



**git**



# What to Expect

- Knowledge about Version Control System
- Be familiar using CLI
- Have good practice of using GIT
- Collaborate with others using GIT
- Working with remote repository
- Explore of Github

# Agenda

## Day 1

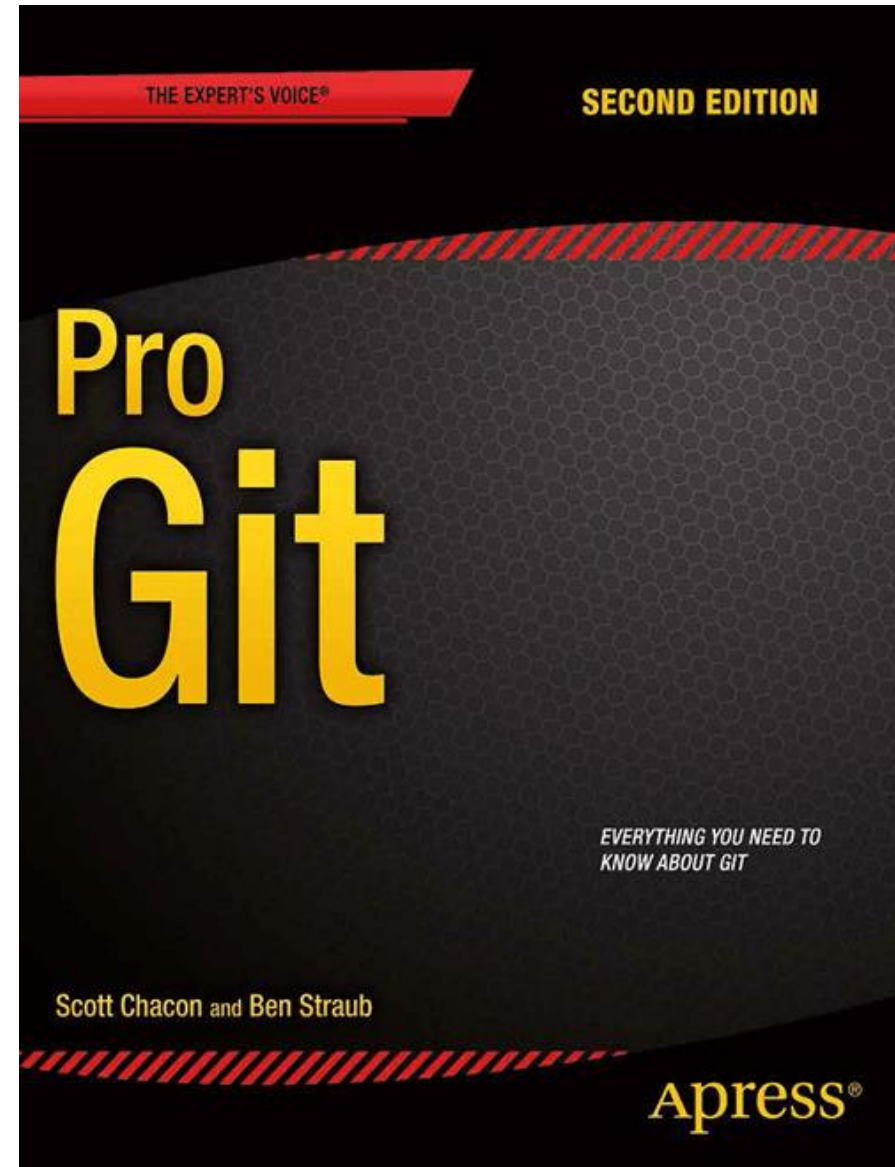
- 09:15 – 09:30: Introduction
- 09:30 – 10:30 : Version Control System (Types – git – how it works)
- 10:30 – 10:50: Break
- 10:50 – 12:30: Dealing with Command line environment

# Agenda

## Day 2

- 09:15 – 10:30: Git core functionalities
- 10:30 – 10:50: Break
- 10:50 – 11:30: Working with remote (github)
- 11:30 – 12:15: Teamwork
- 12:15 – 12:30: Closing remarks

Reading  
recommendation  
& resources



Available online: [link](#)

# Git: Version Control System (VCS)

- Record changes to directory, files, code
- Essential tool when collaborating with others
- Good-practice to use even without working with others
- Git is the most used VCS
- Created by Linus Torvaldes
- Git is a free and open-source software
- It is simply a file inside your directory

# Other VCS

- Apache Subversion (svn)
- Mercurial
- Perforce Helix Core
- Monotone
- Others

# Discussion

- What are the pros and cons of using a version control system?

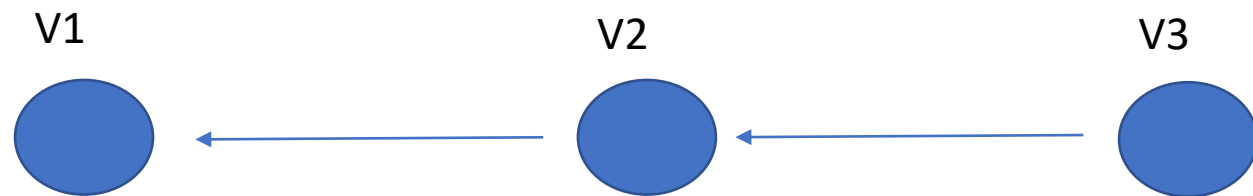


# VCS Approaches

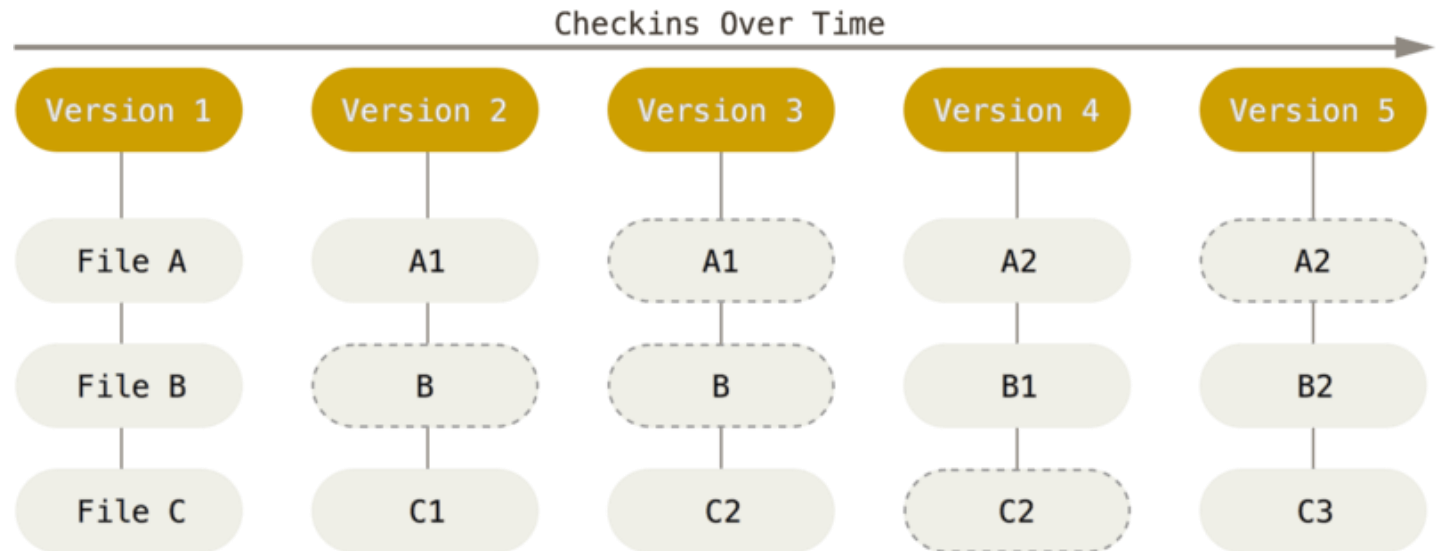
- Basic: make copies of folder with time stamps
- Manually edit changes from others into your code
- Ends up with final copy or version to use

# VCS Approaches

- Take a linear snapshots of the tree history



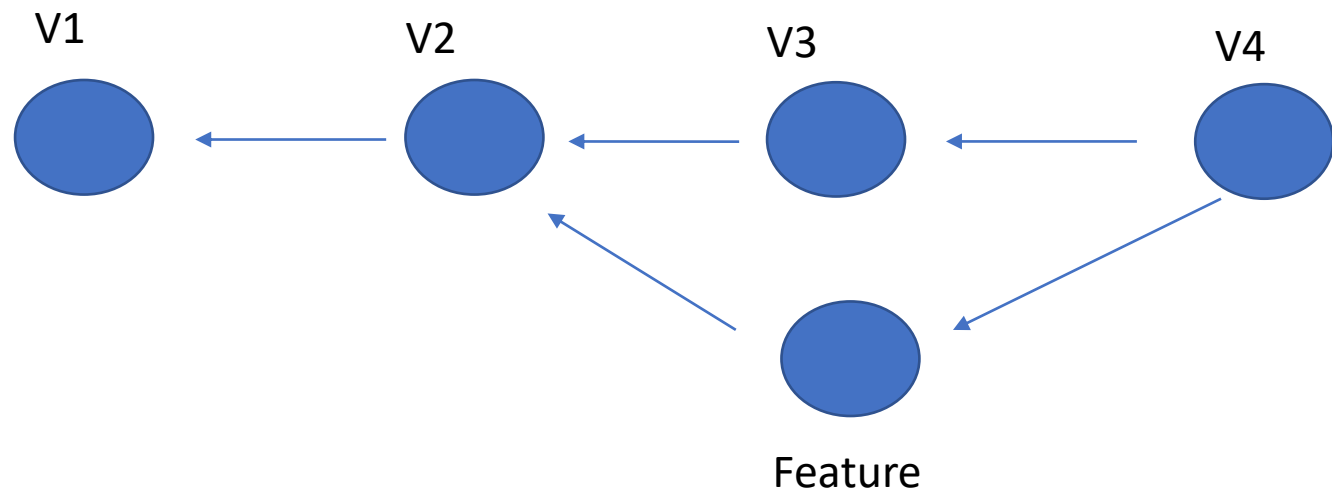
# Git: Snapshots of changes



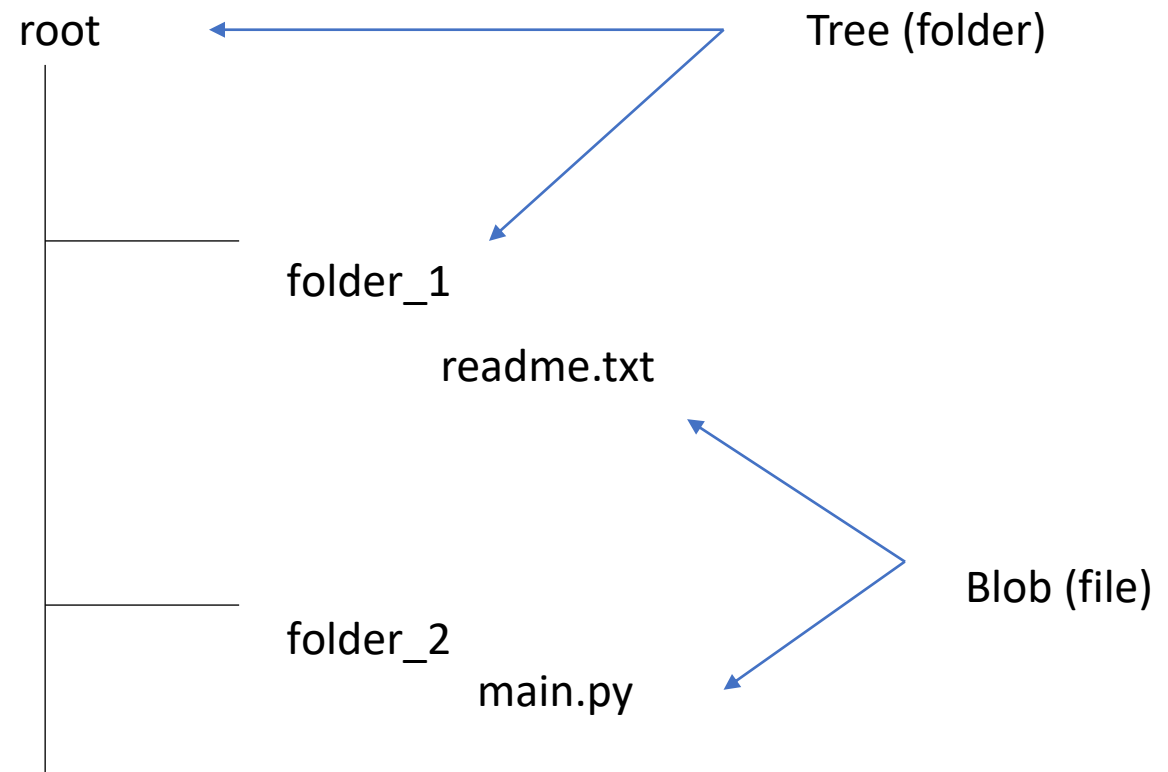
Credit: Pro Git book, Git Documentation

# VCS Approaches

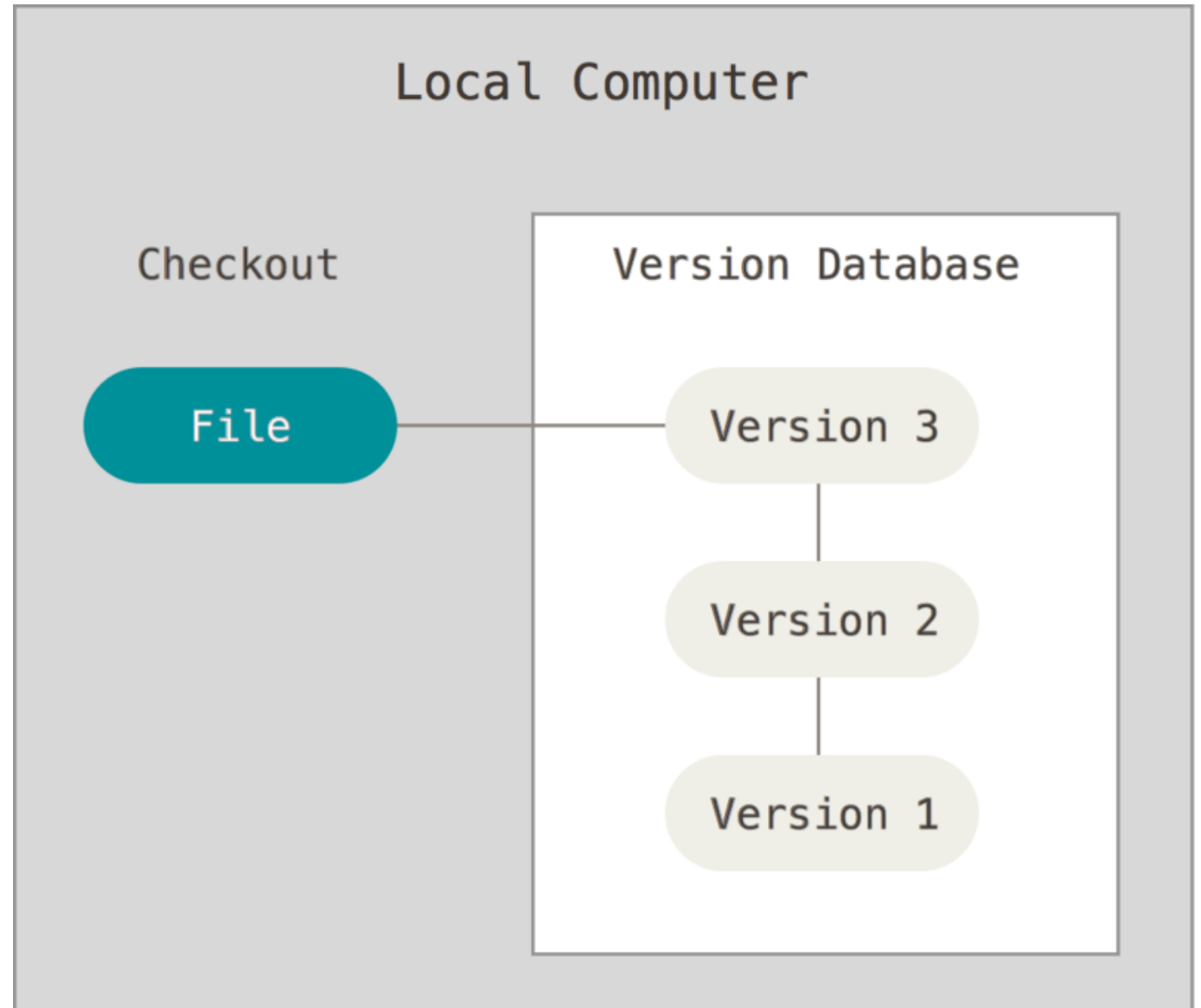
- Git: Directed acyclic graph history



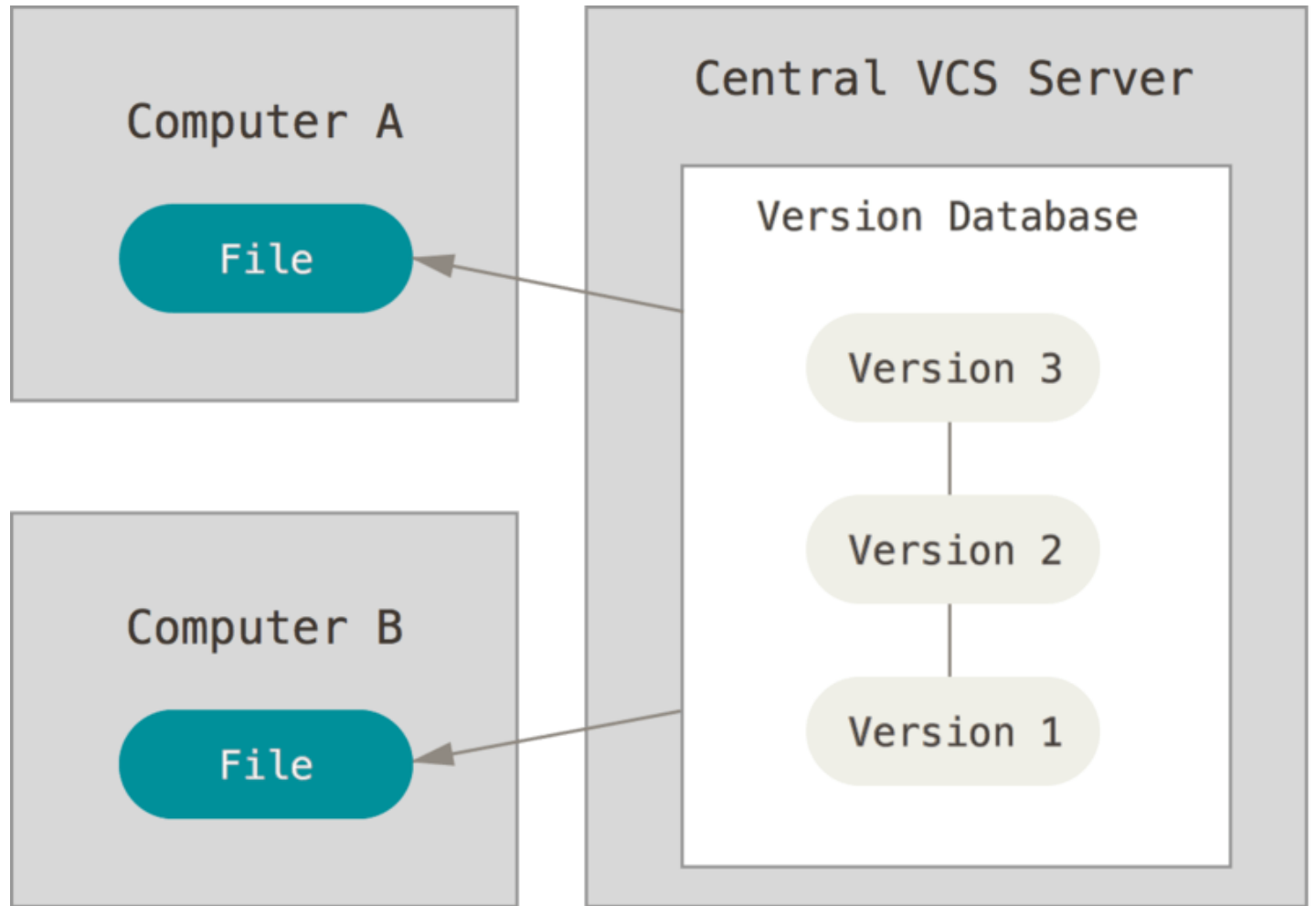
# Directory structure



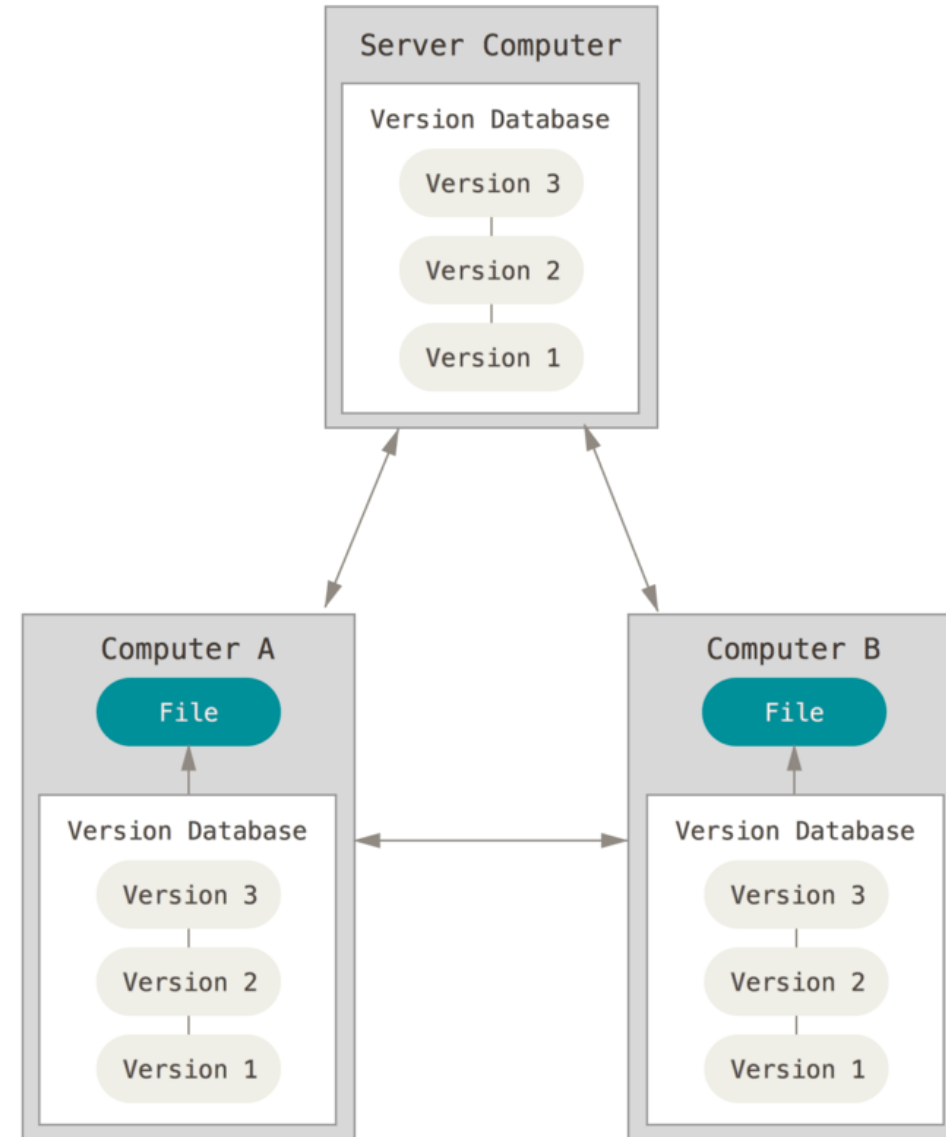
Local VCS



# Centralized VCS



# Distributed VCS





# The Command Line with Git

Why the command line?

- 1- Can run **all** git commands and functionalities
- 2- If you can use git in command line, you will probably know how to use it in GUIs and not vice-versa

# Useful commands

- pwd: Print working directory
- cd: Change directory (cd /mnt/c/users/example/desktop)
- Cd .. : One level up
- ls: «List» It lists the content of directory
- ls -h: list the content of directory in human readable format
- ls -a: list all files and directory (includes hidden)
- cat example\_file: see the content of a file
- mkdir dir\_name: create a new directory in your working path
- man command: gives the documentation of a command
- command --help: also displays information about certain command
- Check the cheat sheet for similar commands on Windows terminal

# Useful commands: Tasks

- List the files of a directory & sort them by size
- Find the usage of `-r` flag in `rm` and use it
- Use `head` or `tail` commands to display certain number of lines (5 or 7 or any number)

# Git: Getting started

- Check the following link to download on your OS: <https://git-scm.com/downloads>
- Debian / Ubuntu: `sudo apt-get install git`

```
ahmad9090@LAPTOP-RTERJCFR:~$ sudo apt-get install git
[sudo] password for ahmad9090:
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.37.1-0ppa1~ubuntu20.04.1).
The following packages were automatically installed and are no longer required:
  libfwupdplugin1 libllvm11
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ahmad9090@LAPTOP-RTERJCFR:~$
```

- Check if installation went well: `# git --version`

```
ahmad9090@LAPTOP-RTERJCFR:~$ git --version
git version 2.37.1
ahmad9090@LAPTOP-RTERJCFR:~$
```

# Git: Getting started

Set up account username, email & password

1- Username

```
# git config --global user.name "Example user"
```

2- Email

```
# git config --global user.email  
"example.user@email.com"
```

3- Password

```
# git config --global user.password "some password"
```

# Git: Getting Help

- # git help command  
Example: git help config
- Shorter help page: # git command -h  
Example: # git config -h

Find the command to display all your global configurations.

# Git: Create a repository

There are two ways to obtain a git repository:

- 1- Turn a local directory into a repository
- 2- clone a git repository from somewhere else

# Git Create a repository

Create a local repository:

- Change your path into the directory where you want to create a repository

```
# cd /home/user/my_project
```

- Type the following command:

```
# git init
```

You can see a new subdirectory created “.git” inside your directory. For more information about the content of “.git”: <https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain#ch10-git-internals>



# The Three States of Git

Modified (Untracked): Changes to file or directory but not added to git database.

```
ahmad9090@LAPTOP-RTJRJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LICENSE
        modified:   README.md
        modified:   name_entities_example_restaurants_zurich.txt
        modified:   webscraping_with_selenium.py

no changes added to commit (use "git add" and/or "git commit -a")
ahmad9090@LAPTOP-RTJRJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$
```

# The Three States of Git

Staged: The modified file has been marked for changes to be committed.

# git add file\_name            or

# git add .                    to mark all the files that are modified

```
ahmad9090@LAPTOP-RTJRJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git add README.md
ahmad9090@LAPTOP-RTJRJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   LICENSE
    modified:   name_entities_example_restaurants_zurich.txt
    modified:   webscraping_with_selenium.py

ahmad9090@LAPTOP-RTJRJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$
```

# The Three States of Git

```
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   name_entities_example_restaurants_zurich.txt

no changes added to commit (use "git add" and/or "git commit -a")
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status -s
M name_entities_example_restaurants_zurich.txt
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git add name_entities_example_restaurants_zurich.txt
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ git status -s
M name_entities_example_restaurants_zurich.txt
ahmad9090@LAPTOP-RTJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$
```

# git status -s

# The Three States of Git

Committed: The changes are safely stored to your local database

```
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/web scraping_with_selenium$ git commit -m "version 2, changes to all files"
[main 0336347] version 2, changes to all files
 4 files changed, 221 insertions(+), 221 deletions(-)
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/web scraping_with_selenium$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/web scraping_with_selenium$
```

Skip the staging area by adding -a argument: `# git commit -a -m "fix bug in example.py"`

# Commit: Best Practices

- Commit Related Changes
- Commit frequently
- Keep it short (about 50 characters)
- Don't commit half-done work
- Test your code before commit
- Use imperative voice “fix bug”, instead of “fixed bug”
- Leave second line blank if writing long commit



# Commit: Best Practices

- What is wrong with this staging / committing?

```
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    new file:   .DS_Store  
    new file:   products.ts  
    new file:   registration.test.ts  
    new file:   registration.ts  
    new file:   validation.test.ts  
    new file:   validation.ts
```

```
git commit -m 'Updated various areas such as validation,  
registration and products pages'
```

Source: freeCodeCamp.org

# Task

- Create a new directory
- Create a git repository inside the directory
- Add files into the directory
- Commit the changes of the directory into git

# Ignore files

- Sometimes there are files that you want git to ignore from tracking.
- Create a file inside your repository called .gitignore
  - on linux: # touch .gitignore
  - on windows: # type nul > .gitignore
- Write the name of the file inside .gitignore

```
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ ls -l
total 11517
-rwxrwxrwx 1 ahmad9090 ahmad9090 1093 Jul 27 04:22 LICENSE
-rwxrwxrwx 1 ahmad9090 ahmad9090 608 Aug 5 04:10 README.md
-rwxrwxrwx 1 ahmad9090 ahmad9090 11775488 Jul 27 02:34 chromedriver.exe
-rwxrwxrwx 1 ahmad9090 ahmad9090 253 Sep 26 15:52 name_entities_example_restaurants_zurich.txt
-rwxrwxrwx 1 ahmad9090 ahmad9090 851 Aug 5 04:05 results_18135.json
-rwxrwxrwx 1 ahmad9090 ahmad9090 34 Sep 26 16:14 to_ignore.txt
-rwxrwxrwx 1 ahmad9090 ahmad9090 6889 Aug 5 04:04 webscraping_with_selenium.py
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$ cat .gitignore
to_ignore.txt
ahmad9090@LAPTOP-RTERJCFR:/mnt/c/users/ahmad/desktop/coding_stuff/webscraping_with_selenium$
```

More information on .gitignore syntax: <https://git-scm.com/docs/gitignore>



# Task

- Add .gitignore into your repository
- Add three .txt or .py files into the repository
- Have git ignore these 3 files by writing 1 line into .gitignore
- Add a fourth txt file into your repository but make sure it doesn't get ignored by git

# Git: Show the history of Commits

To list the commits made in a repository use the command  
log

```
# git log
```

log shows all the commits in a reverse chronological order

log has options. To show the differences introduced in each commit, use the argument --patch or -p. You can also limit the number of commits display by adding -number

```
# git log -p
```

```
# git log -p -2 (shows the last two commits)
```

# Undoing changes in Git



# Undoing changes

Reverse changes to modified file until last commit:

```
# git checkout -- file_name
```

or

```
# git restore file_name
```

USE WITH CAUTION!

# Undoing changes

Unstaging a staged file:

```
# git reset HEAD file_name
```

Useful if you want to have different commits for different files

Newer versions of git introduced `git restore --staged`:

```
# git restore --staged file_name
```

# Undoing changes

You can edit a commit by using the argument `--amend`

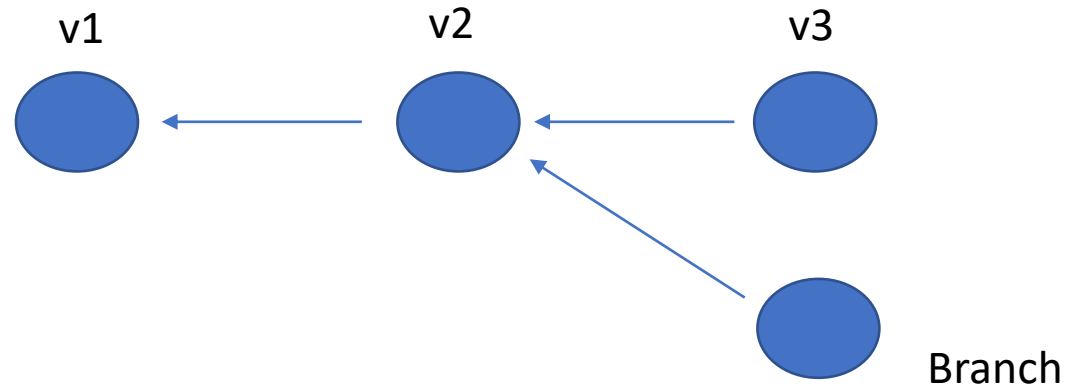
```
# git commit --amend
```

This will allow you to edit a commit. It gives the possibility to stage other files and add them to the commit.

# Task

- Make changes to a file in your git repository
- Undo these changes until the last commit
- Make several changes and corresponding commits to file then undo these changes to a certain commit (not last one)

# Branching



- A pointer to one of your commits
- Most VCS have branching support
- Git is unique in handling branching (Less expensive, lightweight)



# Create New Branch

- Check the branch name of your repository:  
# git branch  
by default, called master or main
- Create new branch:  
# git branch name\_of\_branch
- See the last commit in each branch  
# git branch -v
- Move to work in a certain branch:  
# git checkout name\_of\_branch  
Git does not switch to new branch automatically after creating it!

# Check Branch changes

- Git log will only show the commits history for the main or master branch when you are in it. To show commits in other branches:

```
# git log branch_name
```

or

```
# git log --all
```

- Use the argument --oneline to display less information when having too many commits:

```
# git log --all --oneline
```

# Git: Switch

- Newer versions of git (2.23) uses switch command to instead of checkout:

```
# git switch branch_name
```

- Delete a git branch:

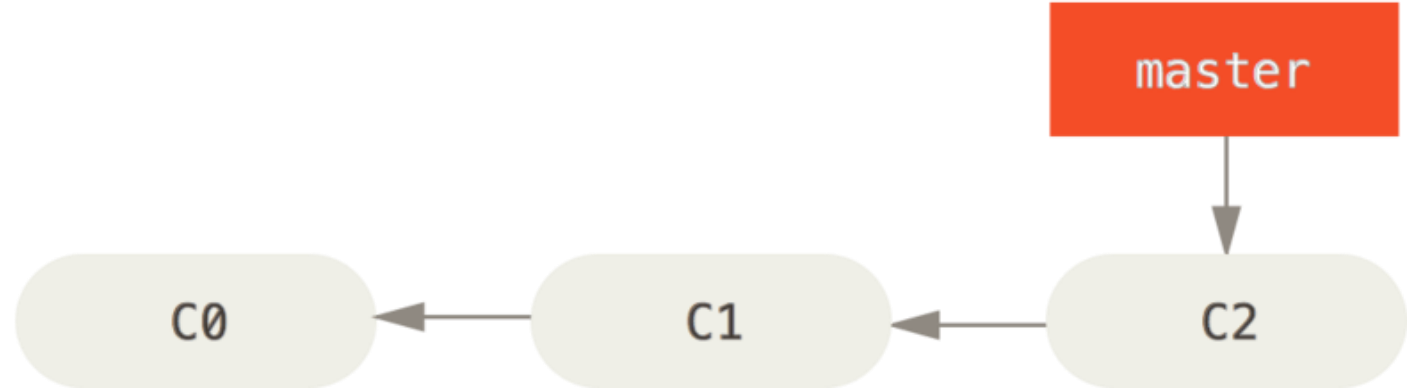
```
# git branch -d branch_name
```

```
-D if there are unmerged elements
```

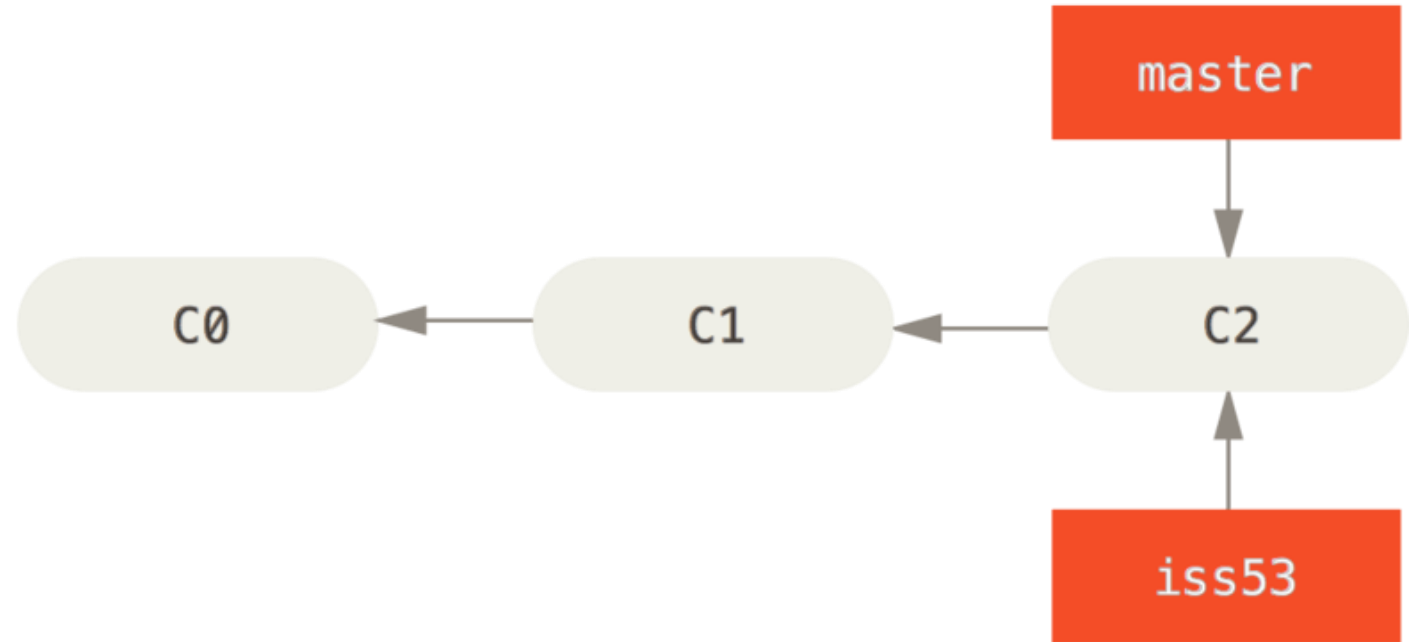
# Discussion & Task

- Why branching is useful?
- Create a new branch and switch to it in one command (there are 2 possible ways)
- Do changes in that branch then try to delete it

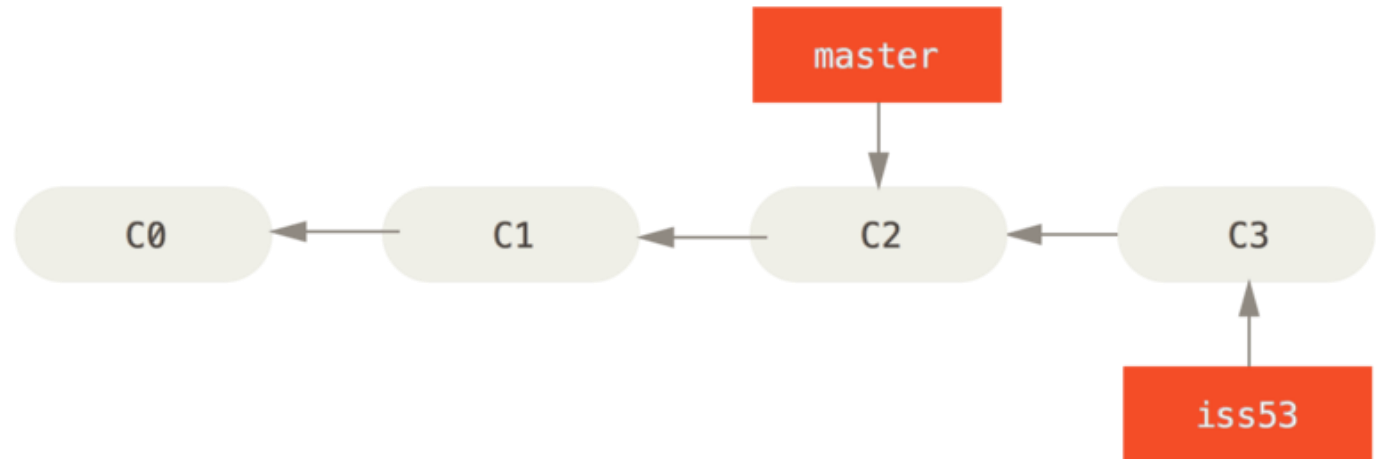
# Branching & Merging



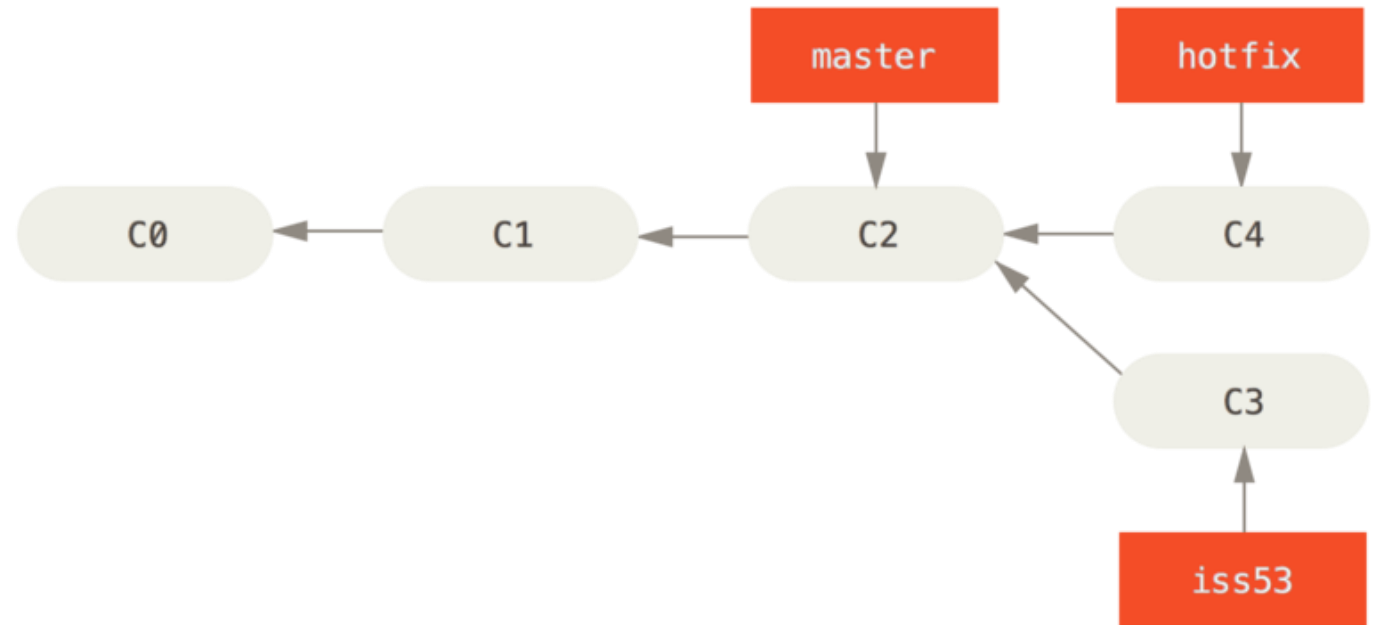
# Branching & Merging



# Branching & Merging

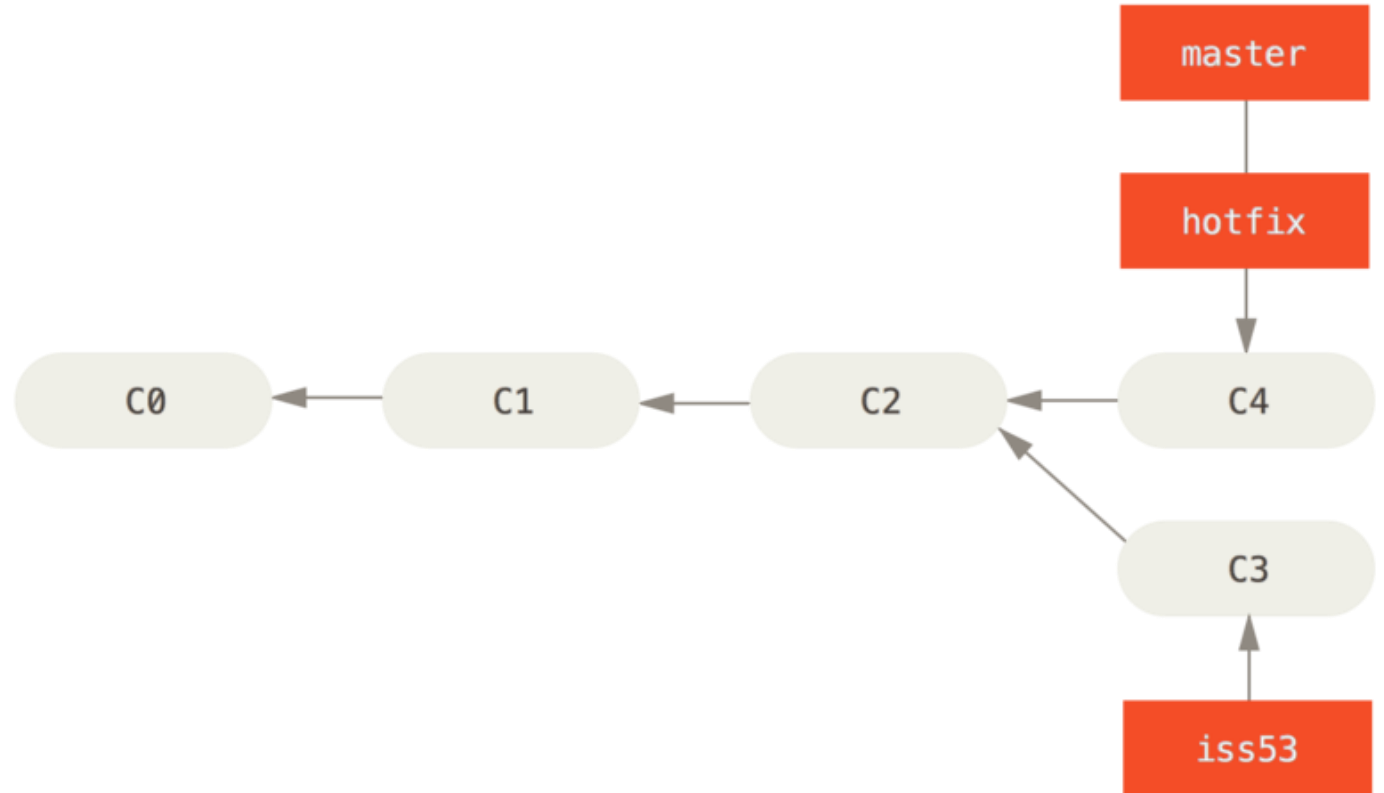


# Branching & Merging

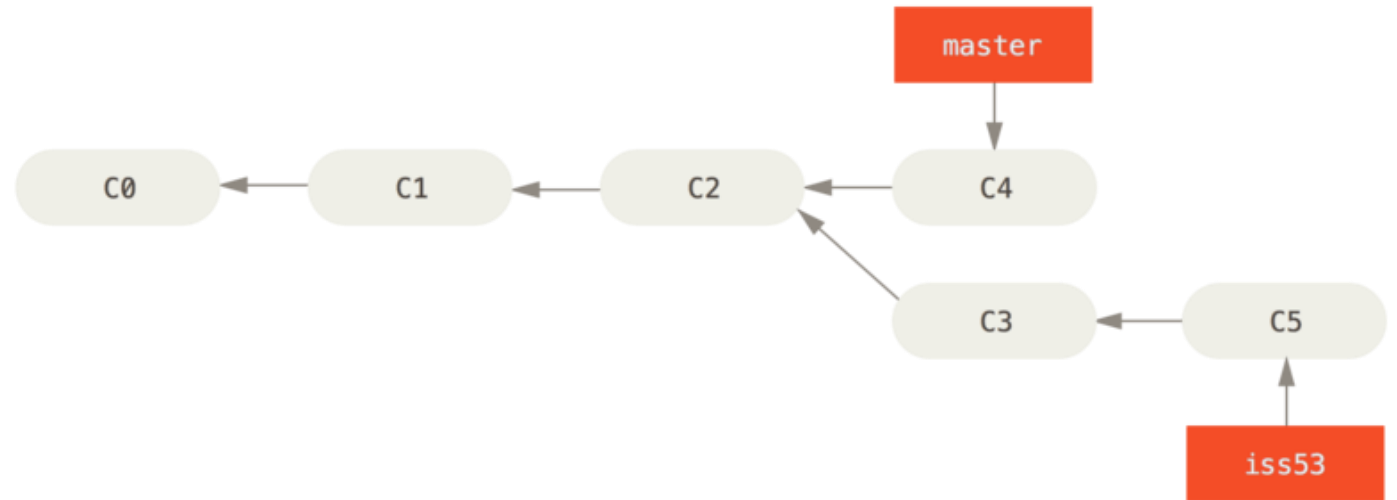




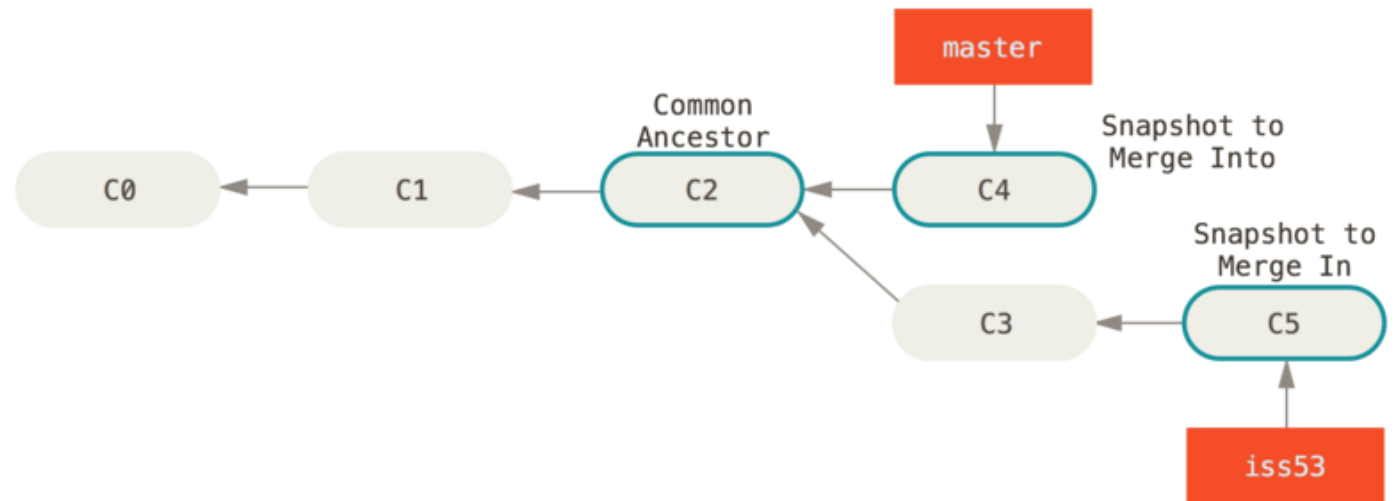
# Branching & Merging



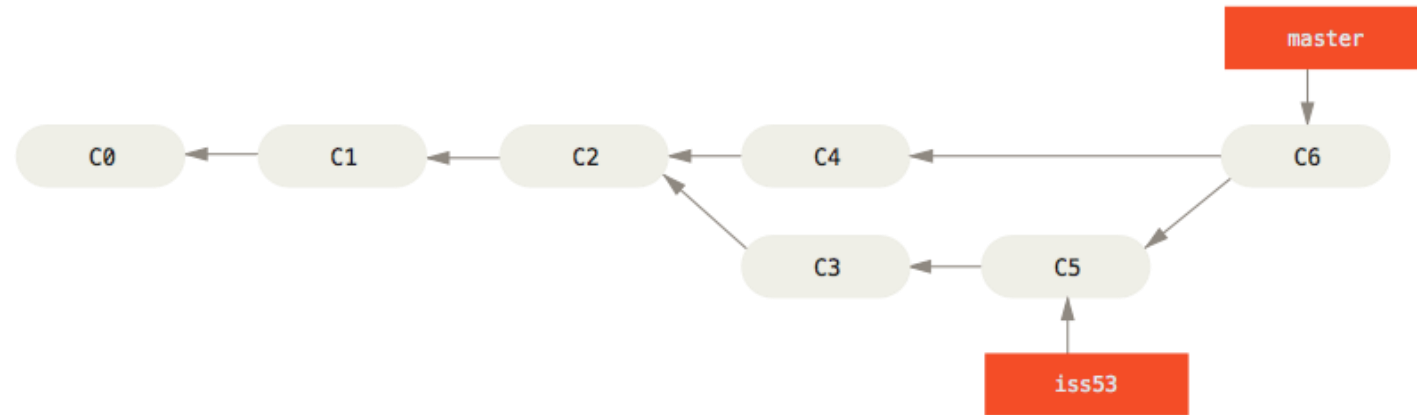
# Branching & Merging



# Branching & Merging



# Branching & Merging



# Git: Branching & merging

- Merging branch: Go to the branch you want to merge into:  
`# git merge branch_name`
- Dealing with merge conflict:  
`# git mergetool` OR manually
- See which branch is merged and not merged into your current branch  
`# git branch --merged`  
`# git branch --no-merged`
- Abort a merge: Sometimes when you have a merge conflict, you want to abort the merge and fix it in a different way. This is done with the abort flag:  
`# git merge --abort`
- An option would be to go back to the last commit:  
`# git reset --hard HEAD`  
Use with caution. All changes until last commit are deleted!

# Task

- Create new branch and do changes in it and commit them.
- Do changes in your master branch to the same file and the same lines
- Try to merge the branch and resolve the conflict

# Git: Branching & merging

- Dealing with merge conflict strategies:
  - -X ours: to accept changes in current branch
  - -X theirs: to accept changes in the merged branch

# Git: Branching & Merging

- Useful flags to use with git merge:
- -Xignore-space-change
- -Xignore-all-space



# Rebasing

- Alternative to git merge
- Provides linear “cleaner” history
- Use with caution
- Can be done with the rebase command:  
    `# git rebase branch_name`

# Git Stash

- If you don't want to commit certain work but still store it for later use, you can use git stashing.

`# git stash`

- You can also add a description of the stash you created:

`# git stash save "changes to file x"`

- To display the stashes you have:

`# git stash list`

- Apply the first stash changes to your branch:

`# git stash pop`

- Reapply certain stash back:

`# git stash apply <stash_id stash@{0}>`

# Git Stash

- Using stash-apply without specifying the stash id will apply the most recent stash you saved
- You can remove a stash by using drop:  
`# git stash drop stash@{0}`
- You can create a branch from the stash that you created:  
`# git stash branch branch_name`
- Clear all the changes but save them into stash:  
`# git stash --all`

# Git Aliases

- For a faster & more sufficient workflow use local aliases
- Examples from Pro git book:

```
$ git config --global alias.co checkout  
$ git config --global alias.br branch  
$ git config --global alias.ci commit  
$ git config --global alias.st status
```

# Git: Searching

- You can use `git grep` to search the content of the files inside your directory. Use the `-n` or `--line-number` option to view the line number.
- Use the `-c` or `--count` to view only the file containing the searched string
- If you are searching for when rather than where a certain change occurred, use the `log -S` functionality

# Cleaning work directory

- You can use git-clean to remove any unnecessary files or subdirectories that are unneeded:

```
# git clean -f -d
```

- Use git-clean with cautious. It is better to run first `--dry-run` option to see what would be removed:

```
# git clean -d -n
```

- The previous command will not remove the files that you have in `.gitignore`. You can include those with the `-x` flag
- The `-i` flag allows interactive mode with the clean command as a more safe mesurment.

# GIT: GUI

- Git comes with graphical user interface (git-gui). Can run it from the command line:

```
# git gui
```

- Most IDE has extension for using git (VS code, Pycharm, etc)

# Github

- What is Github and how is it different from git?



# Github

- Largest online host for git repositories
- Central point for collaboration for millions of developer
- Many open-source projects on Github
- Owned by Microsoft (Not part of the Git open-source project)

# Github: Usage

- Interaction with local git repository
- Github Pages: Static web hosting service
- Gist: code snippets

# Github: Create Account and SSH Access

- GUI to create account at github!
- Generate SSH public key and add it to your github account from settings
- Key is usually stored at ~/.ssh
- To generate a new ssh key use:
  - `# ssh-keygen -o`
- Paste your public key into github

# Github fork

- You can fork a project in github into your private account to work on it. This is usually is done when you do not have push access to that project.

# Github: Fork a repository

The screenshot shows the Django repository on GitHub. The repository name is 'django/django' and it is public. The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The repository statistics show 27.9k forks and 66.6k stars. The 'Fork' button is circled in the top right corner. The repository has 25 branches and 358 tags. The file list on the left includes .github, .tx, django, docs, extras, js\_tests, scripts, tests, .editorconfig, .eslintignore, .eslintrc, .git-blame-ignore-revs, .gitattributes, .gitignore, .pre-commit-config.yaml, .readthedocs.yml, AUTHORS, CONTRIBUTING.rst, Gruntfile.js, INSTALL, and LICENSE. The commit history on the right shows the latest commit by felixxm, titled 'Relaxed system check for template tag modules with the...', 4 hours ago, with 31,102 commits.

Search or jump to... Pull requests Issues Marketplace Explore

django / django Public

Sponsor Watch 2.3k Fork 27.9k Star 66.6k

<> Code Pull requests 158 Actions Security Insights

main 25 branches 358 tags Go to file Add file Code

felixxm Refs #32987 -- Relaxed system check for template tag modules with the... 4 hours ago 31,102 commits

.github	Fixed permissions for GitHub action with welcome message.	14 days ago
.tx	Removed contrib-messages entry in Transifex config file	6 years ago
django	Refs #32987 -- Relaxed system check for template tag modules with the...	4 hours ago
docs	Refs #32987 -- Relaxed system check for template tag modules with the...	4 hours ago
extras	Refs #23433 -- Removed django-admin.py entry point per deprecation t...	2 years ago
js_tests	Fixed #24179 -- Added filtering to selected side of vertical/horizont...	27 days ago
scripts	Refs #33476 -- Reformatted code with Black.	8 months ago
tests	Refs #32987 -- Relaxed system check for template tag modules with the...	4 hours ago
.editorconfig	Refs #33476 -- Adjusted docs and config files for Black.	8 months ago
.eslintignore	Refs #32165 -- Bumped minimum ESLint version to 7.16.0 to match pre...	2 years ago
.eslintrc	Enabled ESLint's "eqeqeq" rule.	2 years ago
.git-blame-ignore-revs	Ignored c188618 formatting changes in git blame.	2 months ago
.gitattributes	Fixed #19670 -- Applied CachedFilesMixin patterns to specific extensions	7 years ago
.gitignore	Fixed #22446 -- Added tox.ini to automate pull request checks.	6 years ago
.pre-commit-config.yaml	Bumped versions in pre-commit and npm configurations.	last month
.readthedocs.yml	Configured Read The Docs to build all formats.	11 months ago
AUTHORS	Fixed #24179 -- Added filtering to selected side of vertical/horizont...	27 days ago
CONTRIBUTING.rst	Added link to the code of conduct from contributing guides.	8 years ago
Gruntfile.js	Refs #31265 -- Updated .eslintignore to match eslint tests	2 years ago
INSTALL	Fixed #32355 -- Dropped support for Python 3.6 and 3.7	2 years ago
LICENSE	Whitespace cleanup.	9 years ago

About

The Web framework for perfectionists with deadlines.

[www.djangoproject.com/](https://www.djangoproject.com/)

python django views framework om web apps templates models

Readme

BSD-3-Clause. Unknown licenses found

Code of conduct

66.6k stars

2.3k watching

27.9k forks

Releases

358 tags

Sponsor this project

dj django Django

<https://www.djangoproject.com/fundrai...>

Learn more about GitHub Sponsors

Packages

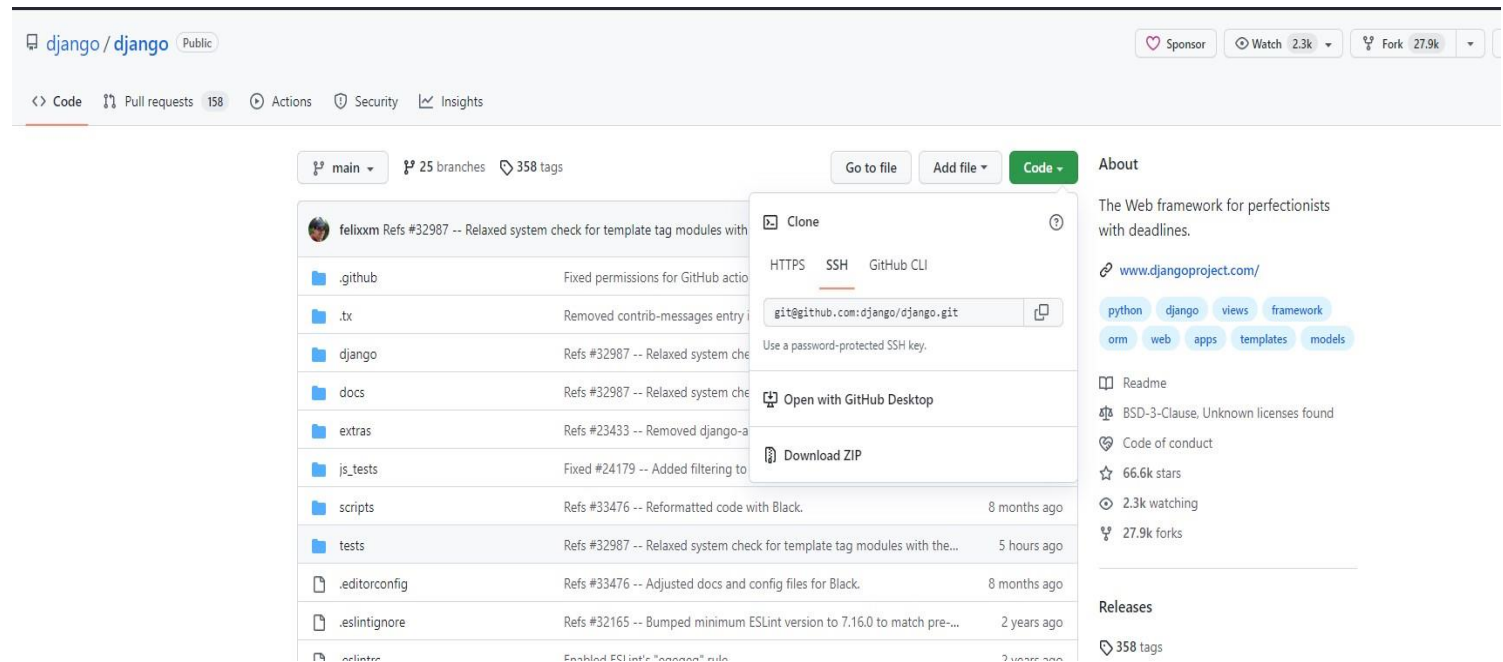
No packages published

Used by 1m

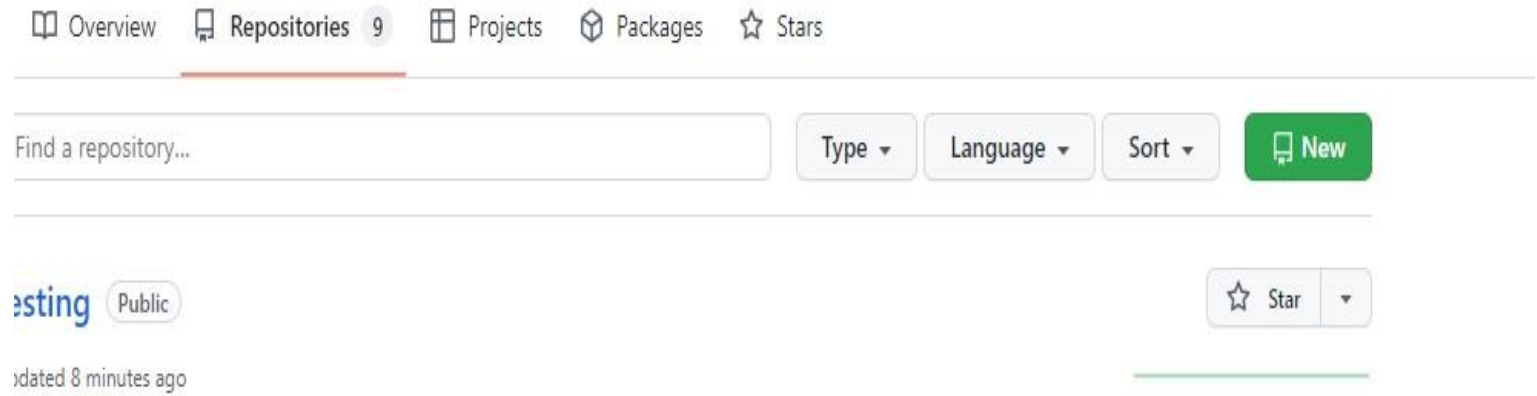
# Git & Github: working with remote

- You can clone a repository from github or any remote repository from a different location by using git clone:  
`# git clone https://github.com/django/django.git`
- You can use either HTTPS or SSH to clone a repository from github

# Git: clone



# Github: Create new repository



From your account -> repositories -> new



# Git: working with remote repository

- Clone your new repository into your local device either with https or ssh:

```
# git clone git@github.com:<user name>/<repo name>.git
```

- You can list the name Git gives to the remote repository that you cloned:

```
# git remote
```

This will give you at least «origin»

- You can also view the URLs which git specify to reading and writing to the remote repo:

```
# git remote -v
```

- Add changes to your remote repository that you did locally:

```
# git push
```

- Pull changes that were made to the remote repository:

```
# git pull
```

# Github: Adding collaborators

- You can add collaborators to your repository on github to work with others.
- Is done from settings inside the repository --> collaborators --> add people

# Task

- Fork a public repo on github then clone it to your device
- Change the readme file
- Push the changes into github
- Add the person next to you as collaborator
- Change the repo into a private one

# Github on the CL

- You can use Github on the command line as well
- To install follow the instructions on this link for your OS:  
<https://github.com/cli/cli#installation>
- To log in to your GitHub account on the CL:  
`# gh auth login`

# Github on the CL

- Assume you have been working locally on a project and you want to have it as a repository on Github. You can create a new repository on Github from your CL.
- Use the command:  
    `# gh repo create`  
    Very user friendly

# Task

- Install github CLI & log in to your account with it
- Try to create a new repo on github via gh CLI tool

# Github on the CL

- After adding the repo on Github, You can push your local work to it with the following commands:

```
# git branch -M main
```

With SSH:

```
# git remote add origin git@github.com:<user  
name>/<repo name>.git
```

With HTTPS:

```
# git remote add origin https://github.com/<user  
name>/<repo name>.git
```

# Teamwork

- Group in teams of 2 or 3 and choose one of the following tasks: