Git for Version Control

(Git)

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

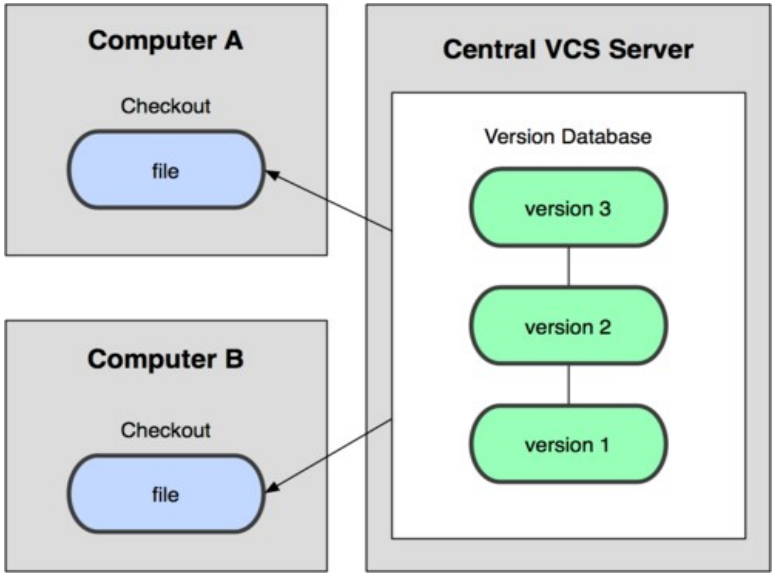
- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

Centralized VCS

- In Subversion, CVS, Perforce, etc.
  A central server repository (repo)
  holds the "official copy" of the code
  - the server maintains the sole
    version history of the repo

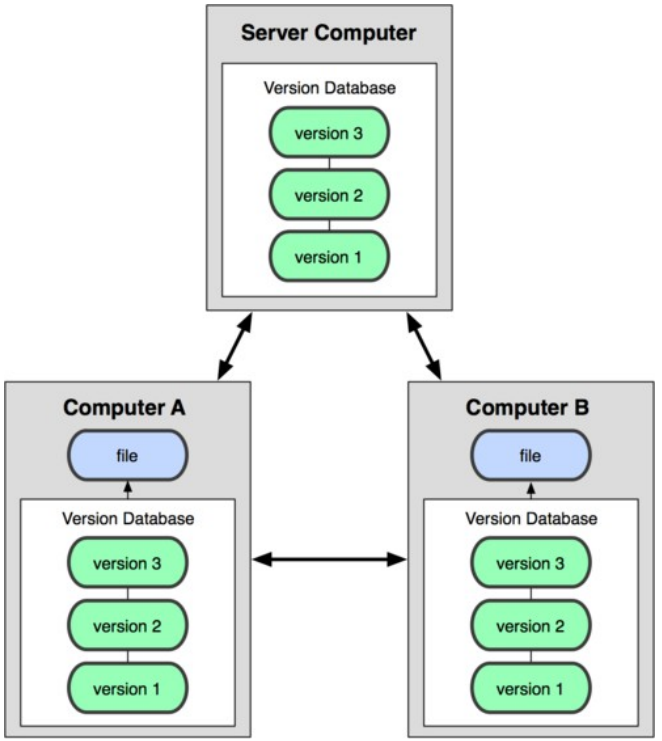
- You make "checkouts" of it
  to your local copy
  - you make local modifications
  - your changes are not versioned

- When you're done, you
  "check in" back to the server
  - your checkin increments the repo's version



Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout" from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from *local* repo
  - commit changes to *local* repo
  - local repo keeps version history

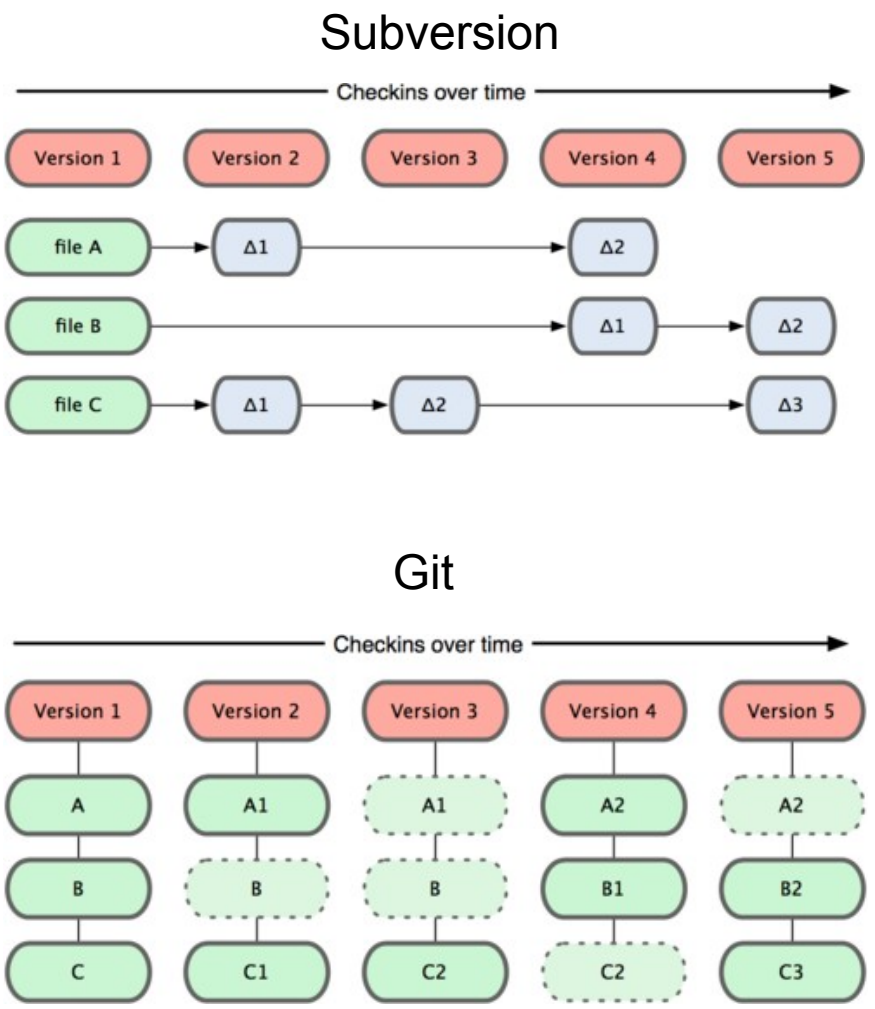


- When you're ready, you can "push" changes back to server

Git snapshots

- Centralized VCS like Subversion track version data on each individual file.

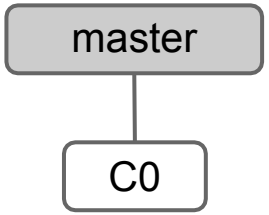
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.



A Git

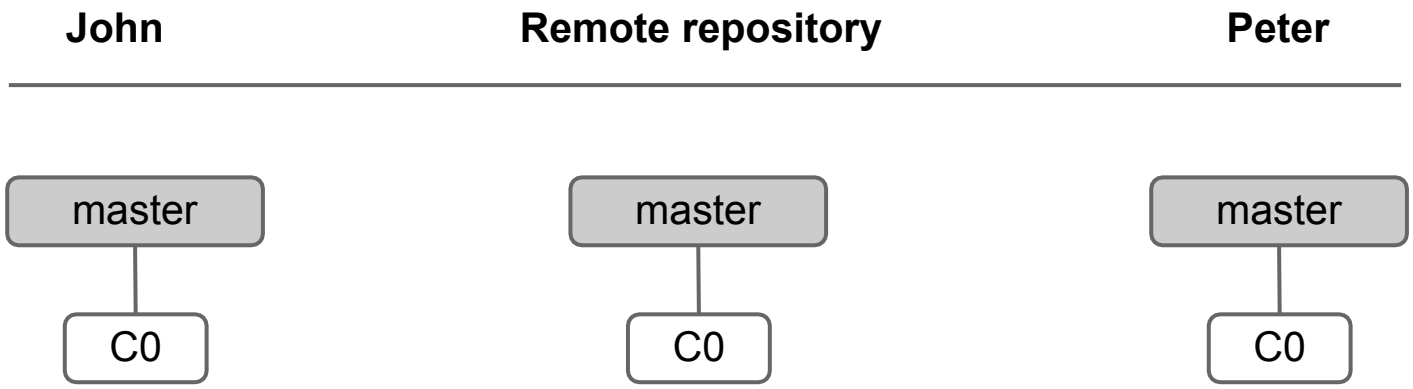
1. A new git is initialized as a remote repository

John **Remote repository** **Peter**



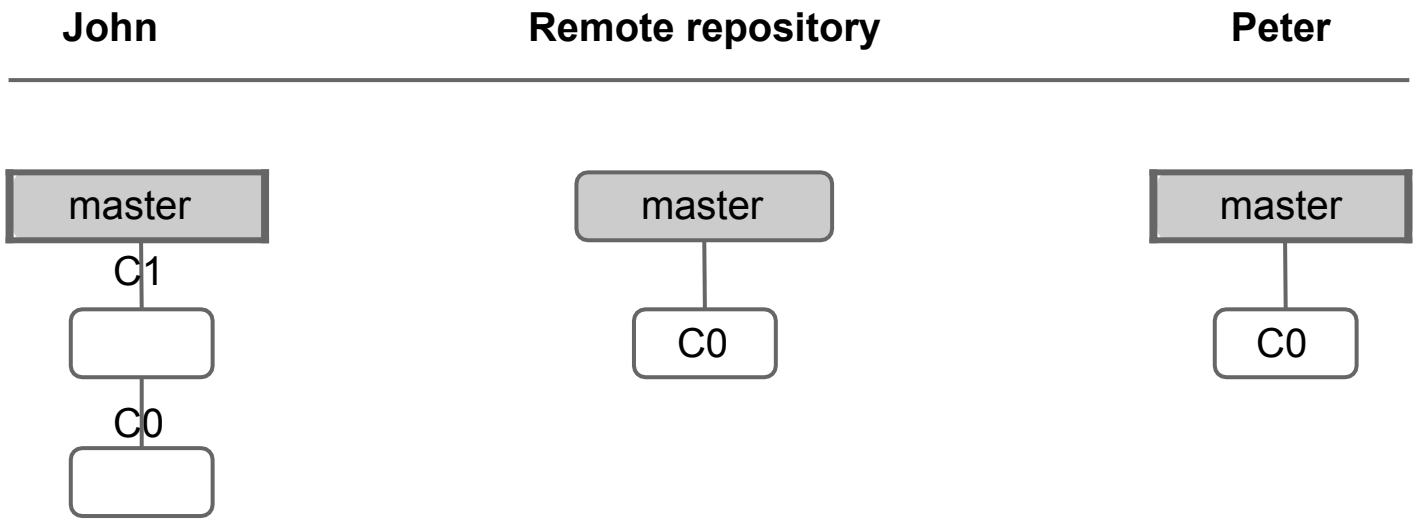
A Git

2. John and Peter clone the git repository



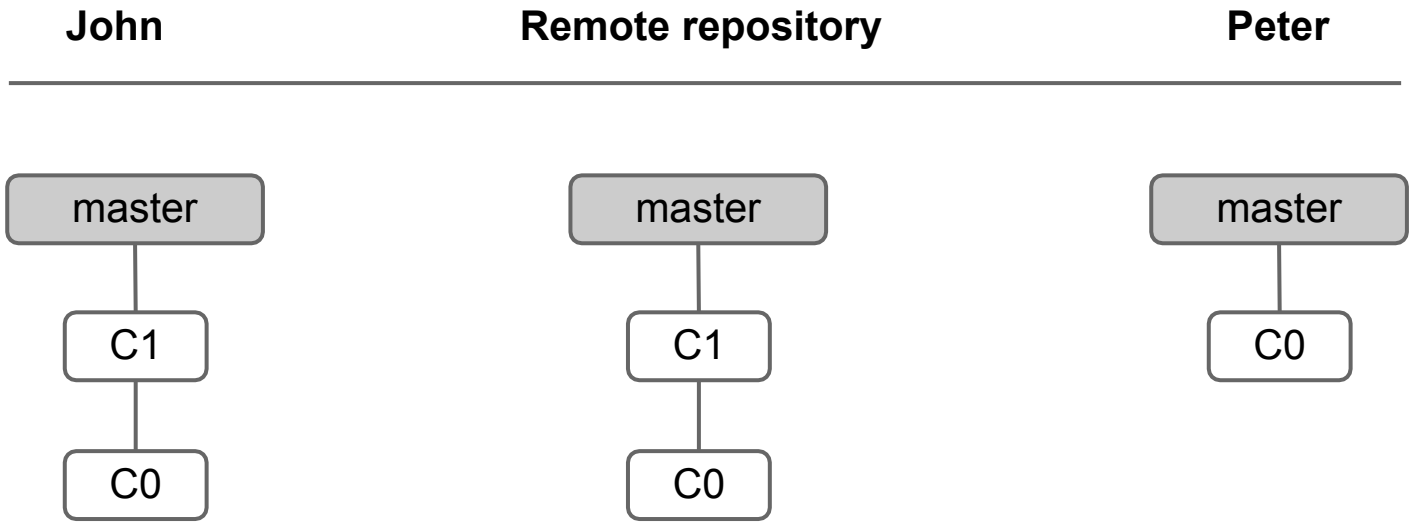
A Git

3. John does a commit



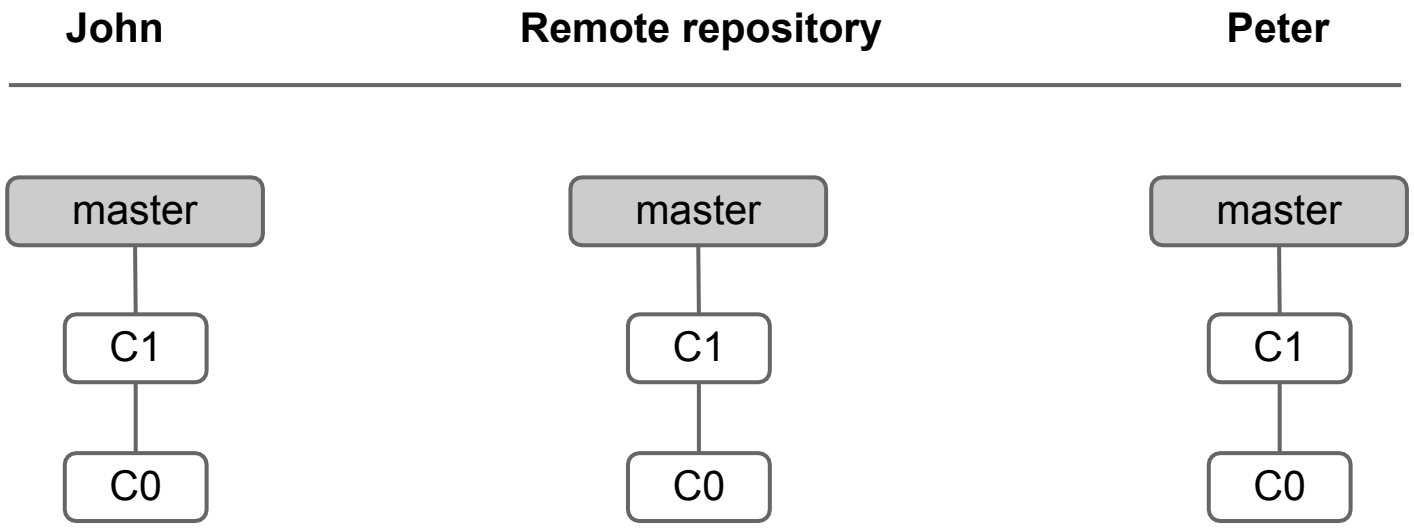
A Git

4. John does a push



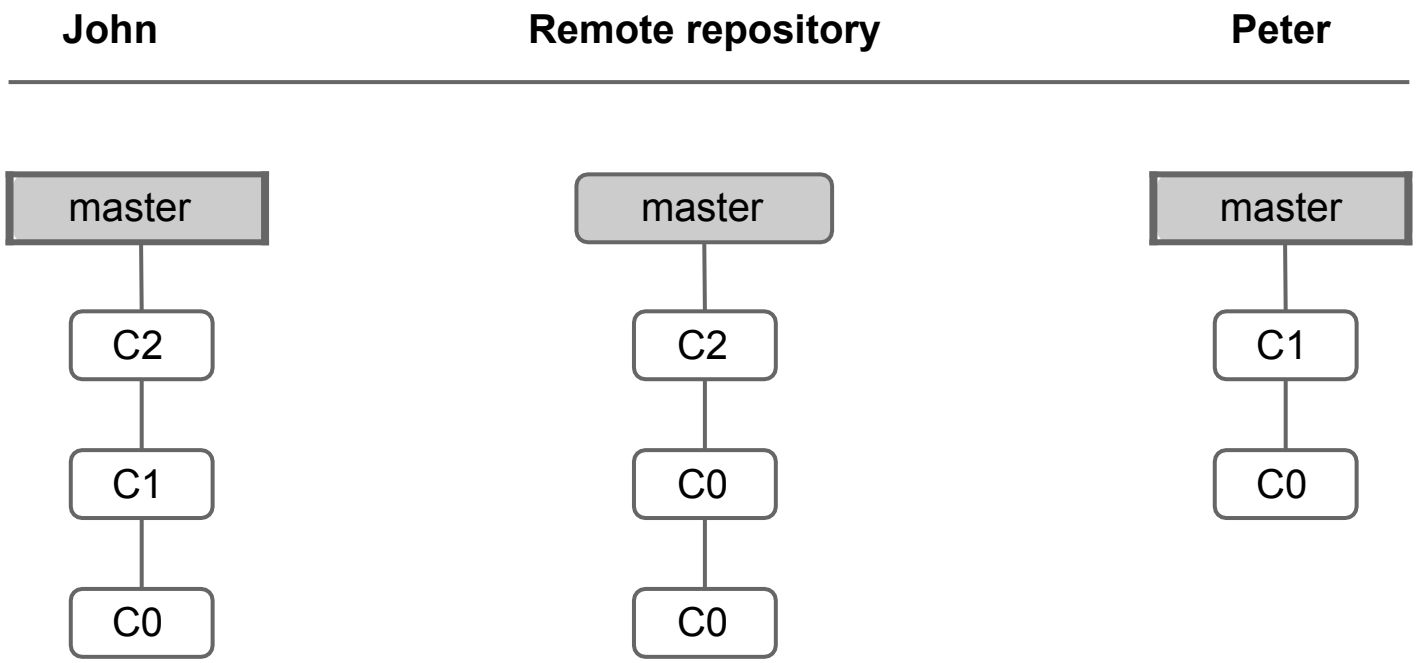
A Git

5. Peter does a pull



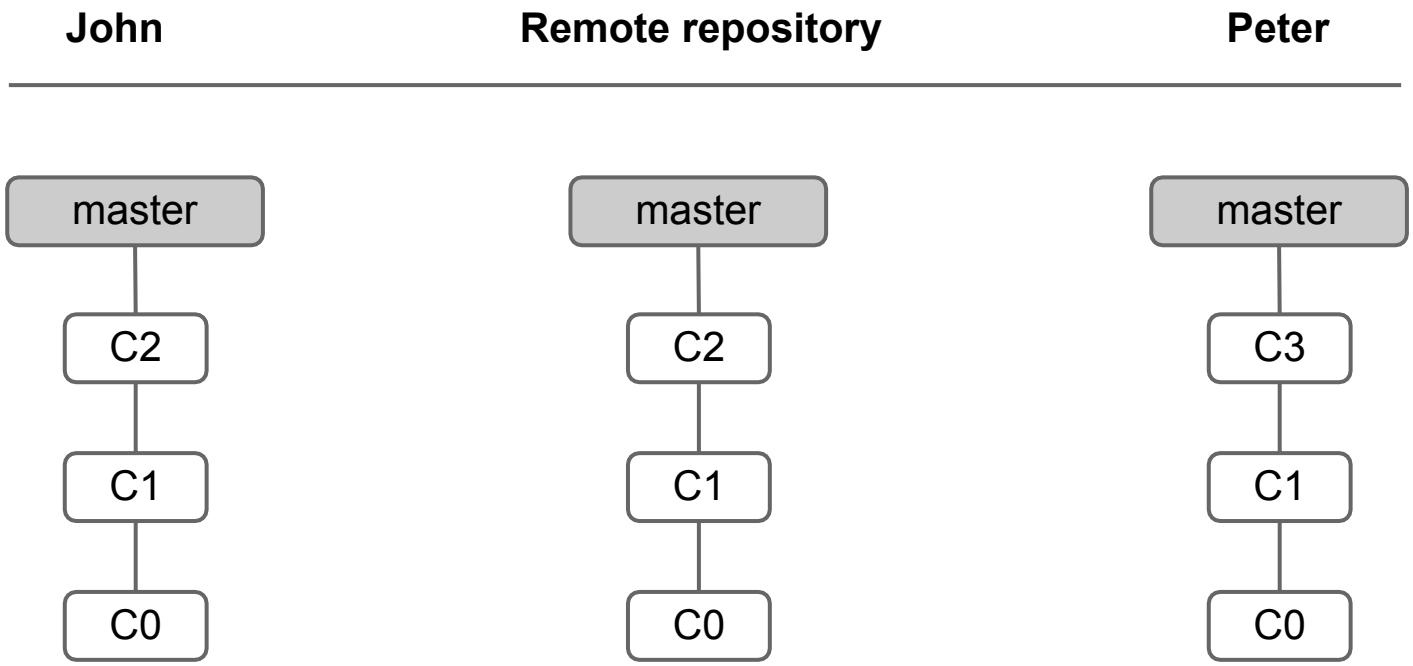
A Git

6. John does a commit & push



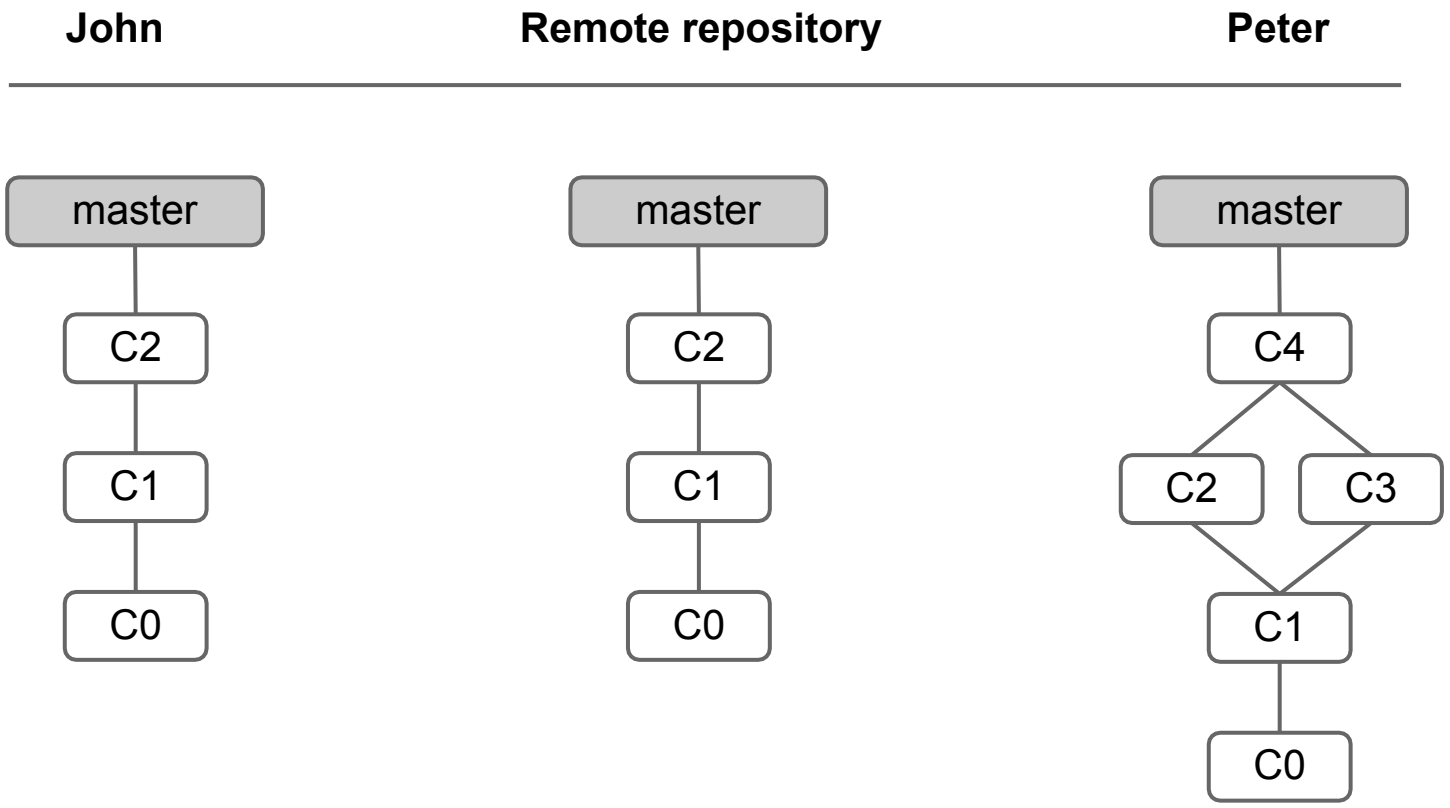
A Git

7. Peter does a commit



A Git

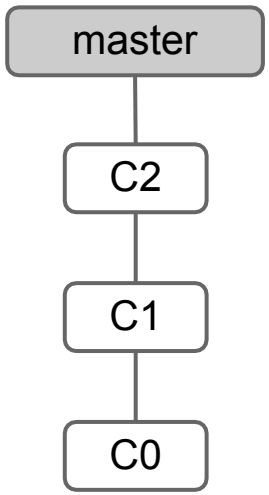
8. Peter does a pull (fetch & merge)

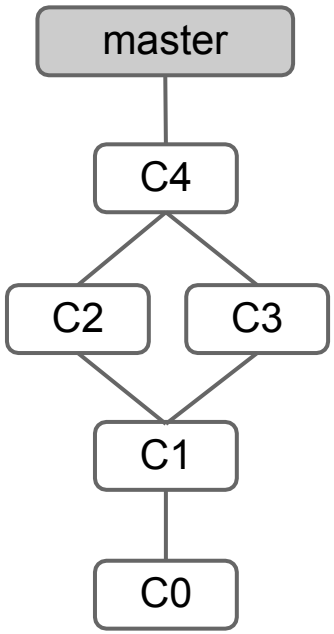


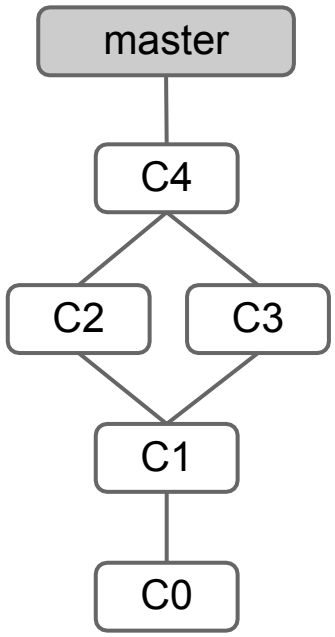
A Git

9. Peter does a push

John	Remote repository	Peter



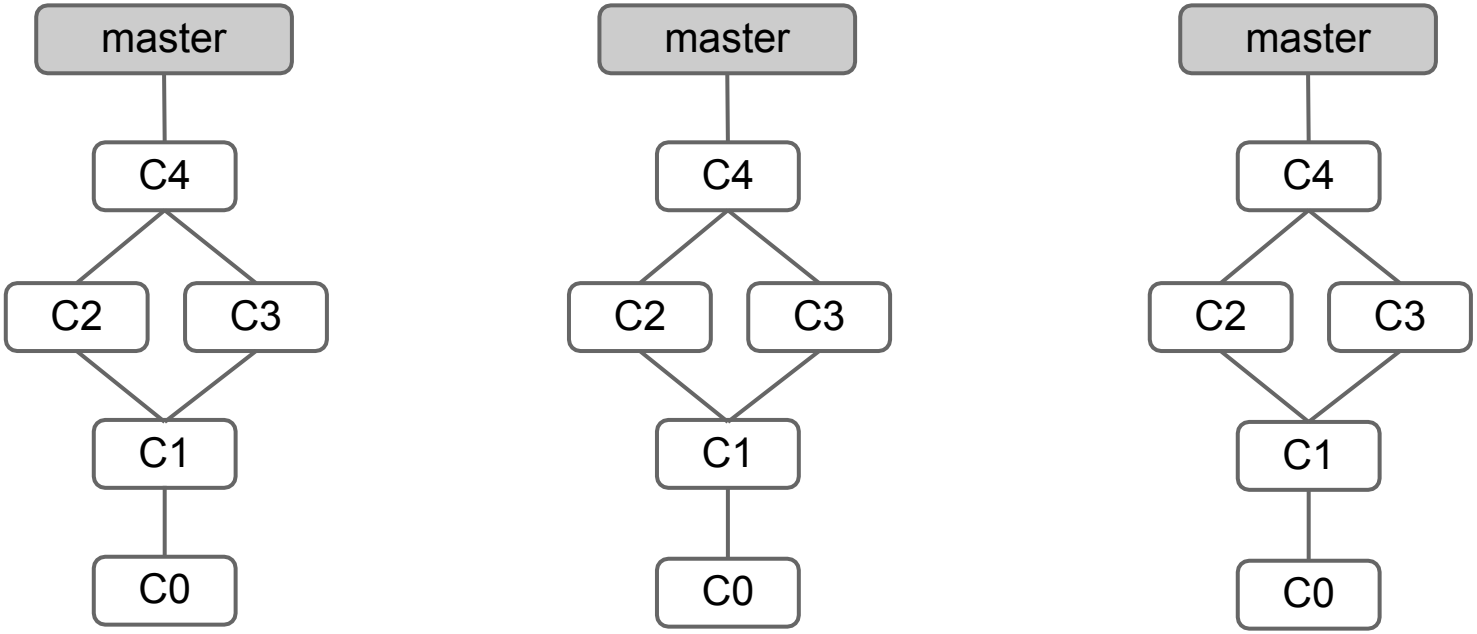




A Git

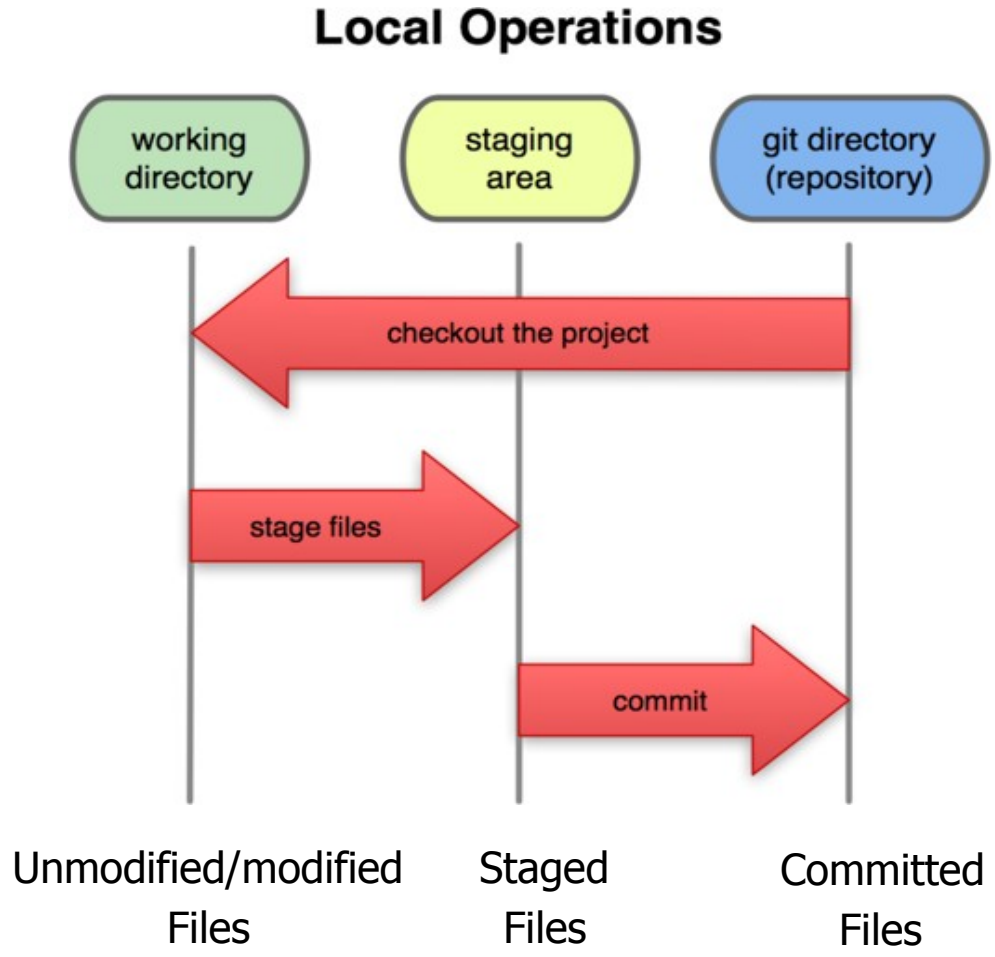
10. John does a pull

John **Remote repository** **Peter**



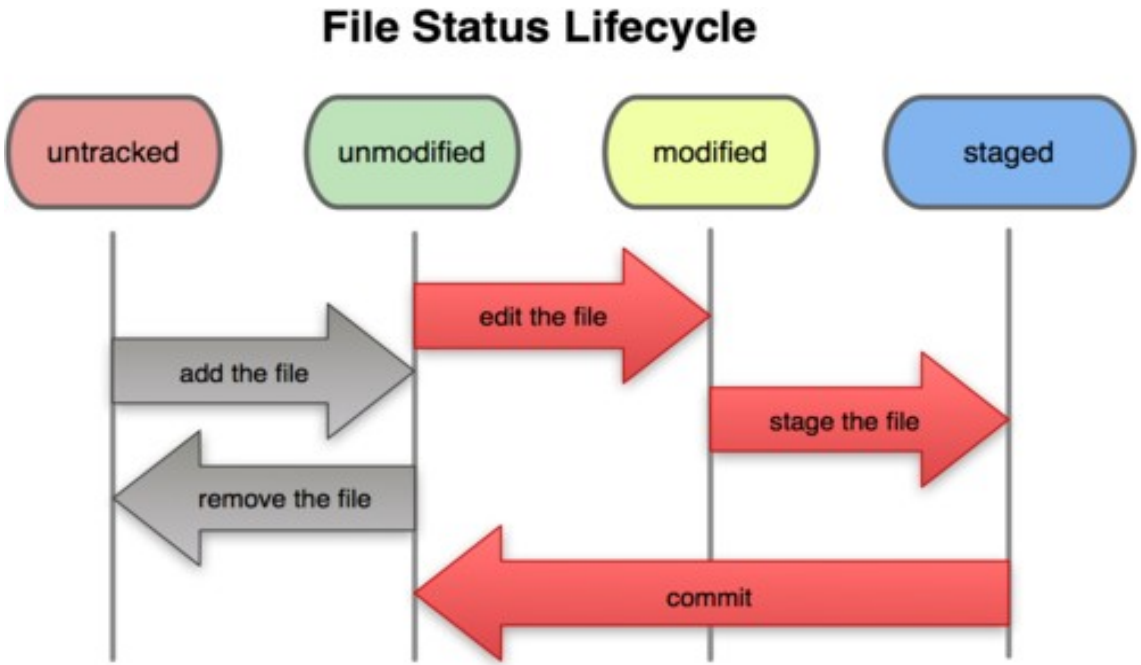
Local git

- In your local copy on git, files can be:
  - In your local repo
    - (committed)
  - Checked out and modified, but not yet committed
    - (working copy)
  - Or, in-between, in a **"staging" area**
    - Staged files are ready to be committed.
    - A commit saves a snapshot of all staged state.



Basic Git

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



Initial Gitconfiguration

- Set the name and email for Git to use when you commit:
  - `git config --global user.name ""`
  - `git config --global user.email @gmail.com`

Creating a Git

Two common scenarios: (only do one of these)

- To create a new **local Git repo** in your current directory:
  - `git init`
    - This will create a `.git` directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - `git add` *`filename`*
  - `git commit -m "`*`commit message`*`"`

- To **clone a remote repo** to your current directory:
  - `git clone` *`url  localDirectoryName`*
    - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual local repo)

Git commands

command	description
git clone ***url [dir]***	copy a Git repository so you can add to it
git add ***file***	adds file contents to the staging area
git commit	records a snapshot of the staging area
git status	view the status of your files in the working directory and staging area
git diff	shows diff of what is staged and what is modified but unstaged
git help [***command***]	get help info about a particular command
git pull	fetch from a remote repo and try to merge into the current branch
git push	push your new branches and data to a remote repository
others: init, reset, branch, checkout, merge, log, tag	

Add and commit a file

- The first time we ask a file to be tracked, *and every time before we commit a file*, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
    - Takes a snapshot of these files, adds them to the staging area.
- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
  - All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or
  - To see what is modified but unstaged:
  - `git diff`

- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

    - `git log -5` (to show only the 5 most recent updates), etc.

Branching and merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - `git branch` *`name`*
- To list all local branches: (* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout` *`branchname`*
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge` *`branchname`*

Interactionw/ remote repo

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)

- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`

- To put your changes from your local repo in the remote repo:
  - `git push origin master`

What is Git & Github ?

Git is an example of **version control**

Version control is a system that records changes to a file or set of files and helps us recall specific versions later if needed. E.g. Subversion (SVN), CVS etc

It allows you to :

- Revert files or the whole project to an earlier state
- Compare changes over time
- See who modified what?
- Control modifications by collaborators with the permission of admin/owners

- it is a version control tool that will allow you to perform all kinds of operations to fetch data from the central server or push data to it whereas GitHub is a core hosting platform for version control collaboration. GitHub is a company that allows you to host a central repository in a remote server.

Github

Personal user account

Organization account

PUBLIC | PRIVATE

REMOTE REPOSITOR

PUBLIC | PRIVATE

- Unlimited public repositories and collaborators on all plans
- Limited Private repositories
- Ability to add unlimited repository collaborators
- Public repositories are open to view and copy but not commit changes.

SYNC

- Organizations are great for that need multiple owners & admins.
- Limited private repositories (> Personal)
- Team-based access permissions
- Unlimited owners, admins, & collaborators using teams

CLONE TO GET LOCAL REPOSITORY

Important Concepts for Github Users

Creating a repo

Creating a repository for multiple people to work together

Master in a repository

This is the final version that is considered ready to use by anybody in the team or outside if repository is public.

Creating a Branch

- Create a branch in your project, for an environment where you can try out new ideas.
- Changes you make on a branch don't affect the master unless pull request is accepted.
- Changes committed to branch reflects for you to keep track of different versions

Adding Commits

- Keeps track of your progress as you work on a branch or master.
- Creates a transparent history that others can follow to understand what you've done and why.

Forking a repository

- It creates a copy for you to work on independently without any changes to theirs.
- Submit a pull request to owner so that the owner can incorporate changes.

Concepts for Github Users

Pull requests

- Pull Requests initiates discussion about your commits or changes made to a code.
- See exactly what changes would be merged if pull request is accepted.
- Use GitHub's @mention system in your Pull Request message to ask for feedback from specific people or teams, or for someone to review your work

Issues

- Highlight bugs or issues with codes that need rectification.
- Issues remain open unless resolved.
- Can be filtered, Can be labeled as bug/enancement/ question/help wanted etc
- @mention can be used to notify someone

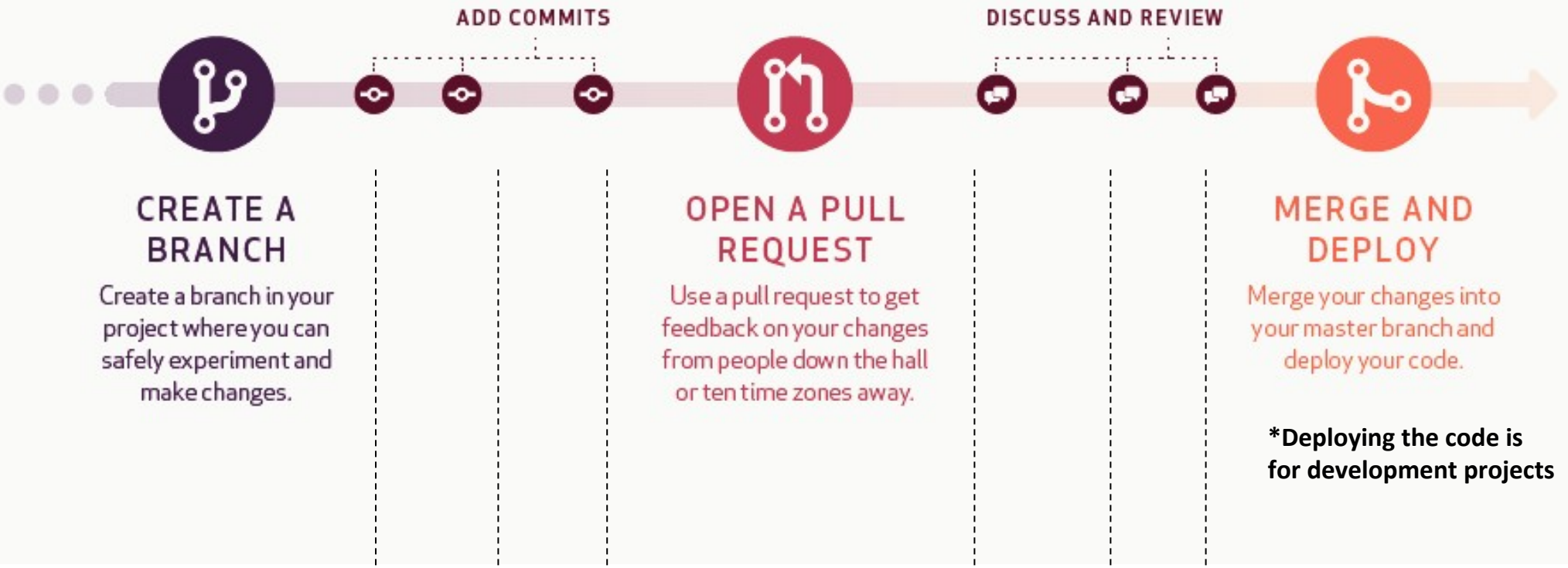
Markdown syntax

- Markdown is a way to style text on the web.
- Available in descriptions and comments of Issues and Pull Requests. These include @mentions as well as references to SHA-1 hashes, Issues, and Pull Requests

Watch and Star

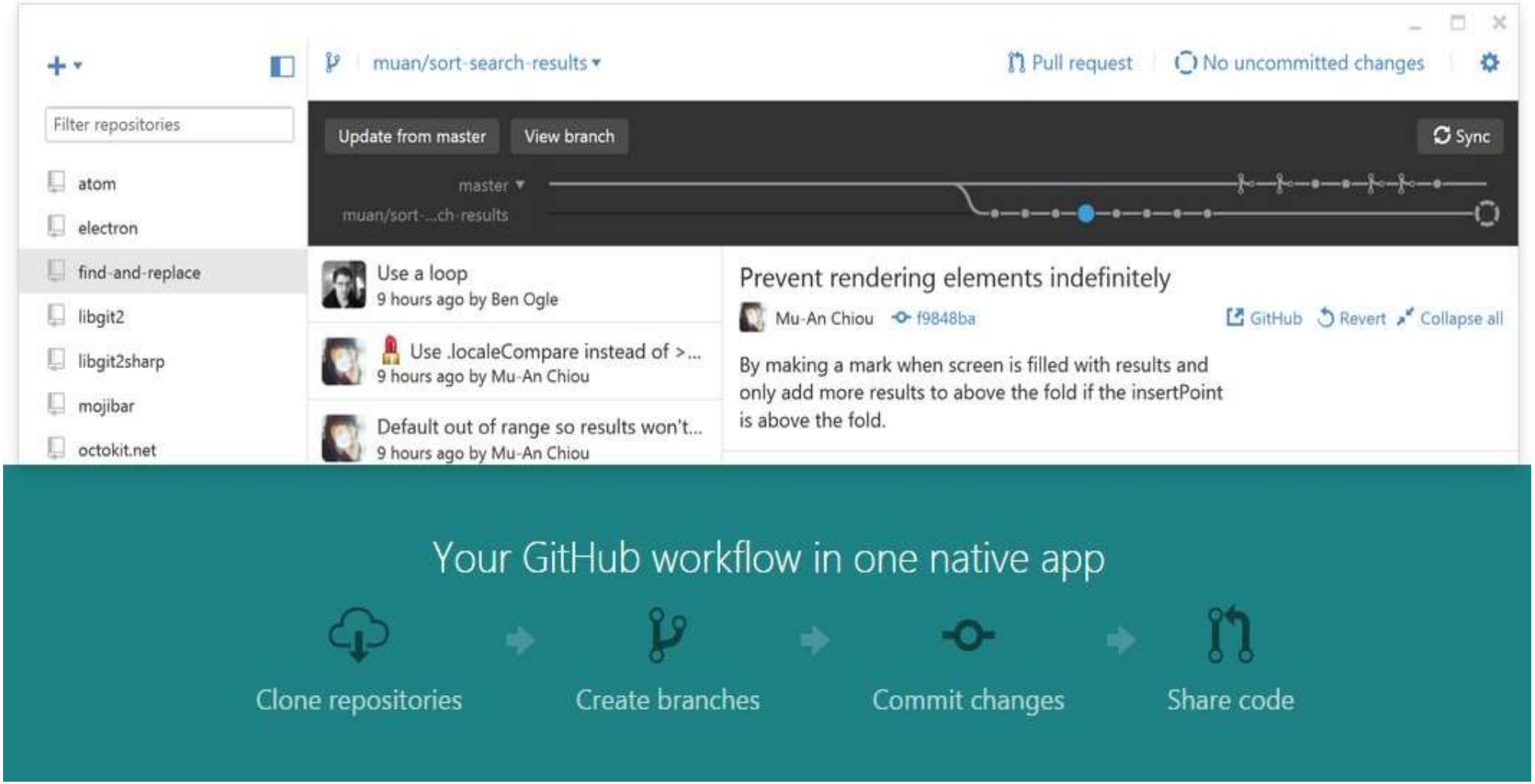
Watch notifies us of all conversations over and above your @mentions, commits, comments on discussion. Star will favorite it but not show on your dashboards like watch

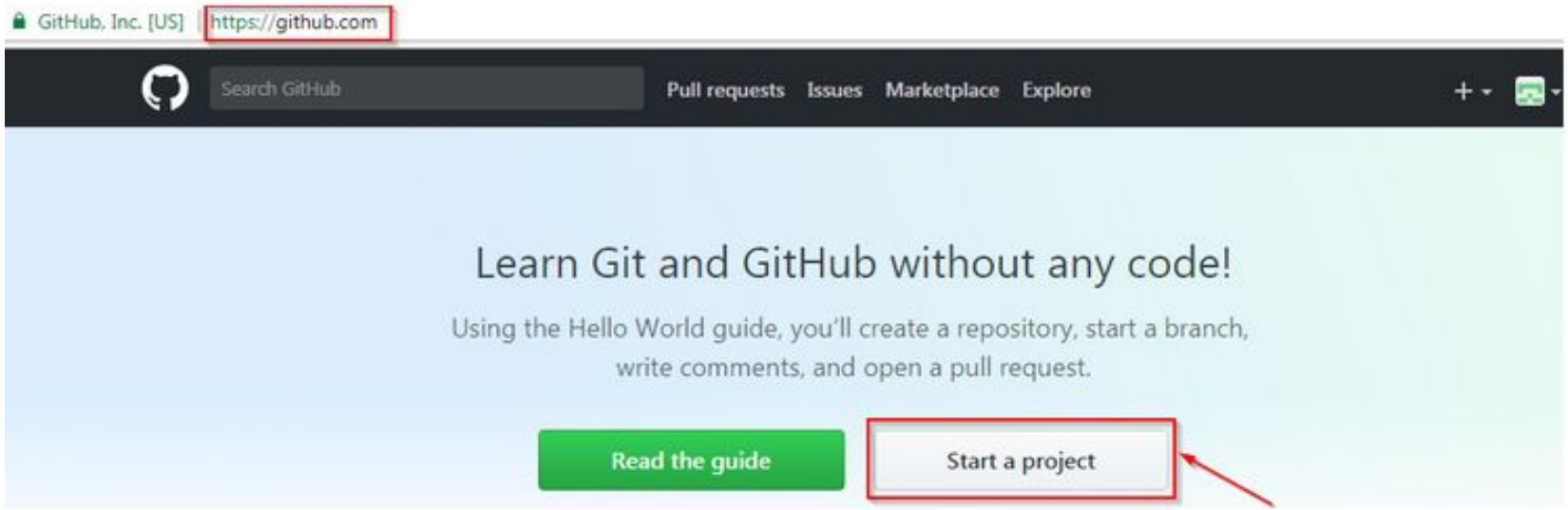
Understanding Github Workflow



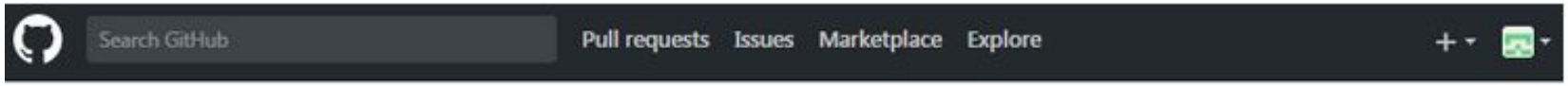
Github Desktop Demo

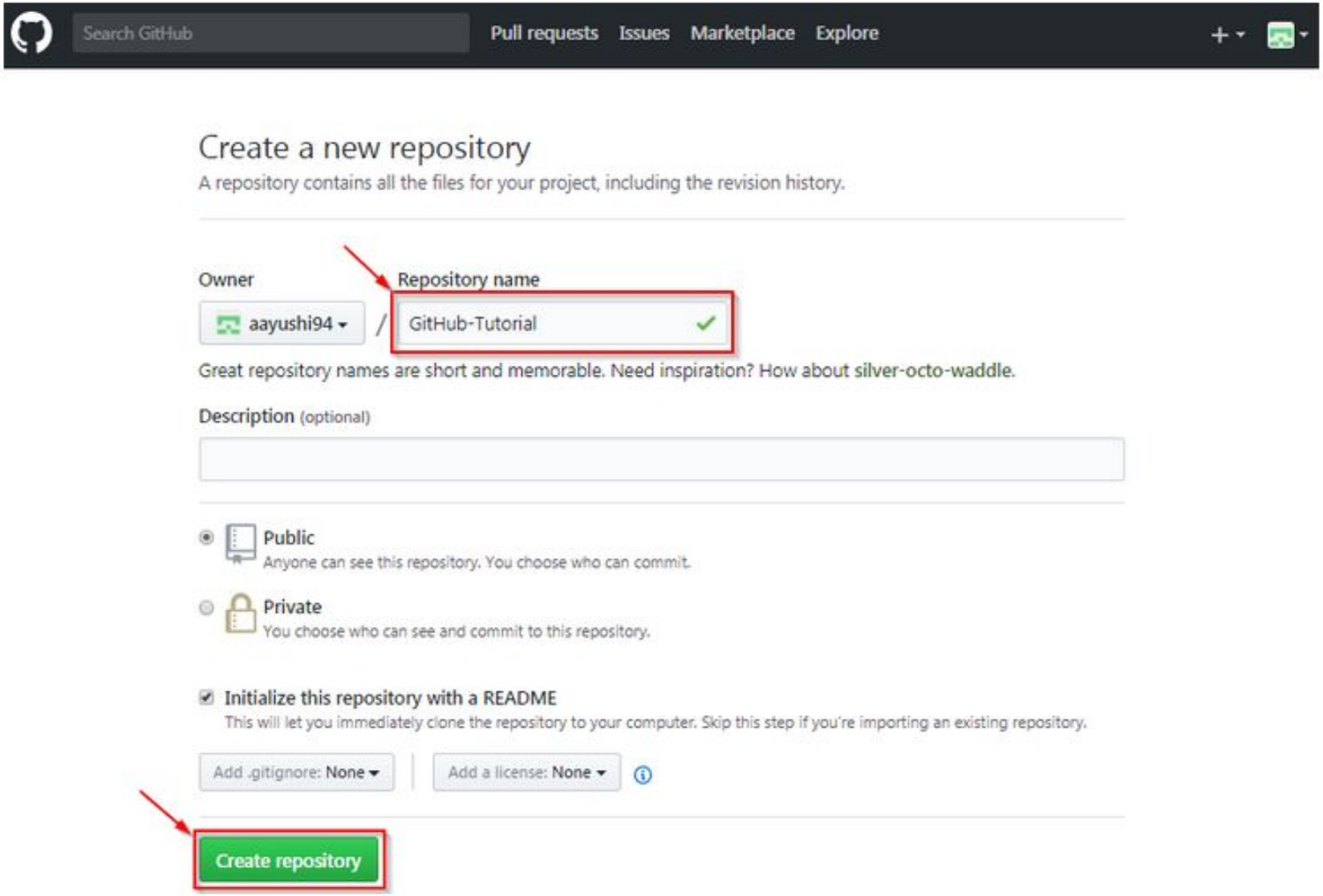
Link to download Github Desktop : https://desktop.github.com/

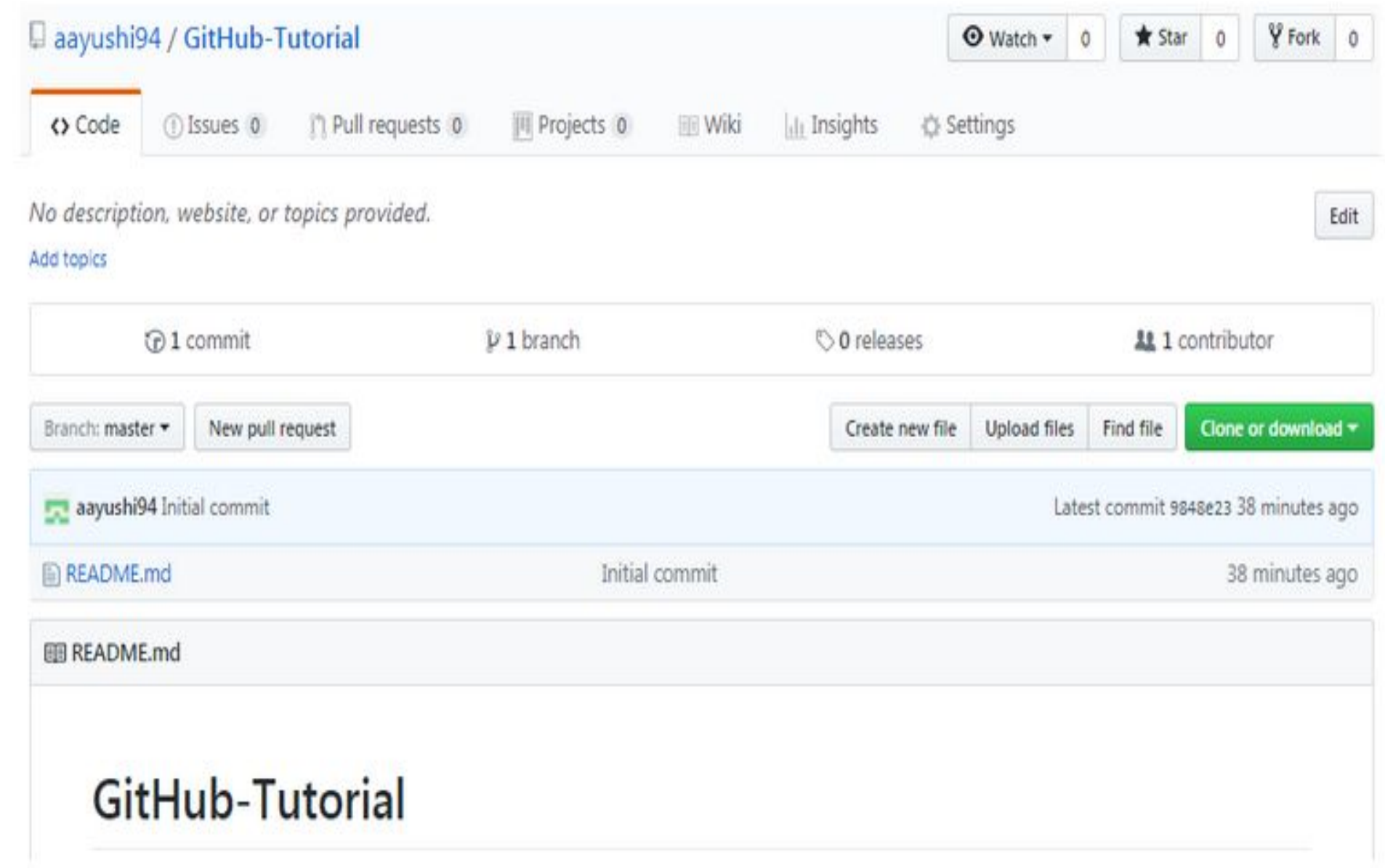


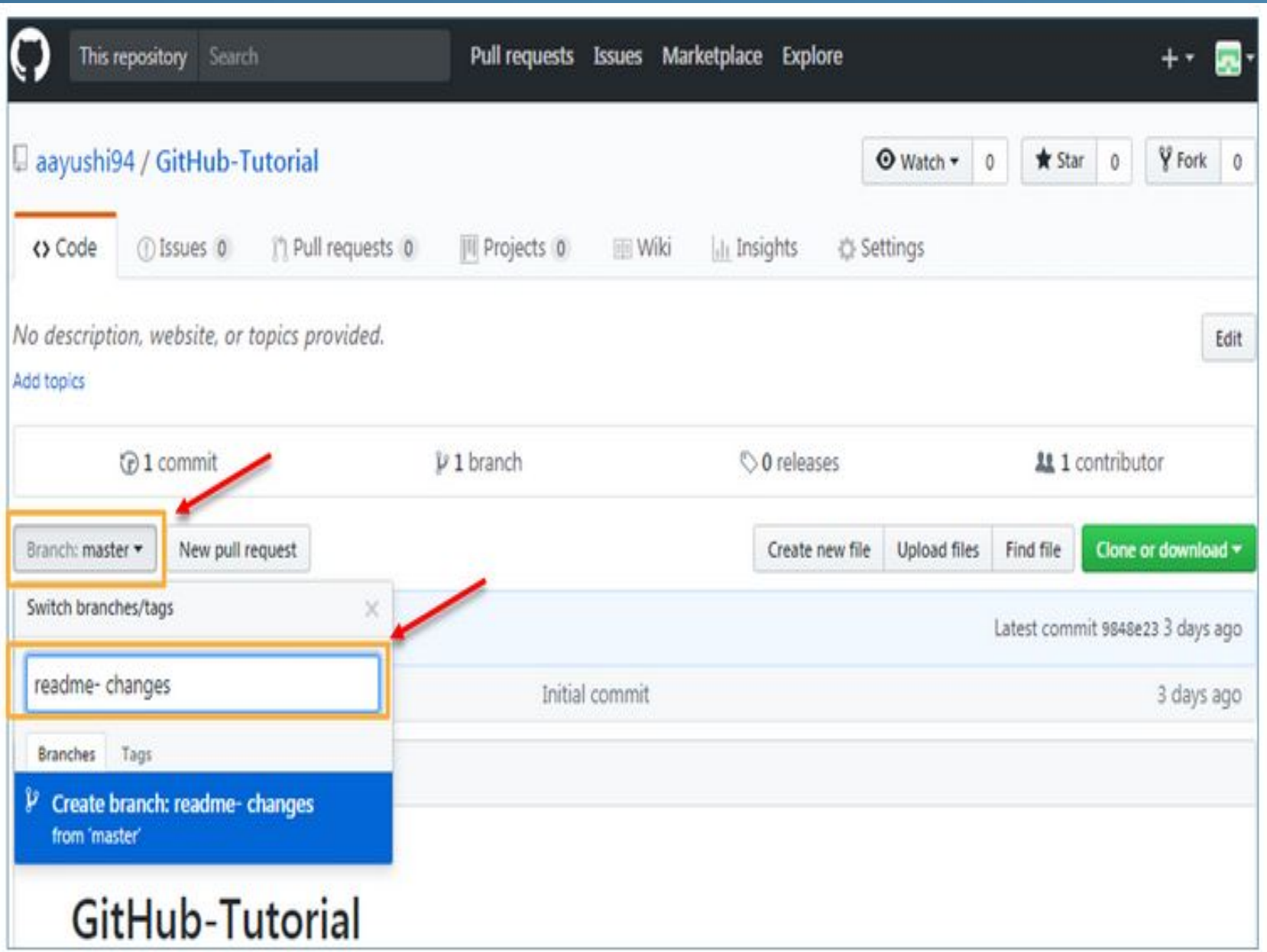


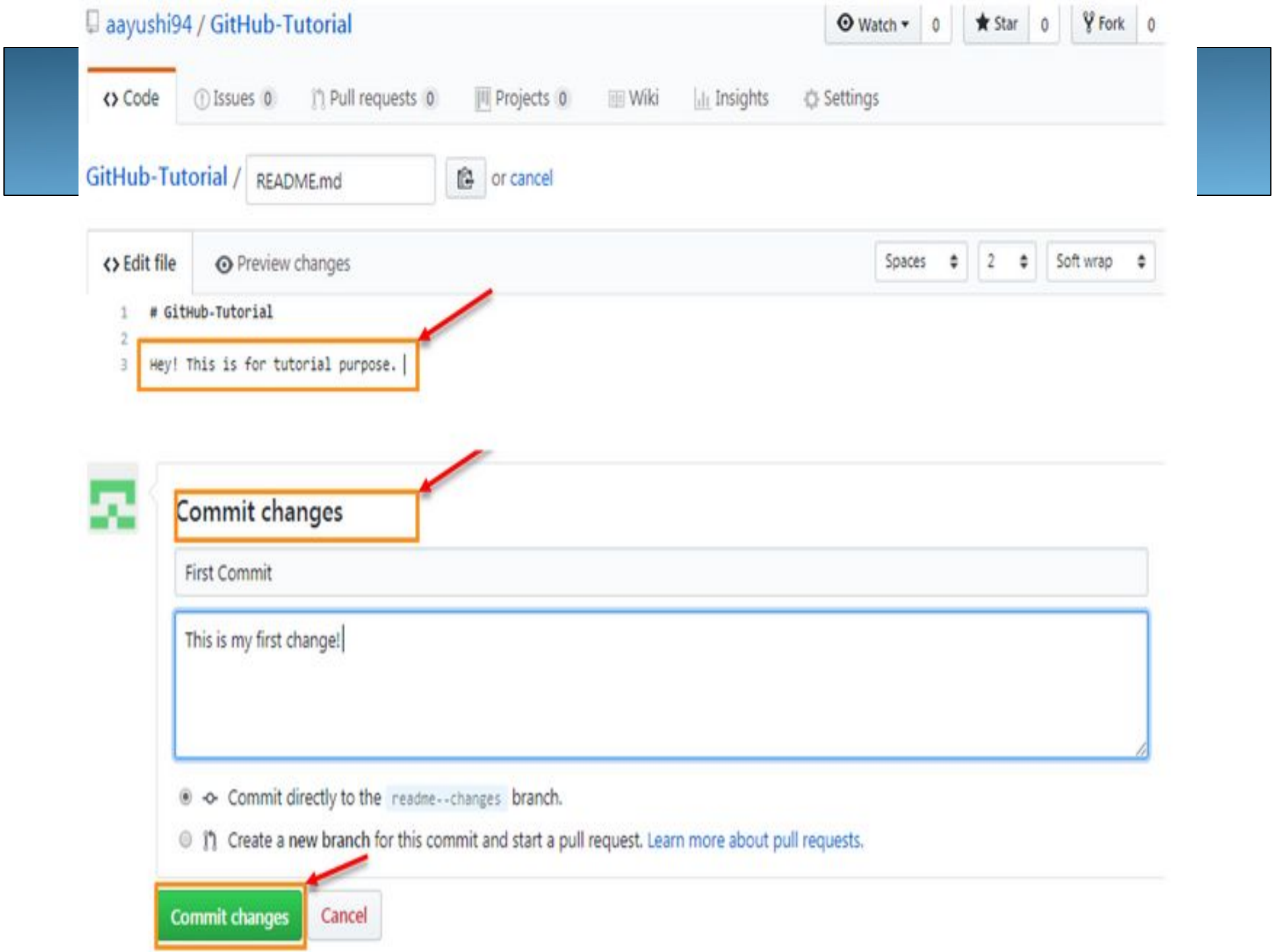
- Enter any repository name and click on “Create Repository”. You can also give a description to your repository (optional).




Search GitHub
Pull requests
Issues
Marketplace
Explore
Create a new repository
A repository contains all the files for your project, including the revision history.
Owner
Repository name
aayushi94
/
GitHub-Tutorial
Great repository names are short and memorable. Need inspiration? How about silver-octo-waddle.
Description (optional)
Public
Anyone can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
Add .gitignore: None
Add a license: None
Create repository


aayushi94 / GitHub-Tutorial
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
No description, website, or topics provided.
Edit
Add topics
1 commit
1 branch
0 releases
1 contributor
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
aayushi94 Initial commit
Latest commit 9848e23 38 minutes ago
README.md
Initial commit
38 minutes ago
README.md
GitHub-Tutorial


This repository
Search
Pull requests
Issues
Marketplace
Explore
aayushi94 / GitHub-Tutorial
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
No description, website, or topics provided.
Edit
Add topics
1 commit
1 branch
0 releases
1 contributor
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
Switch branches/tags
readme- changes
Latest commit 9848e23 3 days ago
Initial commit
3 days ago
Branches
Tags
Create branch: readme- changes
from 'master'
GitHub-Tutorial


aayushi94 / GitHub-Tutorial
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
GitHub-Tutorial / README.md
or cancel
Edit file
Preview changes
Spaces
2
Soft wrap
1 # GitHub-Tutorial
2
3 Hey! This is for tutorial purpose.
Commit changes
First Commit
This is my first change!
Commit directly to the readme--changes branch.
Create a new branch for this commit and start a pull request. Learn more about pull requests.
Commit changes
Cancel

you want to use some code which is present in a public repository, you can directly copy the contents by cloning or downloading.