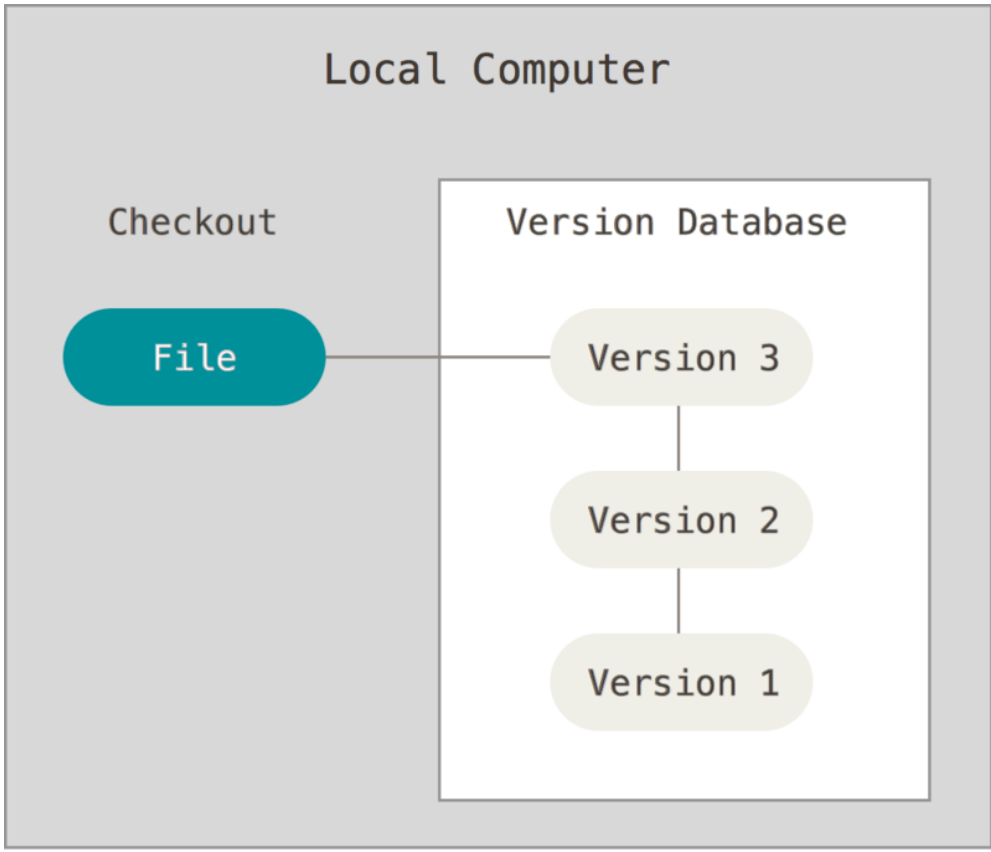
Experiment 1: To study of a version control tool: Git

What is version control?

- Version control, also known as source control, is the practice of tracking and managing changes to software code.
- Version control systems (VCS) are software tools that help software teams manage changes to source code over time.
- It allows to:
  - revert selected files back to a previous state,
  - revert the entire project back to a previous state,
  - compare changes over time,
  - see who last modified something that might be causing a problem,
  - who introduced an issue and when, and more.
- Using a VCS also generally means that if things go wrong or files are lost, they can be easily recovered.

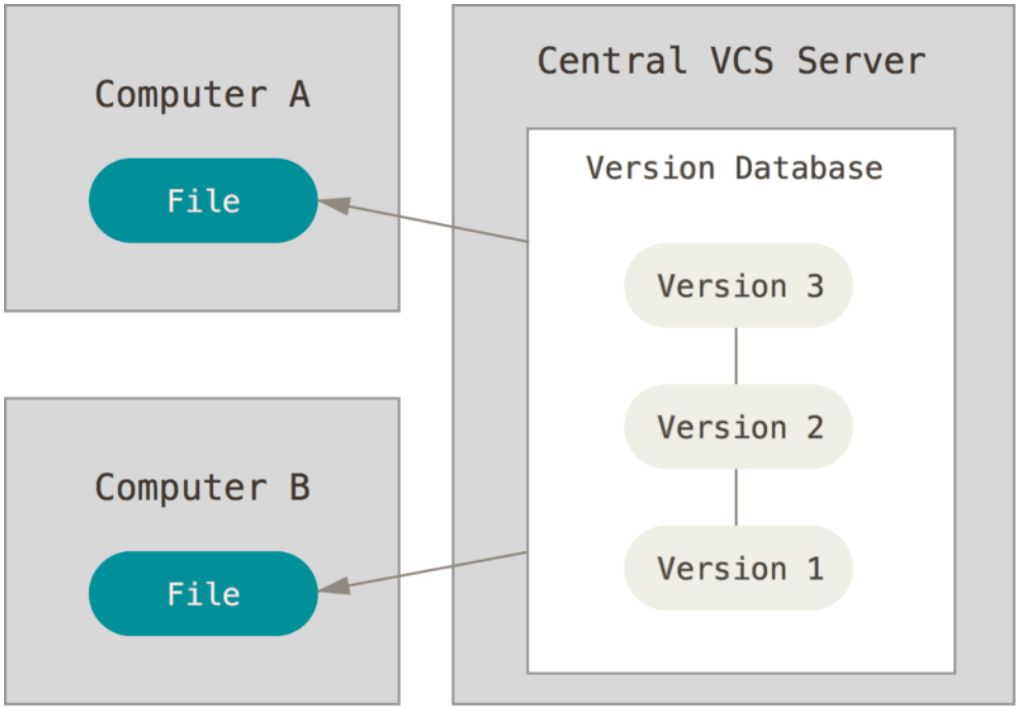
Local Version Control Systems

- Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever).
- This approach is very common because it is so simple, but it is also incredibly error prone.
- It is easy to forget which directory one is in and accidentally write to the wrong file or copy over files you don't mean to.
- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



Centralized Version Control Systems

- A major issue encountered is that there is a need to collaborate with developers on other systems.
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.
- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- For many years, this has been the standard for version control.

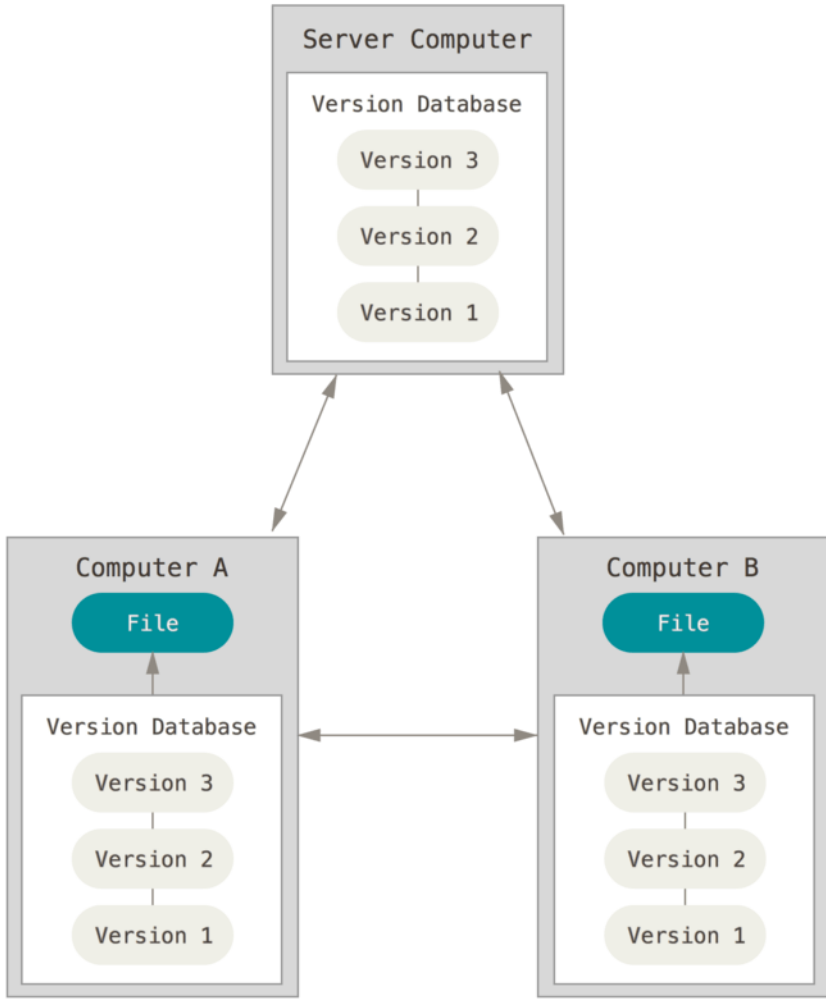


Centralized Version Control Systems

- Advantages:
  - For example, everyone knows to a certain degree what everyone else on the project is doing.
  - Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.
- Drawbacks:
  - The most obvious drawback is the single point of failure that the centralized server represents.
  - If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.
  - If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose everything.

Distributed Version Control Systems

- In a Distributed Version Control Systems (DVCS), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.
- Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.



Examples of DVCS Tools

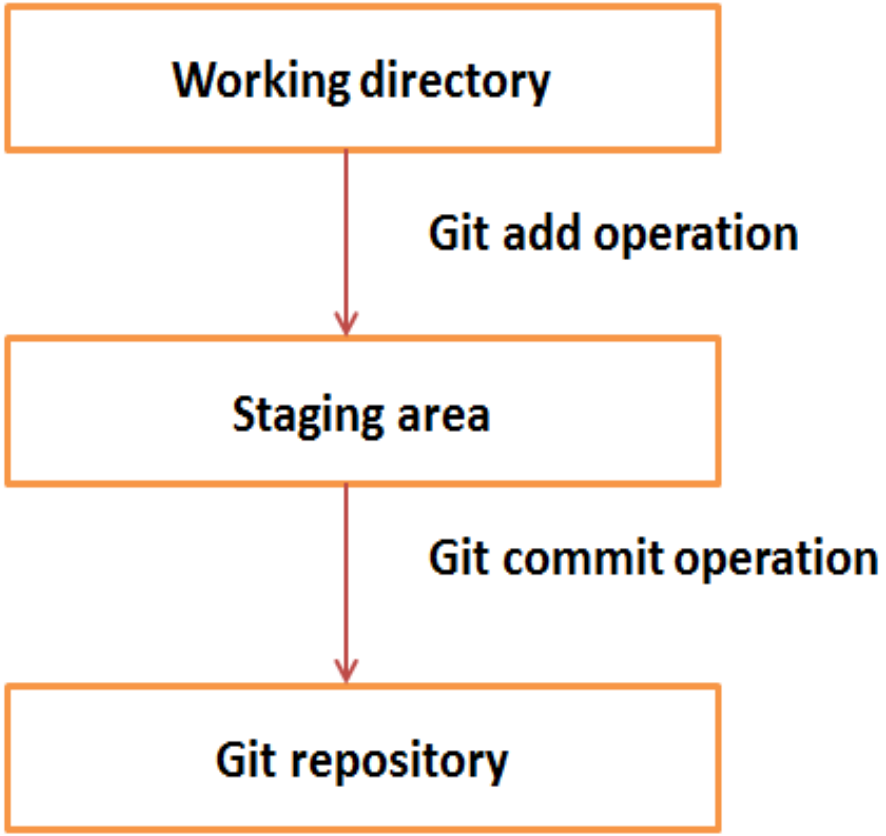
- ***Git***
- Mercurial
- Bazaar
- Darcs

Working Directory and Staging Area or Index

- ***Git*** doesn't track each and every modified file.
- Whenever you do commit an operation, Git looks for the files present in the staging area.
- Only those files present in the staging area are considered for commit and not all the modified files.

Working Directory and Staging Area or Index

- **Step 1** – You modify a file from the working directory.
- **Step 2** – You **add** these files to the staging area.
- **Step 3** – You perform **commit** operation that moves the files from the staging area. After **push** operation, it stores the changes permanently to the Git repository.



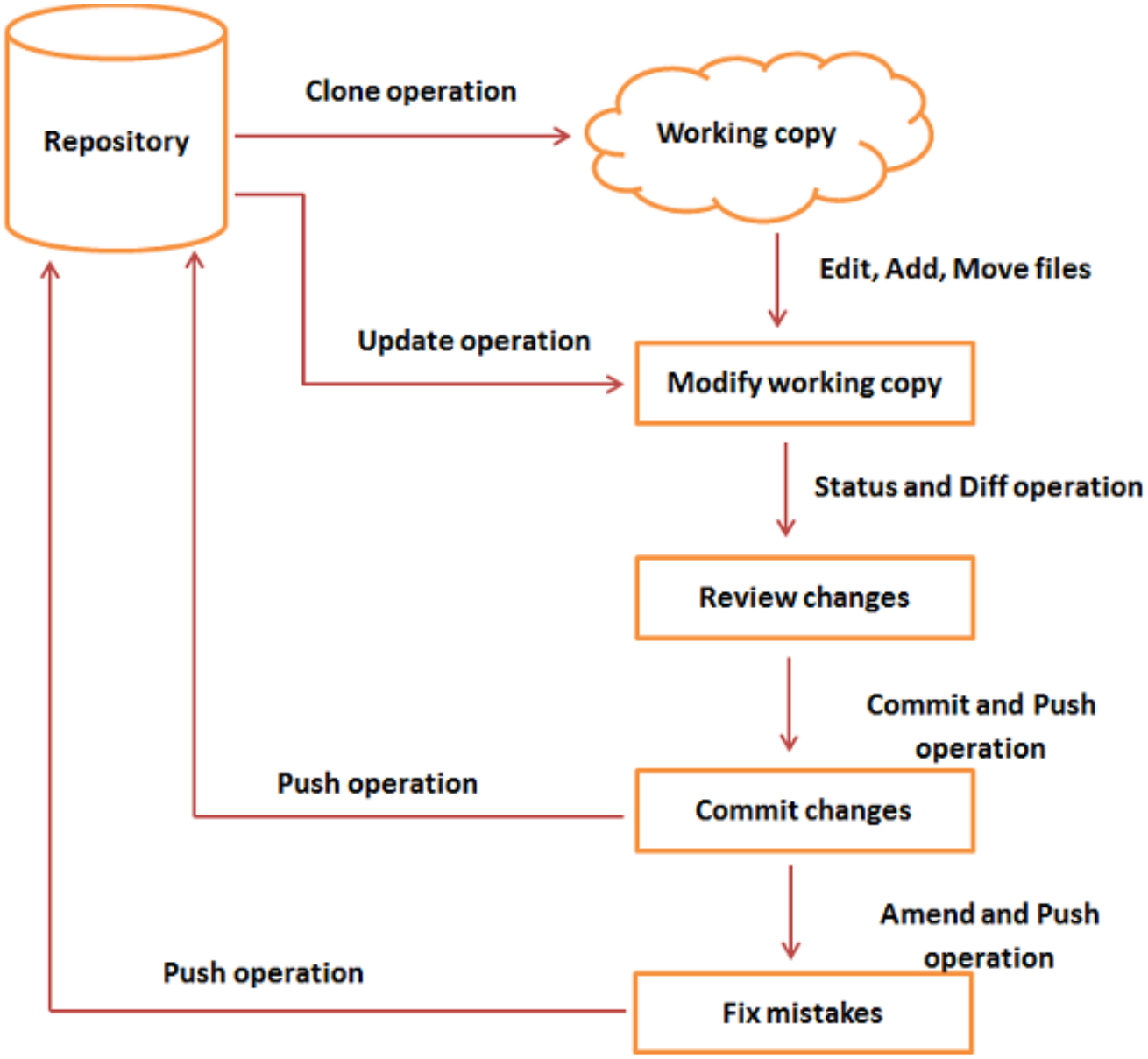
DVCS Terminologies

- **Blobs** (**B**inary **L**arge **Ob**ject): Each version of a file is represented by blob. A blob holds the file data but doesn't contain any metadata about the file. In Git, files are not addressed by names. Everything is content-addressed.
- **Trees**: Tree is an object, which represents a directory. It holds blobs as well as other sub-directories. A tree is a binary file that stores references to blobs.
- **Commits**: Commit holds the current state of the repository. You can consider a commit object as a node of the linked list.
  - Every commit object has a pointer to the parent commit object.
  - From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit.
  - If a commit has multiple parent commits, then that particular commit has been created by merging two branches.
- **Branches**: Branches are used to create another line of development. By default, Git has a master branch, which is same as trunk in Subversion.
  - Usually, a branch is created to work on a new feature.
  - Once the feature is completed, it is merged back with the master branch and we delete the branch.
  - Every branch is referenced by HEAD, which points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.

DVCS Terminologies

- **Tags**: Tag assigns a meaningful name with a specific version in the repository.
  - Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify.
  - Once a tag is created for a particular commit, even if you create a new commit, it will not be updated.
- **Clone**: Clone operation creates the instance of the repository.
  - Clone operation not only checks out the working copy, but it also mirrors the complete repository.
  - Users can perform many operations with this local repository.
- **Pull**: Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances.
- **Push**: Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository.
- **HEAD**: HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.

Git - Life Cycle



Git Commands

- '**git add**' is a command used to add a file that is in the working directory to the staging area.
- '**git commit**' is a command used to add all files that are staged to the local repository.
- '**git push**' is a command used to add all committed files in the local repository to the remote repository. So in the remote repository, all files and changes will be visible to anyone with access to the remote repository.
- '**git fetch**' is a command used to get files from the remote repository to the local repository but not into the working directory.
- '**git merge**' is a command used to get the files from the local repository into the working directory.
- '**git pull**' is command used to get files from the remote repository directly into the working directory. It is equivalent to a git fetch and a git merge .

Git V/S GitHub

- Git is an open-source, version control tool created in 2005 by developers working on the Linux operating system
- GitHub is a company founded in 2008 that makes tools which integrate with git.
- You do not need GitHub to use git, but you cannot use GitHub without using git

Install Git

- Get a github account
  - https://github.com/join?return_to=%2Fjoin%3Fref_cta%3DSign%2Bup%26ref_loc%3Dheader%2Blogged%2Bout%26ref_page%3D%252F%26source%3Dheader-home&source=login
- Download and install git
  - Please visit following link to install Git
  - https://git-scm.com/downloads
- Set up git with your user name and email.

- Ensure you have Git installed on you machine
  - git –version
- Tell Git who you are
  - $ git config --global user.name "YOUR_USERNAME"
  - $ git config --global user.email “name@xyz.com"
  - $ git config --global --list # To check the info you just provided

- Create a local Git repository
  - To use git we'll be using the terminal
  - To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command.
  - Example:

    $ cd ~/Desktop

    $ mkdir myproject

    $ cd myproject/
  - To initialize a git repository in the root of the folder, run the git init command:

    $ git init

Add a new file to the repo

- Go ahead and add a new file to the project, using any text editor you like or running a touch command.
  - `touch newfile.txt` just creates and saves a blank file named newfile.txt.
- Once you've added or modified files in a folder containing a git repo, git will notice that the file exists inside the repo.
- But, git won't track the file unless you explicitly tell it to.
- Git only saves/manages changes to files that it *tracks*, so we'll need to send a command to confirm that yes, we want git to track our new file.

  $ touch mnelson.txt

  $ ls mnelson.txt
- After creating the new file, you can use the 'git status' command to see which files git knows exist.
- What this basically says is, "Hey, we noticed you created a new file called mnelson.txt, but unless you use the 'git add' command we aren't going to do anything with it."

Add a file to the staging environment

- Add a file to the staging environment using the git add command.
- If you rerun the git status command, you'll see that git has added the file to the staging environment (notice the "Changes to be committed" line).
- $git add filename
- NOTE: To reiterate, the file has **not** yet been added to a commit, but it's about to be.

```
mnelson:myproject mnelson$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   mnelson.txt
```

Create a commit

- Create your first commit!
- Run the command
- $git commit -m "Your message about the commit“
- Check the commit using the following command
- $git status

```
mnelson:myproject mnelson$ git commit -m "This is my first commit!"
[master (root-commit) b345d9a] This is my first commit!
 1 file changed, 1 insertion(+)
 create mode 100644 mnelson.txt
```

Create a new repository on GitHub

- If you only want to keep track of your code locally, you don't need to use GitHub.
- But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.
- To create a new repository on GitHub, log in and go to the GitHub home page.
- You can find the “New repository” option under the “+” sign next to your profile picture, in the top right corner of the navbar
- After clicking the button, GitHub will ask you to name your repo and provide a brief description
- GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally.
- In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the '....or push an existing repository from the command line' section

Push a branch to GitHub

- Now we'll push the commit in your branch to your new GitHub repo.
- This allows other people to see the changes you've made.
- If they're approved by the repository's owner, the changes can then be merged into the primary branch.
- To push changes onto a new branch on GitHub, you'll want to
  - **git remote add origin *<REMOTE_URL>*** # Sets the new remote
  - **git remote -v** # Verifies the new remote URL
  - Push the changes in your local repository to GitHub.

    Run: git push origin yourbranchname
- GitHub will automatically create the branch for you on the remote repository
- NOTE: what that "origin" word means in the command above? What happens is that when you clone a remote repository to your local machine, git creates an alias for you. In nearly all cases this alias is called "origin."

- If you refresh the GitHub page, you'll see note saying a branch with your name has just been pushed into the repository.
- You can also click the 'branches' link to see your branch listed there.

See the Changes you made to your file:

- Once you start making changes on your files and you save them, the file won't match the last version that was committed to git. To see the changes you just made:
  - $ git diff # To show the files changes not yet staged

View Commit History:

- You can use the **git log** command to see the history of commit you made to your files:
  - $ git log

Each time you make changes that you want to be reflected on GitHub, the following are the most common flow of commands:

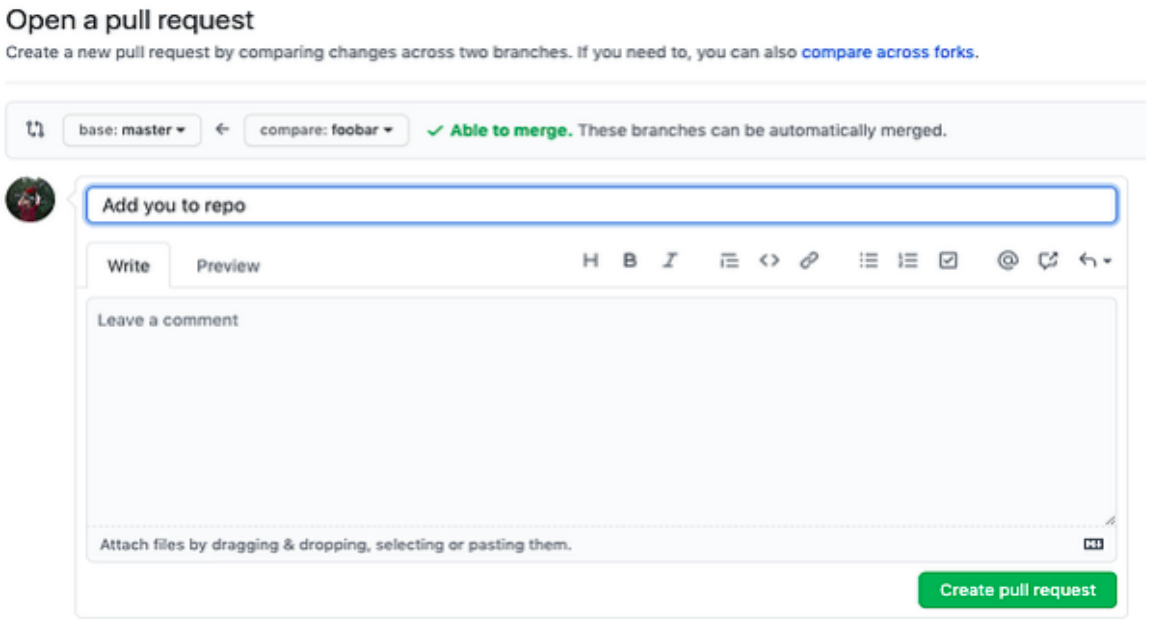
```
$ git add .
$ git status # Lists all new or modified files to be committed
$ git commit -m "Second commit"
$ git push -u origin master
```

Summary of commands to push changes in local repository (your computer) to remote repository (GitHub)

- To create local Git repository
  - git init
- To create/edit file
  - nano filename.txt (to save and exit ctrl+x→ Y → enter
- To add one file to staging area
  - git add filename.txt
- To add multiple changes/files to staging area
  - git add .
- Check status of local repository
  - git status
- To commit changes to local Git repository
  - git commit –m “your comment”
- To add remote repository url
  - git remote add origin https://github.com/username
- To check remote repository connection
  - git remote –v
- To send changes made in local repository to remote repository
  - git push –u origin master

Create a pull request (PR)

- A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code.
- It allows them to review the code and make sure it looks good before putting your changes on the primary branch.
- **$ git pull origin master**
- This is what the PR page looks like before you've submitted it:



Get changes on GitHub back to your computer

- The repo on GitHub looks a little different than what you have on your local machine.
- For example, the commit you made in your branch and merged into the primary branch doesn't exist in the primary branch on your local machine.
- In order to get the most recent changes that you or others have merged on GitHub, use the git pull origin master command (when working on the primary branch).
- In most cases, this can be shortened to “git pull”.

Clone a file available in GitHub

- Locate to the directory you want to clone the repo. Copy the link of the repository you want and enter the following:
- $ git clone *remote_repository_URL*
- *Try*

Feel free to go ahead and clone the repo below using: https://github.com/Gothamv/MuskCult

References

- https://www.atlassian.com/git/tutorials/what-is-version-control
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/
- https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners
- Step-by-step tutorial: https://github.com/Gothamv/MuskCult