Scientific Computing @ CLASSE

Version control

2025-06-13

Overview

- What is version control
- What is git
- Where to host your project’s code repository
- Practice using git

Version control

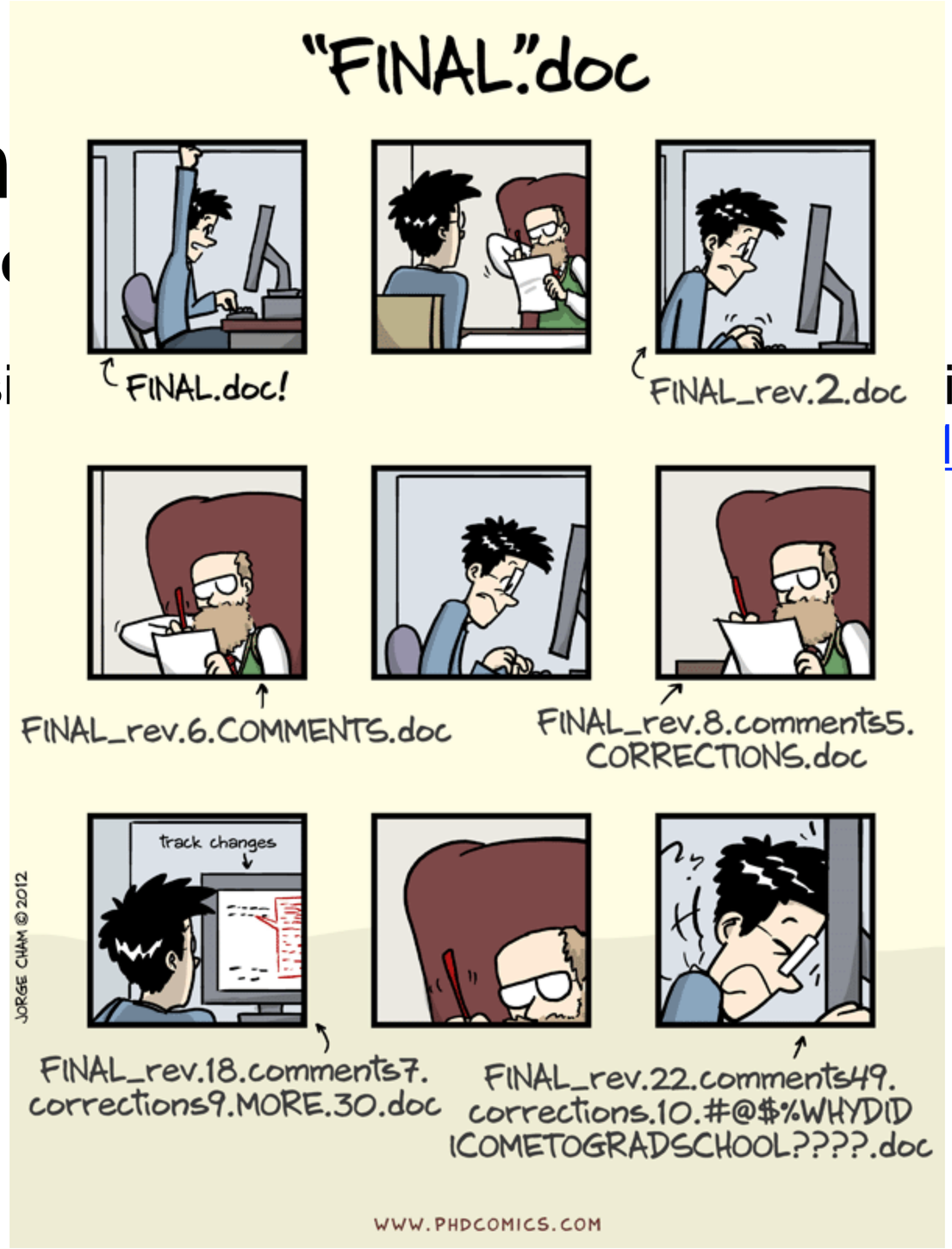
What is version control?

- Description of version control, and the motivation for having sophisticated tools to help us do it: https://xcitecourse.org/theme1/PE103/vcs.html

Version con

What is version c

- Description of versi ing sophisticated tools to help us do 103/vcs.html



https://phdcomics.com/comics/archive.php?comicid=1531

Version control

What is git?

- git: the near-ubiquitous choice for version control software
- `git`: command-line tool for interacting with git repositories
- git repository: a copy of your project files and the history of changes you made to them. It manages snapshots of your project at different stages of development
- A git repository can be hosted remotely using tools like github or gitlab. These help you:
  - Share / distribute / deploy your project
  - Track issues
  - Publish documentation

Version control (git)

Where to host your repository

- CHESS students:
  - Your project's git repository will be hosted in one of two places:
    - For projects that should be public:

      https://github.com/CHESSComputing

    - For projects that should be shared only within CLASSE:

      https://gitlab01.classe.cornell.edu

  - There is a "correct" place to host every one of your projects. Ask your mentor if you are unsure where your project belongs.
- All other students: ask your project mentor

Version control (git)

Vocabulary

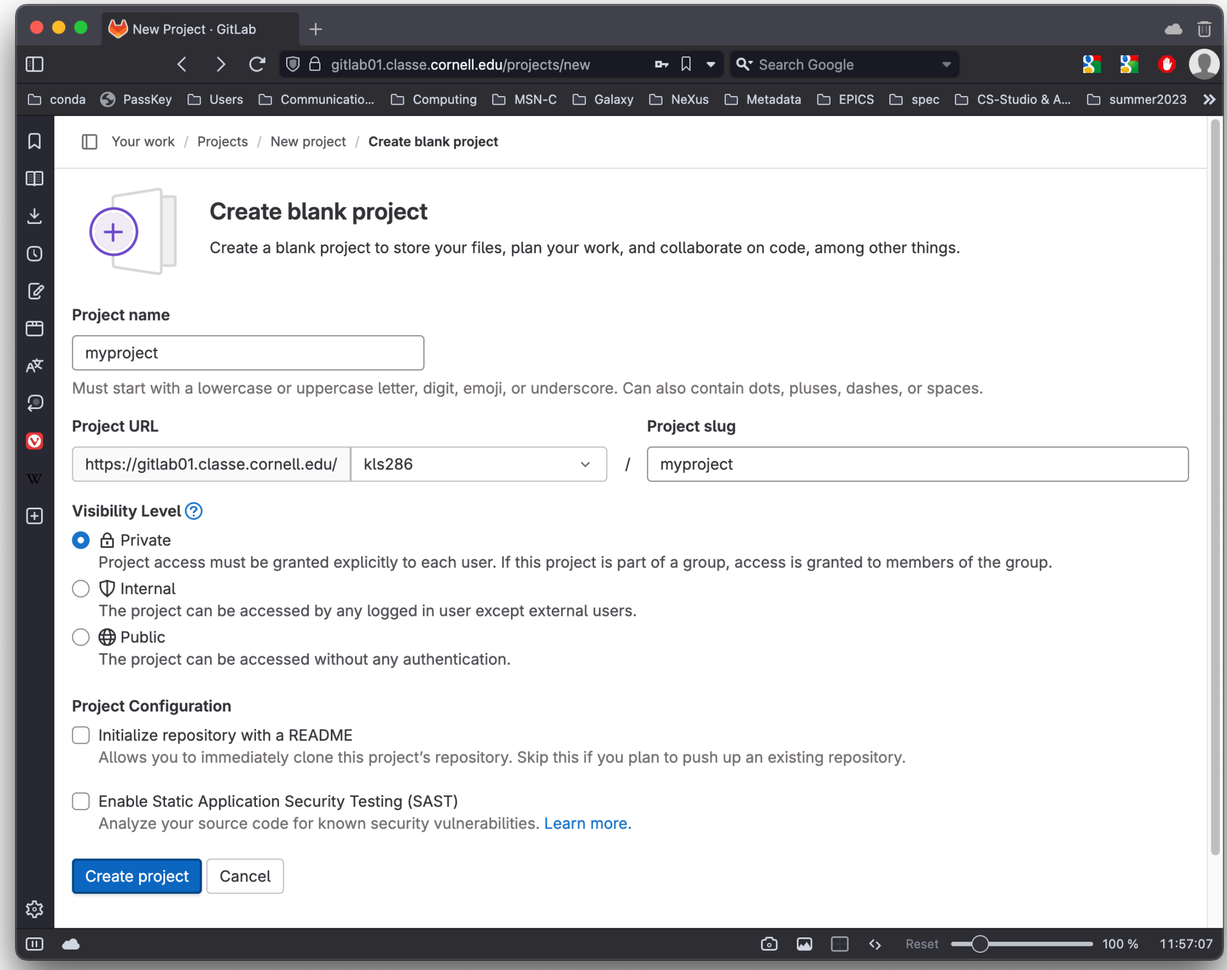
- **repository**: a collection of files with a history of developer-created checkpoints that preserve the state of those files at different stages (e.g. commits)
- **remote repository**: a repository accessible via URL (i.e. the version hosted on github/gitlab)
- **local repository**: a repository on your local filesystem
- **clone**: a local repository that is a copy of a remote repository
- **pull**: an act that updates your local repository with any new changes on the remote repository
- **commit**: an entry / checkpoint in the preserved history of a repository
- **push**: an act that updates a remote repository with changes (like commits) made on a local repository

Version control (git)

Exercise 1

Set up a blank repository:

1. Log on to https://gitlab01.classe.cornell.edu
2. Create a new repository for your project
    1. In upper righthand corner, click “New Project”
    2. Create a blank repository (see screenshot —>)
3. In a terminal on the CLASSE system, run:
    1. `cd <your working directory here>`
    2. `git clone https://gitlab01.classe.cornell.edu/<CLASSE-ID>/<projectname>.git>`
    3. `cd <projectname>`
    4. `git checkout -b main`



Version control (git)

Basic development workflow

- At any point:
  - Use `git status` to examine the state of your local clone
  - Use `git diff` to examine the differences between files in the local clone and their “official” copy
- To make changes:
  1. Create / modify files in a local clone
  2. `git add` the files whose changes you want to keep
  3. `git commit` the additions
  4. (optional) repeat steps 1-3
  5. `git push` the commits

Version control (git)

Exercise 2

Practice committing & pushing your first change to the repo created in Exercise 1. In the terminal, run:

1. `git status`
2. `nano README.md`
3. Add some text to the file, save, and exit
4. `git status`
5. `git add README.md`
6. `git status`
7. `git commit -m "add README"`
8. `git status`
9. `git push --set-upstream origin main`
10. `git status`

Version control (git)

Exercise 3

Practice committing & pushing another change. In the terminal, run:

1. `nano env.yml`
2. Add some text to the file; suggested: →

```
name: myenv
channels:
  - conda-forge
  - nodefaults
dependencies:
  - python
  - numpy
```

3. `git status`
4. `git add env.yml`
5. `git commit -m "add env.yml"`
6. `git push`

Version control (git)

Exercise 4

Practice pulling changes to your local clone of the repo created in Exercise 1.

- In the gitlab web interface, edit README.md
- In the terminal, run:

```
1. git fetch
2. git status
3. git pull
4. git status
```

Version control (git)

commits

- Commits are the save states / checkpoints for each version of your project
- Commit early and often so you can go back to previous versions of your project if needed
- Each commit should contain *one small, incremental* change to your project (e.g. don't fix a bug *and* update the documentation in the same commit)
  - If you are new to git and development is too chaotic to keep up with incremental commits: at least make a commit at the end of every day / period of code activity
- Remember to `push` your local commits to the remote repo, too!
- Every commit needs a message that describes the changes you're making
- To make the commit history easy to read, stick to conventions

Version control (git)

commit message conventions

- Use the imperative mood (e.g. “Fix bug”, NOT “Fixed bug” or “Fixes bug”)
- Format your commit messages:

```
type: short summary (50 chars or fewer)

More detailed description wrapped to 72 chars wide (optional)
```

Subject line will be previewed in github / gitlab UIs

Blank line

Commit message body is optional — be as detailed as is appropriate, and use Markdown syntax if it helps legibility

- Some common commit types and their descriptions:

```
feat: addition of some new features
add: changes to add new capability or functions
cut: removing the capability or functions
fix: a bug fix
bump: increasing the versions or dependency versions
build: changes to build system or external dependencies
make: change to the build process, or tooling, or infra
ci: changes to CI configuration files and scripts
doc: changes to the documentation
test: adding missing tests or correcting existing tests
chore: changes for housekeeping (avoiding this will force more meaningful message)
refactor: a code change that neither fixes a bug nor adds a feature
style: changes to the code that do not affect the meaning
perf: a code change that improves performance
revert: reverting an accidental commit
```

Version control (git)

Some security considerations

- Never keep passwords, API keys, or other “secrets” in regular files in any git repository
- Never keep CLASSE hostnames, addresses, or file paths in a git repository not hosted on gitlab01.classse.cornell.edu
- If you are unsure: ask your project mentor

End