VERSION CONTROL WITH GIT

Ox-Ber Autumn School

Bernd Taschler

21 Nov 2023

VERSION CONTROL WITH GIT

Ox-Ber Autumn School

Bernd Taschler

21 Nov 2023

Contents

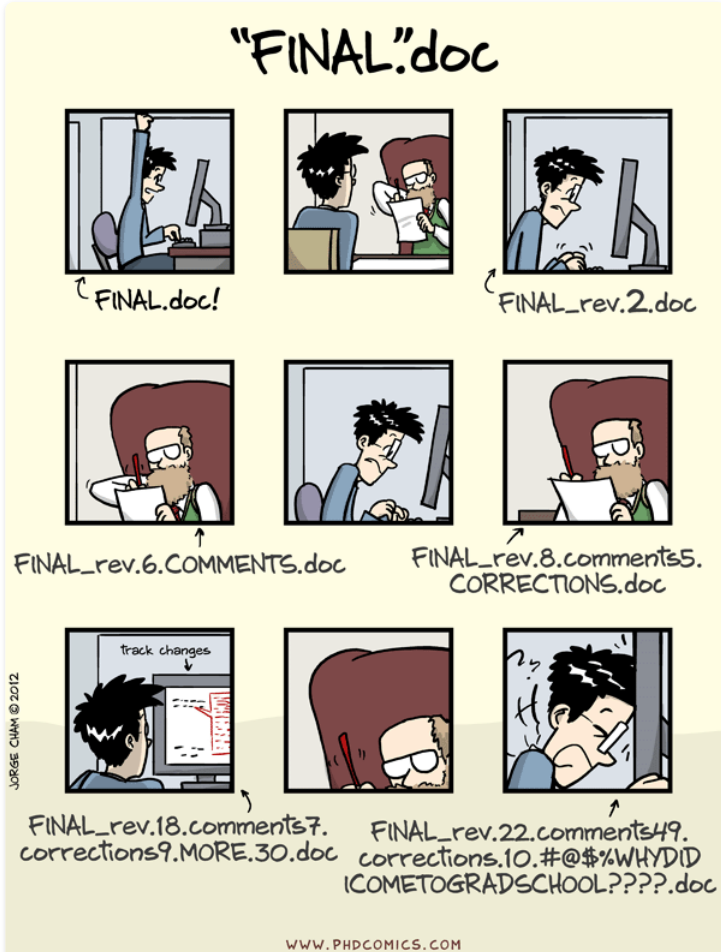
Part 1 - What is Version Control?

Why use version control?

- a better way to “undo” changes,
- a better way to collaborate than mailing files back and forth, and
- a better way to share your code and other scientific work with the world.

Why use version control?

- a better way to “undo” changes,
- a better way to collaborate than mailing files back and forth, and
- a better way to share your code and other scientific work with the world.



my_paper

```
- final_version.pdf
- final_version2.pdf
- final_version_with_comments.pdf
- final_final_version_REVISED.pdf
- ...
- definitely_final_version03_updated_LASTedit.pdf
```

Main concepts

A commit

… a recorded set of changes in your project's files

A repository

… the history of all your project's commits

The staging area

… where every commit is prepared - changes must be added here before they can be committed

- A **version control system** is a tool that keeps track of all changes, effectively recording snapshots of different versions of our files.
- You decide which changes will be included in the next snapshot (each record of these changes is called a **commit**)
- The version control system keeps useful metadata about each commit.
- The complete history of commits for a particular project and their metadata make up a **repository**.
- Repositories can be kept in sync across different computers, facilitating collaboration among different people.

Git workflow

Current state of repository → Staging area → Tracked changes (commit)

Configuring Git

Open a Terminal and type the following command

```
git config --list
```

(exit menu by typing `q`)

If your details are not listed:

```
git config --global user.name "My Name"

git config --global user.email "name@somewhere.ac.uk"
```

Let's create a new project: "Time Travel"

Create new project folder

```
cd ~/Desktop
mkdir time_travel
cd time_travel
```

Initialise Git

```
git init
```

Check folder contents

```
ls
ls -a
ls -la
```

Check Git status

```
git status
```

Part 2 - Standard Git Commands

General command structure

```
git <verb> <options>
```

for example: `` `git add --all` ``

```
git <noun> <options>
```

for example: `` `git status` ``

Where does Git store information?

ℹ️ For each project (repository), Git stores all its files in a subdirectory called `` `.git` ``.

Tracking changes

Let's create a new file

```
touch time_machine.md
ls
```

Open and edit the file

- … with any text editor (nano, Sublime text, vim, emacs, ~~MS Word~~, VS Code, RStudio, etc.)
- add a title and a short description:

  e.g. *Project Time Travel – How I plan to tell my younger self which netflix series are not worth watching!*
- save changes

Check status

- Switch back to the Terminal and use command:

```
git status
```

Preparing to save changes: Staging Area

Add a file to the staging area

```
git add time_machine.md
```

Check status

```
git status
```

Try the help for `git status`

```
git status --help
```

What does this do?

```
git status -s
```

Saving changes: Commits

Track changes (add to version control record)

```
git commit -m "my first commit"
```

Check status

```
git status
```

Exercise:

0. (between each step: check current state with `` `git status` ``)

1. make new changes to the file `` `time_machine.md` ``
2. stage and commit new changes (describe changes in commit message)
3. repeat 1-2
4. create a new file `` `new_physics.md` ``
5. edit `` `new_physics.md` ``
   (e.g. *To make time travel work: need to find new laws of physics! – Where did Newton and Einstein go wrong?*)
6. add `` `new_physics.md` `` to the git record (with a new commit)
7. make changes to both `` `time_machine.md` `` and `` `new_physics.md` ``
8. stage and commit all changes
9. check the commit history with `` `git log` `` (exit log by hitting `` `q` ``)
10. create a new subdirectory with a new file and commit

Comparing new changes to previous version

1. Make a small change to `time_machine.md`

(delete a word, add a sentence)

2. Check what has changed since most recently saved version

```
git diff

git diff --staged
```

Quick summary

- `git status` shows the status of a repository.
- Files can be stored in a project's working directory, the staging area (where the next commit is being built up) and the local repository (where commits are permanently recorded).
- `git add` puts files in the staging area.
- `git commit` saves the staged content as a new commit in the local repository.
- Tip: Write short, informative commit messages that accurately describe your changes.

Comparing to older versions

```
git diff time_machine.md

git diff HEAD time_machine.md

git diff HEAD~1 time_machine.md

git diff HEAD~2 time_machine.md

git diff <commit ID> time_machine.md
```

Reverting to an earlier version

If file is not in staging area (discard untracked changes)

```
git checkout -- time_machine.md

git checkout HEAD time_machine.md

git checkout <commit ID> time_machine.md
```

If file is already in staging area

```
git reset HEAD time_machine.md
```

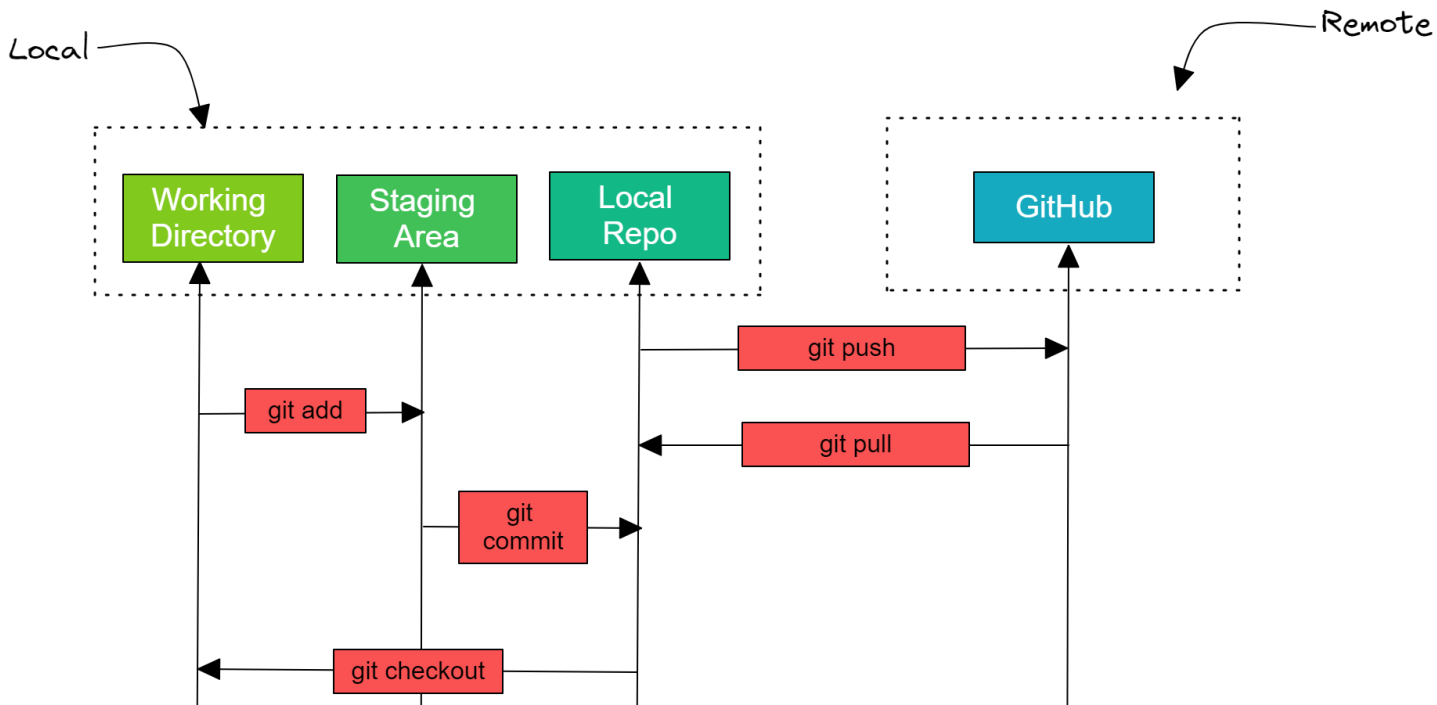
Part 3 - Git Online: GitHub

Why use GitHub?

- easy to set up
- free
- huge community
- dominates open software space
- record of what was changed, when and by whom

From *local* to *global*: Remote repositories

- services like GitHub, Gitlab, BitBucket, etc.
- host a copy of your local repository on a server (in the cloud)



Create a remote repository for the time travel project

1. log into your GitHub account
2. click in top right corner to create a new repository

⚠ *NOTE:* Note: Since we want this repository to be connected to our (existing) local repository, it needs to be empty. Leave “Initialize this repository with a README” unchecked, and keep “None” as options for both “Add .gitignore” and “Add a license.”

3. this essentially creates a folder on GitHub’s servers and initialises Git there

Connect local and remote repositories

4. click on the `SSH` tab
5. copy the URL of your remote repository
6. go back to your *local* repository in the terminal
7. add information about the remote location to your local repository

```
git remote add origin <my_remote_url>
```

8. check that it worked

```
git remote -v
```

Uploading changes to a remote repository

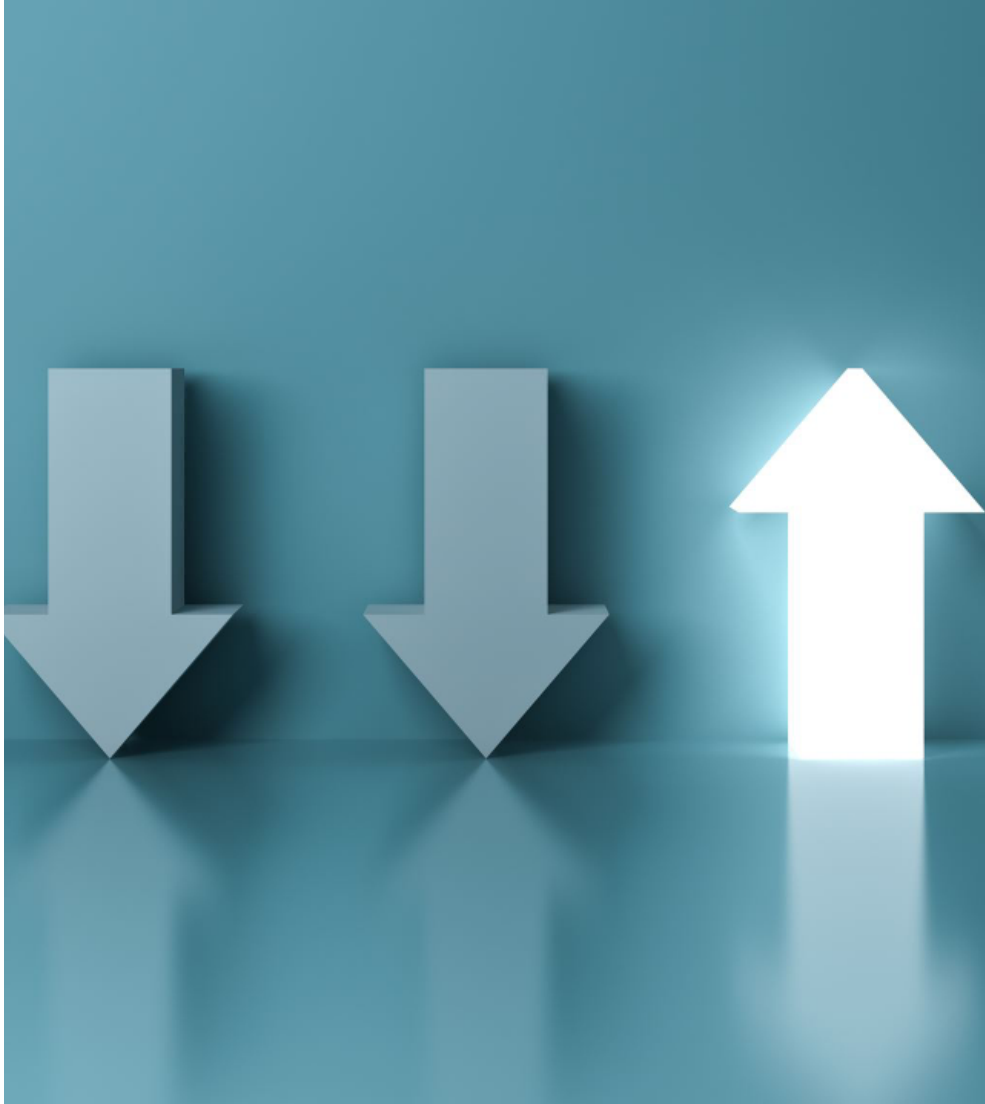
```
git push origin main

git push -u origin main

git push
```

Downloading changes from the remote repository

```
git pull origin main
```



Cloning a repository

- This is the standard way to collaborate with others

```
git clone <url-to-repo>
```

- Note: the remote repository is already set up and called `origin` by default

Exercise: Explore GitHub

1. Poke around the GitHub interface
2. Create an issue
3. Assign yourself to the issue
4. Look at the commit history
5. Edit a file directly on GitHub
6. Pull changes to your local repository

Resolving conflicts (advanced git)

⛔ ISSUE: the *same* section of a file in the local repositiory and the remote repository have both been changed independently

Example scenario:

- person A works on the same code as person B
- person A makes a change and pushes it to the remote
- person B makes a change and wants to push to the remote
- git doesn't know which change to keep

Possible solutions:

- B pulls first from the remote and resolves the conflict locally
- A and B work on different branches
- B creates a merge request (to be resolved on GitHub)

Commands Summary

- `git status`
- `git add <filename>`
- `git commit -m "<my message>"`
- `git log`
- `git diff`
- `git clone`
- `git pull`
- `git push`
- `git remote`

Getting help on any command

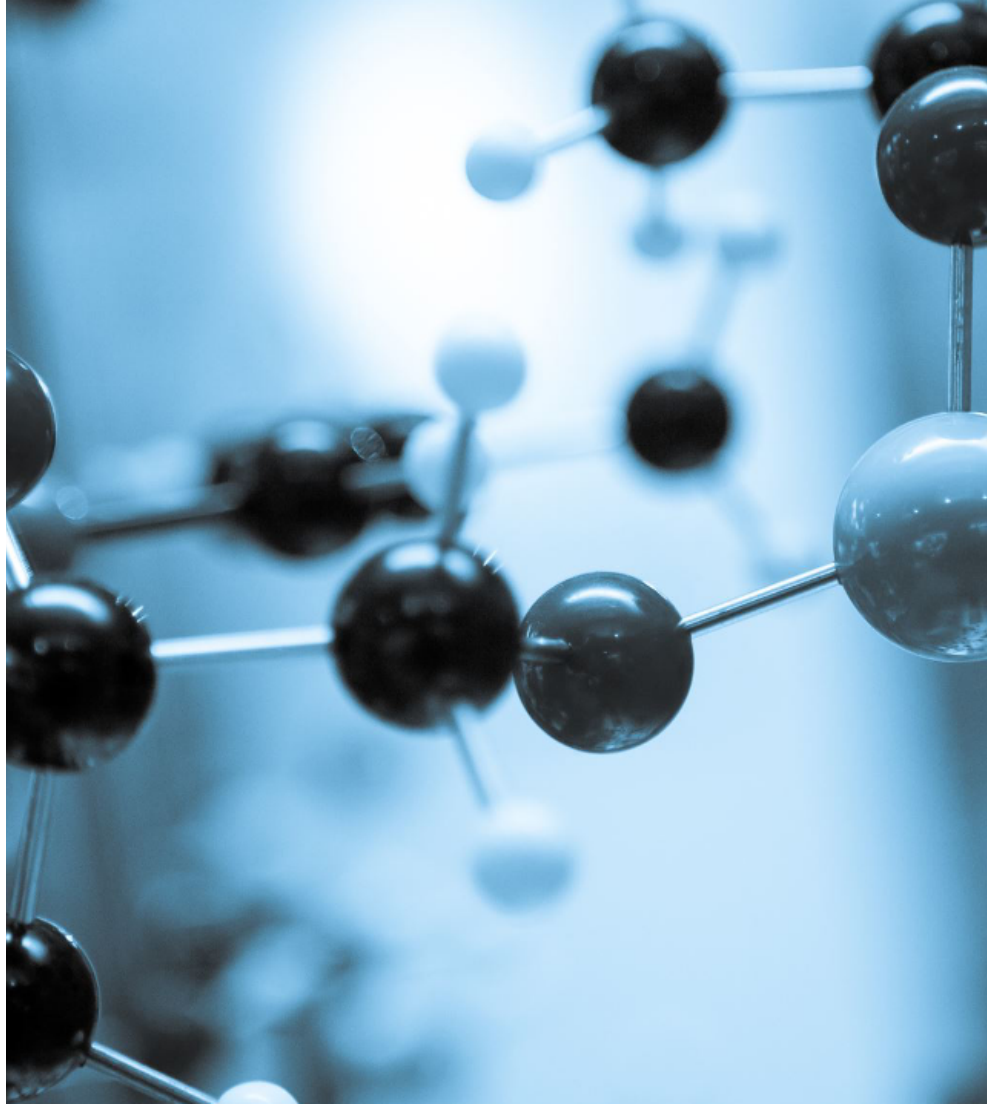
- `git help`
- `git <command> -h`

More advanced commands

- `git branch`
- `git switch`
- `git fetch`
- `git checkout`
- `git rebase`
- …

How does version control benefit Open Science?

- makes code reproducible
- makes it easier to collaborate
- hosted repositories can be cited
- automatically backed-up (distributed servers)
- ...



Git can also be used with a graphical interface

Git integrations in other software

- RStudio
- VSCode
- PyCharm
- Sublime Text
- *and many others …*

GUI apps for Git

- GitHub Desktop
- GitKraken
- SourceTree
- TortoiseGit
- …

Additional Exercises and Resources

1. Create a new repo on GitHub (initialise with README and LICENCE files)
    - clone it to a local directory
    - make changes locally, commit and push to the remote
    - make changes on GitHub, commit and pull to local repo
    - familiarise yourself with GitHub's online functionalities
    - open an issue on GitHub
2. Learn about branches
    - e.g. follow this tutorial
3. Git in Rstudio
    - If you want to use git from within RStudio, see e.g. this tutorial or this tutorial
4. Try a GUI, e.g. GitHub Desktop or GitKraken

Additional Resources

- Continuous integration: run tests automatically every time you push changes to the remote
  - see e.g. here
- `.gitignore`: tell git which files to ignore see e.g. here
- Merge conflicts and how to resolve them
  - see e.g. this tutorial
- Git GUIs:
  - see e.g.g this overview
- a self-paced tutorial app to learn Git: git-it -electron
- check out in-depth tutorials from Software Carpentry (incl. git, shell, python, R, etc.)