



git & GitHub

Workshop

How we learned to love the git

Who are we?



Assistant Professor, Research,
Health Data Science (HeaDS).



Data Scientists, SUND Data
Lab, Centre for Health Data
Science (HeaDS).



Post-doc, Computational Biology
Laboratory, Danish cancer society
Research Centre (DCRC).



Jonas
Sibbesen



Diana
Andrejeva



Henrike
Zschach



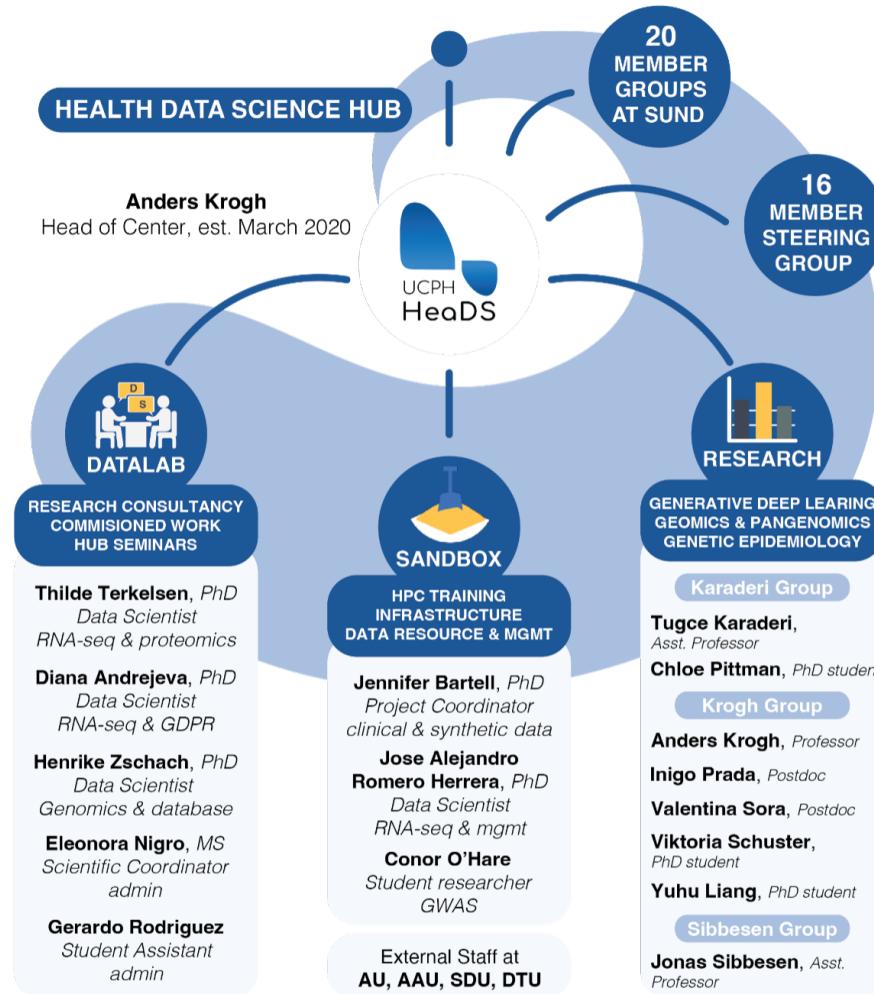
Matteo
Tiberti

HeaDS

Center for Health Data Science

- Conduct Health Data Science research.
- Serve as the UCPH Data Lab.
 - Be the hub for health data science research.
 - Provide SUND researchers with health data science services & training.
- Develop National Health Data Science Sandbox for Training and Research

<https://heads.ku.dk/>





SUND DATA LAB



Diana Andrejeva



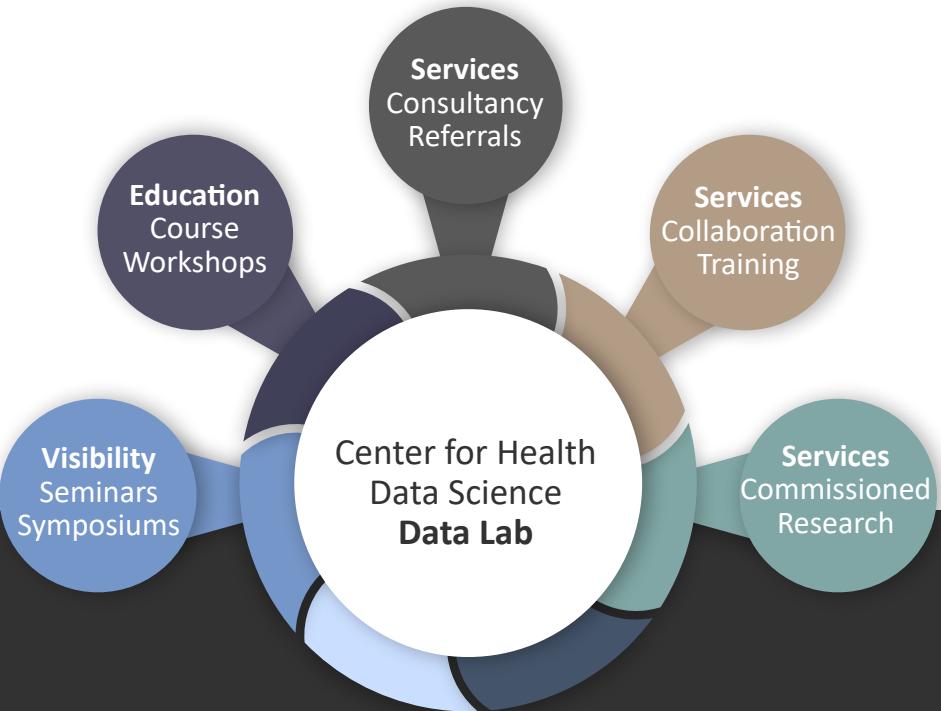
Henrike Zschach



Eleonora Nigro



Thilde Terkelsen



Website - <https://heads.ku.dk>

Email - datalab@sund.ku.dk

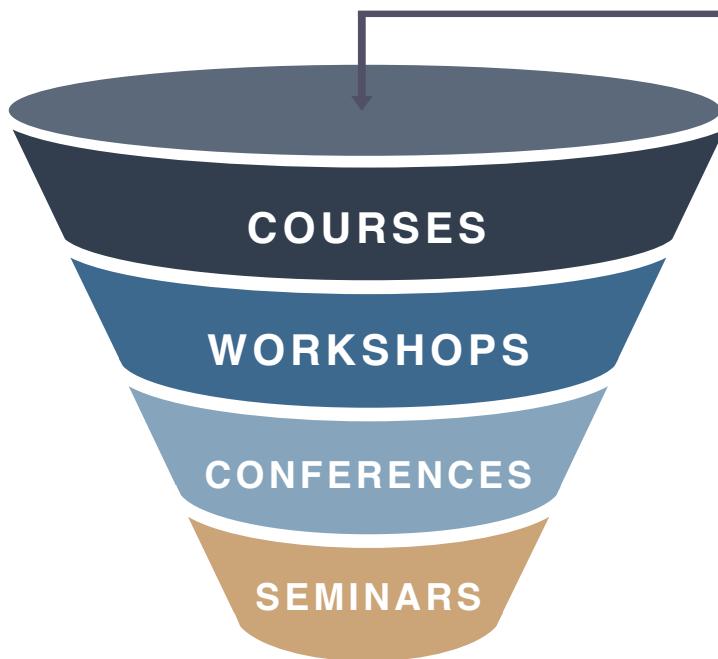
Follow us on **Twitter** at [@ucph_heads!](#)

Slack channel and **mailing list**

courses, conferences, research activities, job postings, etc. within the field of data science.



Center for Health Data Science



git/Github Workshop (Reproducible Science):
Today.



From Excel to R (module 1): **March & June**, 2023.



Python Tsunami: **March**



GDPR and personal data in biomedical
research: **April**



Introduction to RNA-seq analysis, **January & June**



Program

9.00-9.45: Theoretical introduction

9.45-10.15: Creating a git repository

10.15-10.30: Coffee Break

10.30-11.15: Making changes and checking history

11.15-12:00: Creating and merging branches

12.00-13.00: Lunch break

13.00-13.30: Going to the past and fixing mistakes

13.30-14.00: Interacting with a remote repository

14.00-14.30: Working together on GitHub

14.30-14.45: Coffee Break

14.45-15:30: Git & RStudio

15.30-16.00: Q & A (and help with repository for own code)

Resources

- Git Pro Book:
<https://git-scm.com/book/en/v2>



- Introduction to git for bioinformaticians:
<http://ponderomatics.com/git.html>



- Get started with SourceTree:
<https://confluence.atlassian.com/get-started-with-sourcetree>



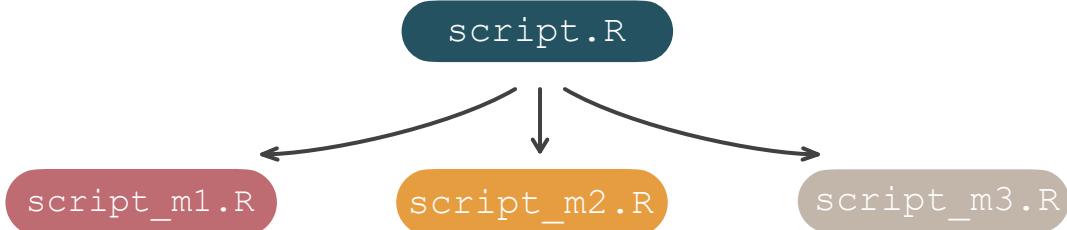
- CodeRefinery website:
<https://coderefinery.org/lessons/>



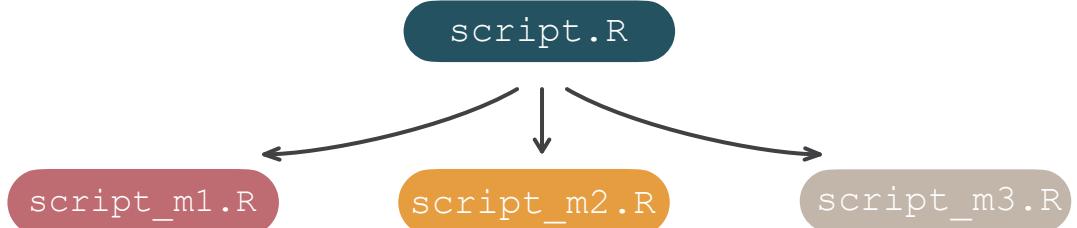
Justification

script.R

Justification

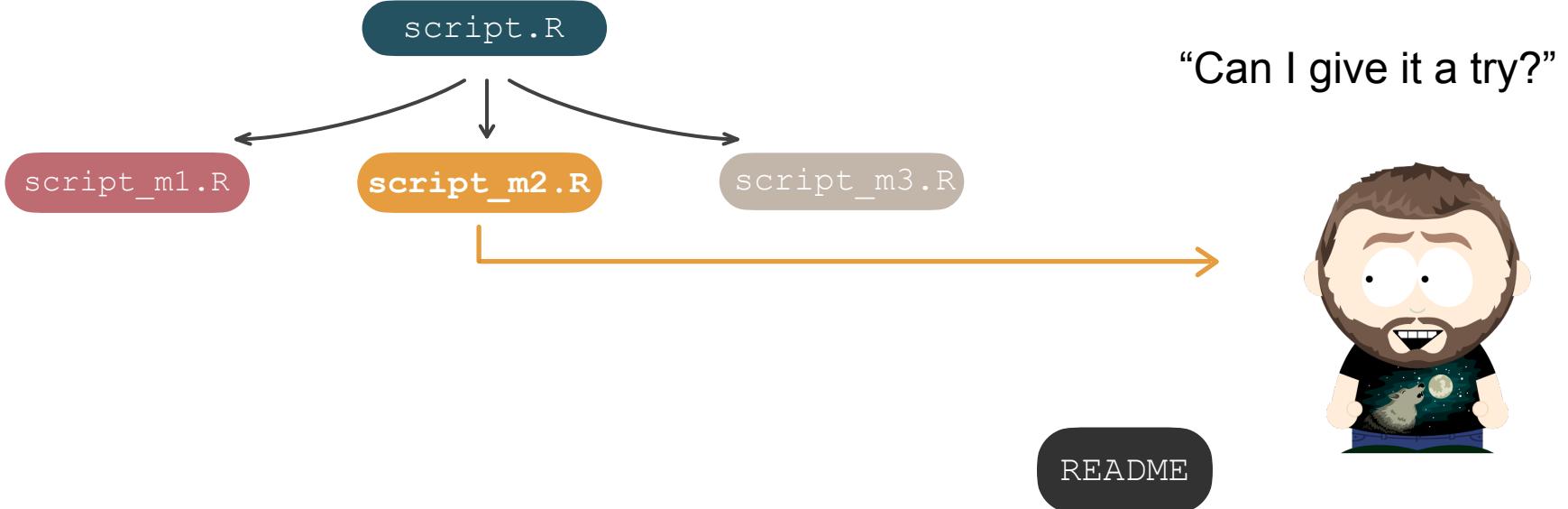


Justification

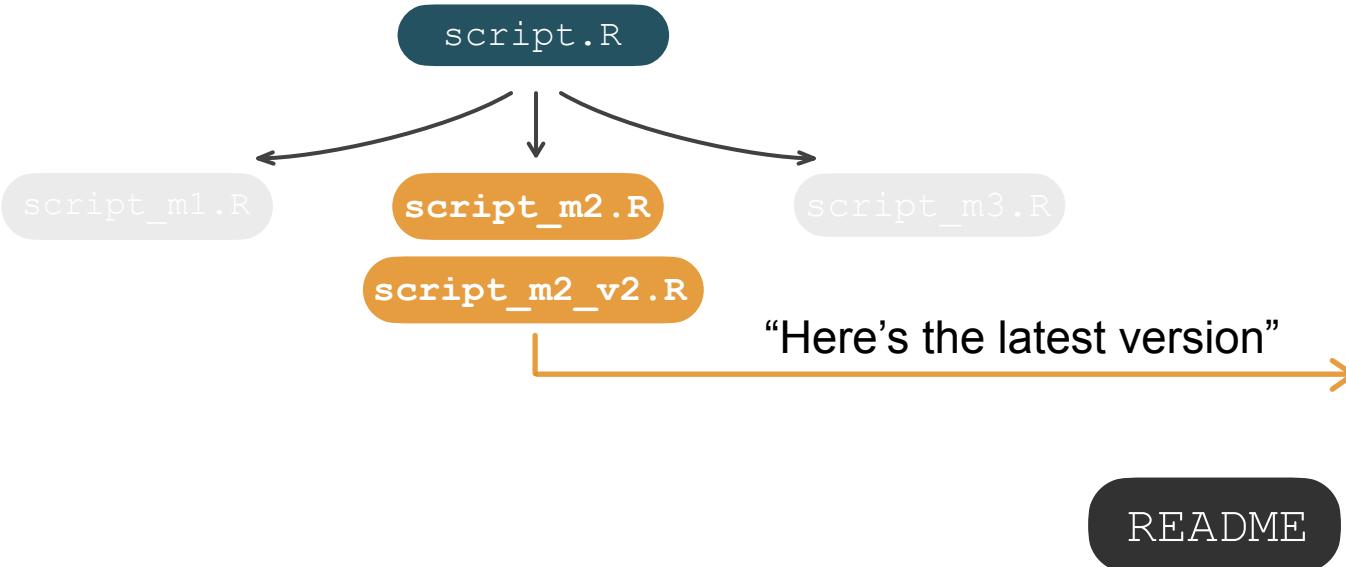


README

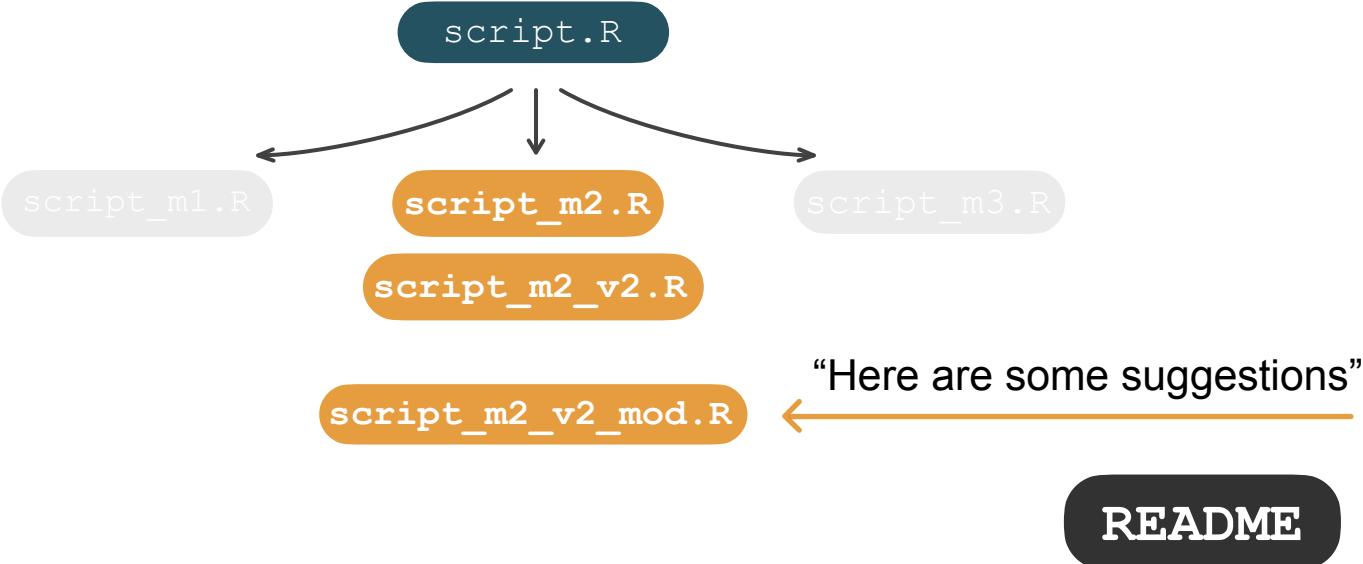
Justification



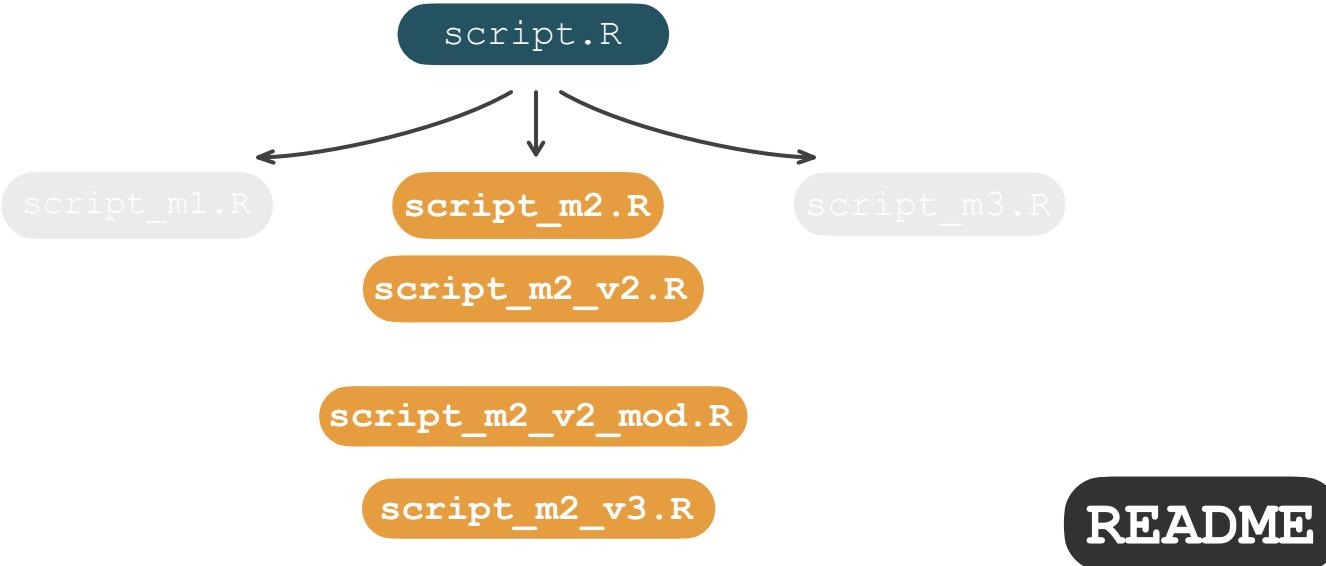
Justification



Justification



Justification



Justification

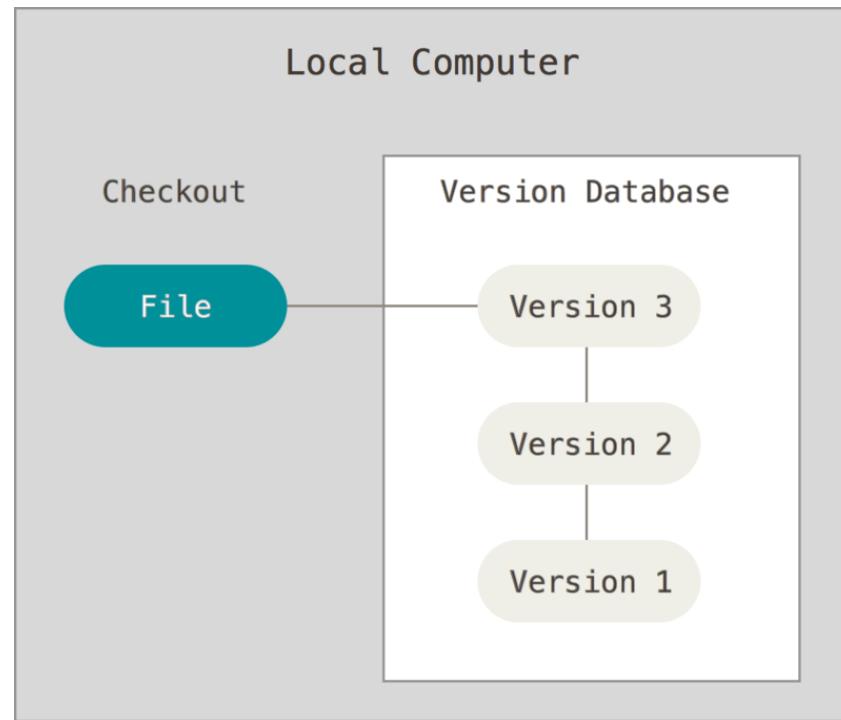
Keeping track of manually means...

- Significant overhead
 - Organization
 - Documentation
 - Communication
- Error-prone and time-consuming
- Makes collaboration difficult
- Six months from now...???



Version control systems

"Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later (**history**)"



Version control systems

Referring to and tracking back to specific versions

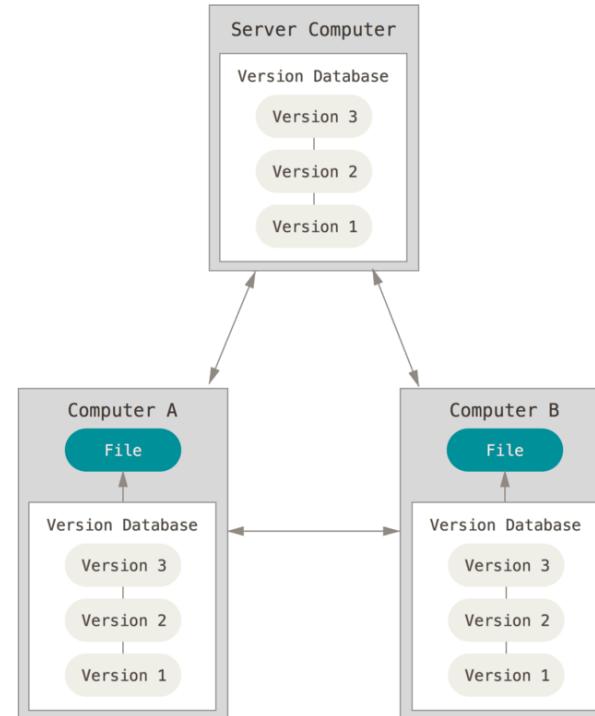
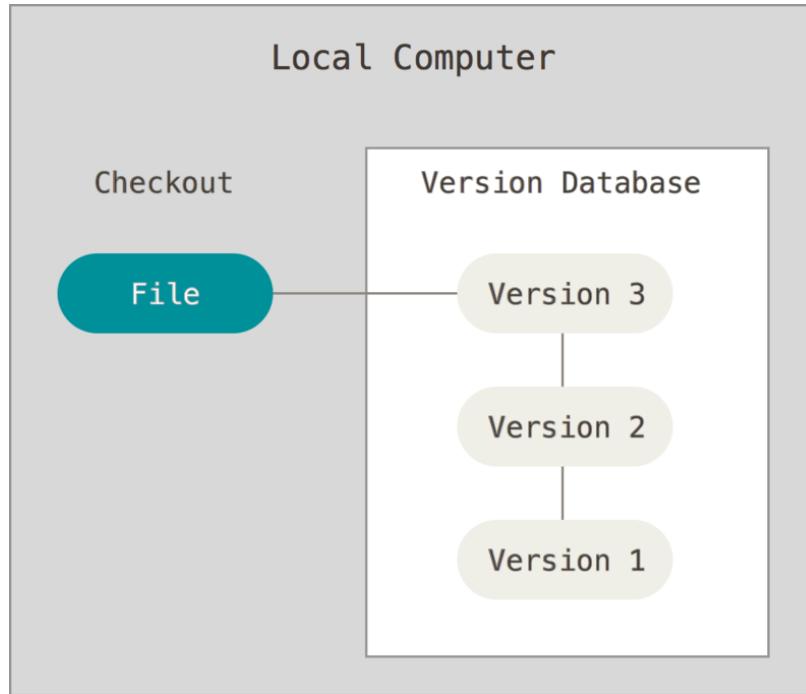
- When was “that” change introduced (backtracking)?
- Includes metadata (who, what, when, why)

Be organized and maintain long-term control over your codebase

Keep several versions of code at the same time, without clutter

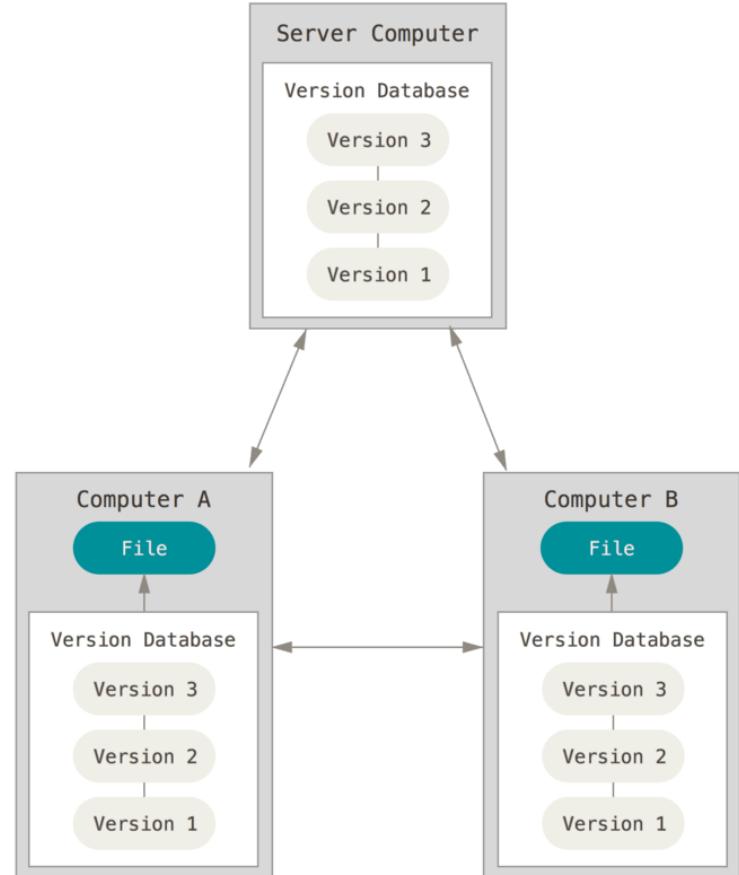
- Keeps known working versions safe
- Experimenting is easy and organized
- Experiments are self-contained
- Experiments can be kept, removed, or merged with the “main” codebase

Distributed version control systems



Distributed version control systems

- No single point of failure
- The system keeps everyone synchronized...
- ...but asynchronous work is possible
- Group-based model - shared access is default
- Shared responsibility of being up to date



git and GitHub

Git is a DVCS software, created by Linus Torvalds for the management of the Linux kernel.

Other clients exist.

<https://git-scm.com>

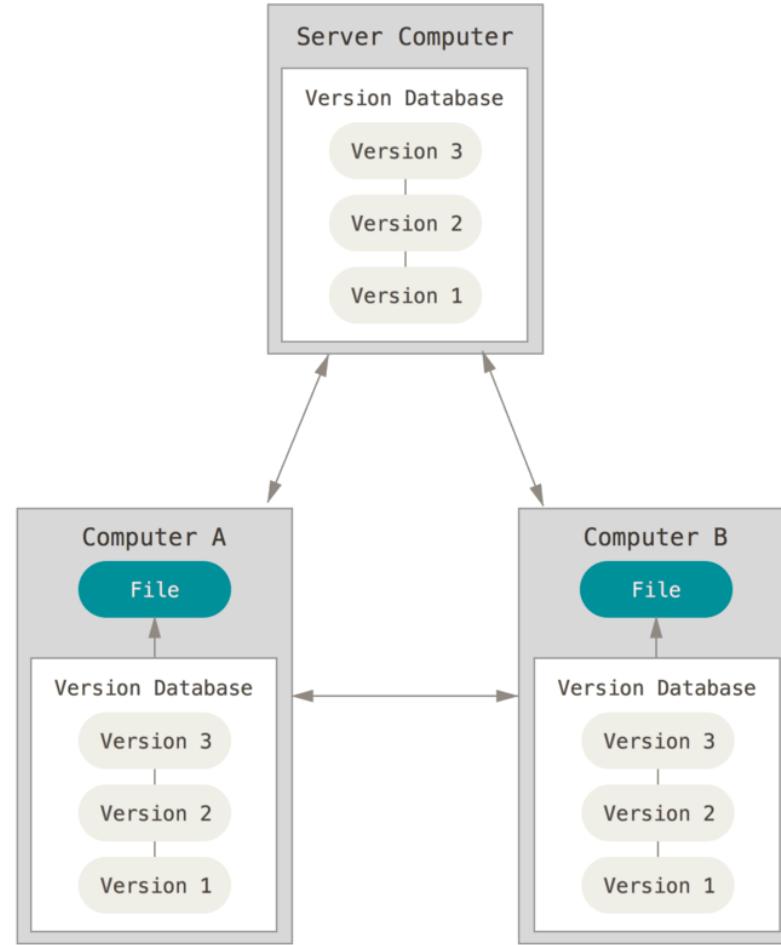


GitHub is a web-based hosting service for version control that uses git

Other services exist.

<https://github.com>

git and GitHub



Useful concepts

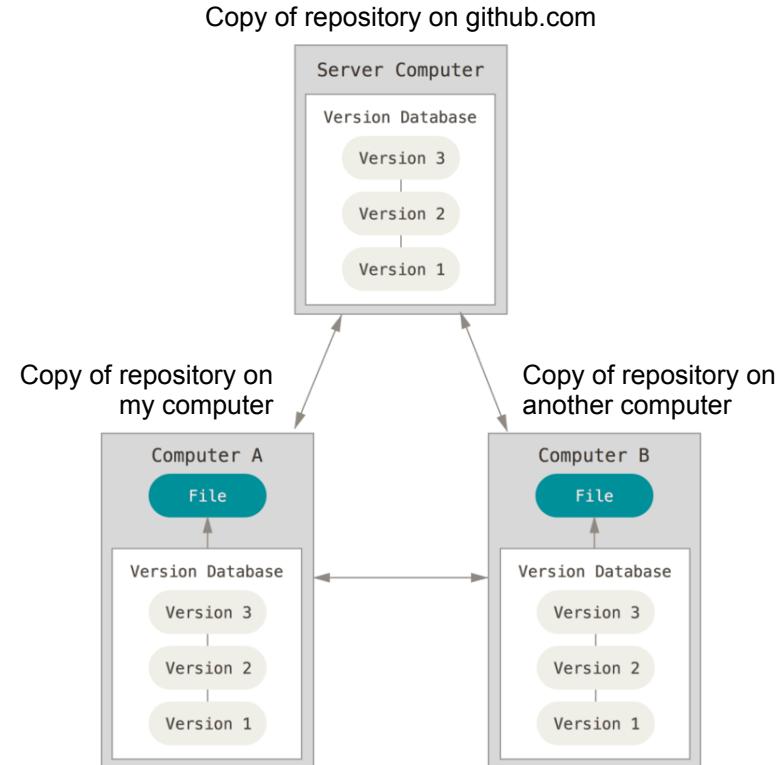
File tree (working directory):

All the files (organized in directories) that we want to keep track of, originating from a single directory.

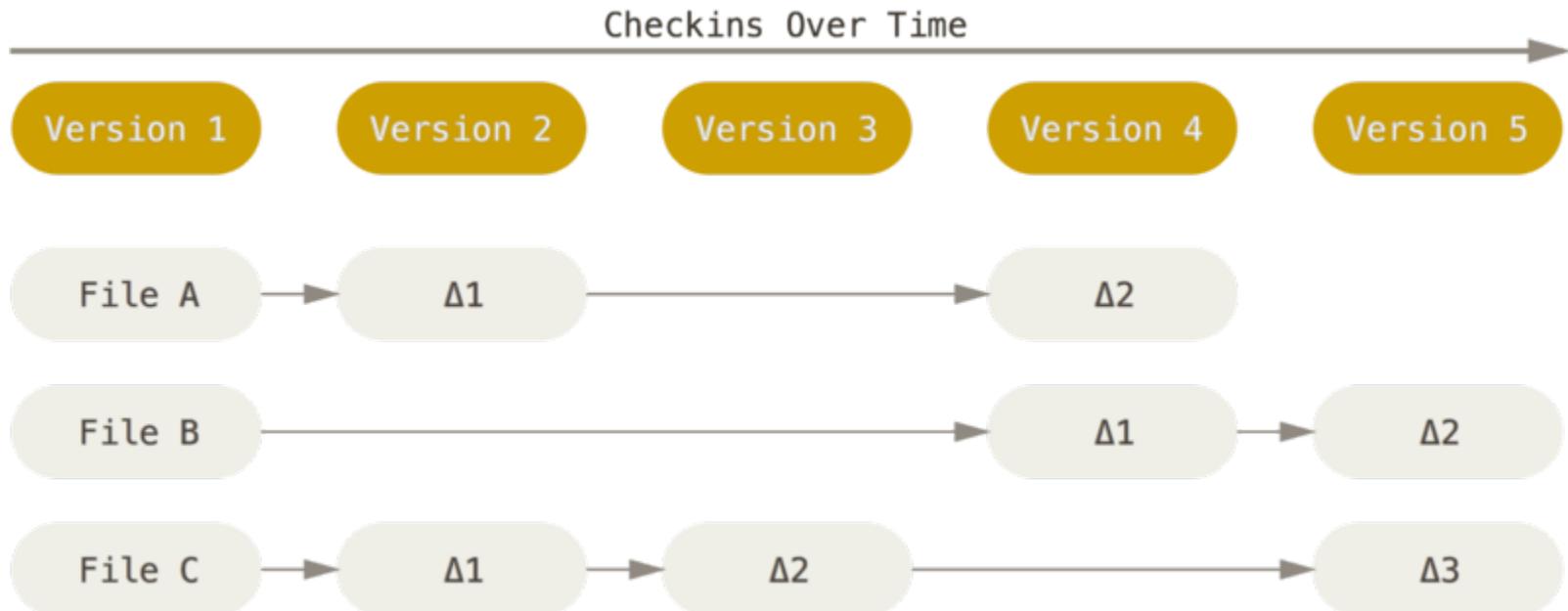
Repository (repo)

Place in which changes to the file tree are recorded.
Can be local or remote.
Several copies can exist.

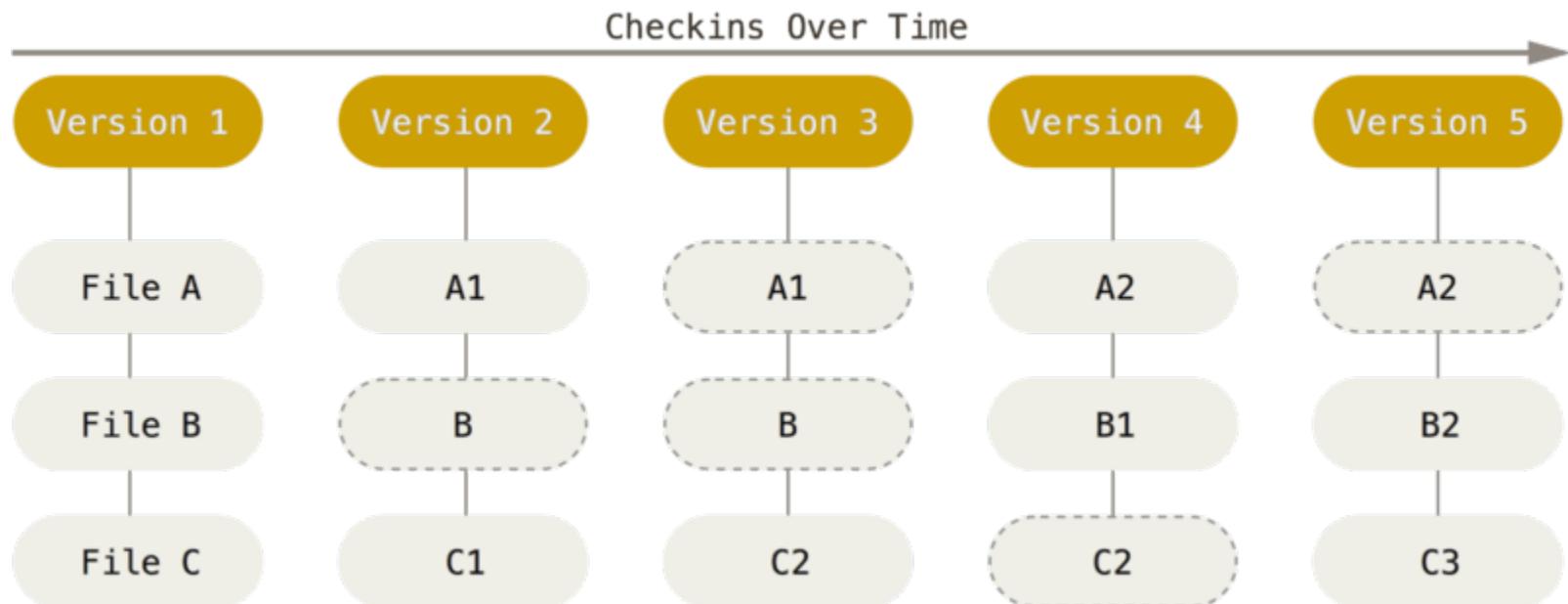
NOTE: We will be working on our local copy first,
meaning that there will be no changes on the repos on
GitHub, until we decide to synchronize.



How are differences stored?



Git stores snapshots of the file tree



Why GitHub?



Helpful web interface:

- Visually browse and explore
- Community aspect (Issues, Wiki, references ...)
- Granular access control

Public space: Track records, visibility, transparency, reproducibility

Easy fork/merge system; easy contributing to projects

Remote repository; backup

Shared responsibility of being up to date

GitHub

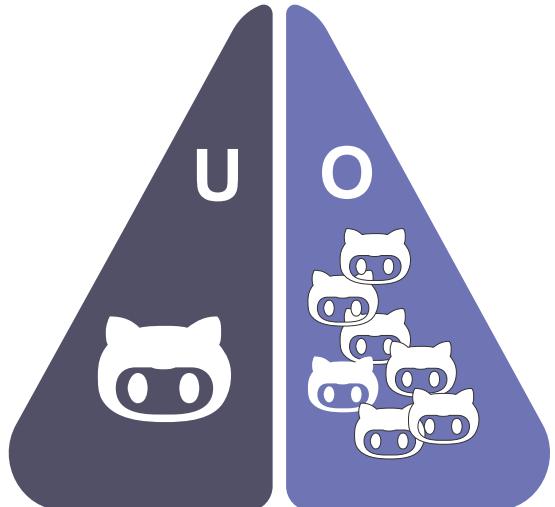
GitHub is organized in

U User accounts:

Personal repositories that have a single owner.

O Organizations:

- Designed to better accommodate the needs of a team (ex. lab)
- One or multiple owners and sophisticated access control systems to the repos.



GitHub

GitHub Repositories can be:

PUBLIC

- Accessible to everyone
- Usually released with an open source license
- Still possible to have control over how many people interact with the repo.

PRIVATE

- Not accessible to the public.
- Accessible only from allowed accounts – requires password.

NOTE: Organizations have further options to limit and organize access to repositories

GitHub

The screenshot shows the GitHub pricing page with three main plans: Free, Team, and Enterprise.

Free
The basics for individuals and organizations

- Unlimited public/private repositories
- 2,000 Actions minutes/month
Free for public repositories
- 500MB of Packages storage
Free for public repositories
- Community support

Team
Advanced collaboration for individuals and organizations

- Everything included in Free, plus...
- Protected branches
- Multiple reviewers in pull requests
- Draft pull requests
- Code owners
- Required reviewers
- Pages and Wikis
- 3,000 Actions minutes/month
Free for public repositories
- 2GB of Packages storage
Free for public repositories
- Web-based support

Enterprise
Security, compliance, and flexible deployment

- Everything included in Team, plus...
- Automatic security and version updates
- SAML single sign-on
- Advanced auditing
- Github Connect
- 50,000 Actions minutes/month
Free for public repositories
- 50GB of Packages storage
Free for public repositories

EXCLUSIVE OPTIONS

- Token, secret, and code scanning
- Premium support

Costs:

- Free:** \$0 per month
- Team:** \$4 per user/month
- Enterprise:** \$21 per user/month

[Create a free organization](#) [Continue with Team ▾](#) [Contact Sales](#) [Start a free trial](#)

**Academics get free
“Team” plan!**

A look at git and GitHub

```
git-GitHub-workshop-2022 -- zsh -- 101x30
Last login: Mon Feb  7 10:29:21 on ttys000
[kgx936@SUN1007442 ~ % cd Desktop/git-GitHub-workshop-2022
[kgx936@SUN1007442 git-GitHub-workshop-2022 % ls
ChocolateCake README.md
[kgx936@SUN1007442 git-GitHub-workshop-2022 % git status ChocolateCake/ingredients.txt ]
```



I gang med Firefox Hjem - KUNet AdminJump EditJump

https://github.com/Center-for-Health-Data-Science/git-GitHub-workshop-2022

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

Center-for-Health-Data-Science / git-GitHub-workshop-2022 (Private)

Unwatch 2 Fork 0

Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags Go to file Add file Code About

No description, website, or provided.

Readme 0 stars 2 watching 0 forks

5 commits

more vanilla needed 6a4bdb5 14 seconds ago

ChocolateCake more vanilla needed 14 seconds ago

README.md Update README.md 7 minutes ago

README.md

git-GitHub-workshop-2022

This repository is used for the HeaDS git and GitHub workshop 2022. Workshop participants should clone the repo after which they can use it to complete workshop exercises associated. The shared repo is made for hosting cooking recipes which are made locally by participants and then pushed to remote for everyone to view.

HeaDS workshop contacts:

Releases

No releases published Create a new release

Packages

No packages published Publish your first package



Let's create a GitHub repo

The screenshot shows a GitHub user profile for "Jonas Andreas Sibbesen" (jonassibbesen). The profile includes a large circular profile picture, a bio, follower counts, achievements, and organization links. The main dashboard displays a list of "Popular repositories" and a "Contribution grid" showing activity over the last year. A red box highlights the "New repository" button in the top right corner of the header.

Profile Information:

- Search or jump to...
- Pull requests Issues Marketplace Explore
- New repository (highlighted by a red box)
- Import repository
- New gist
- New organization
- New project

Popular repositories:

- rpgv** Public
Method for inferring path posterior probabilities and abundances from pan-genome graph read alignments
C++ ⭐ 12 2
- vg** Public
Forked from vgteam/vg
Tools for working with genome variation graphs
C++
- vgrna-project-paper** Public
Bash scripts and data used in pantranscriptomic paper
Shell ⭐ 1
- BayesTyper** Public
Forked from bioinformatics-centre/BayesTyper
A method for variant graph genotyping based on exact alignment of k-mers
C++
- vgrna-project-scripts** Public
Scripts used for pantranscriptome analyses
R ⭐ 1
- xg** Public
Forked from vgteam/xg
xg0: a simpler xg index
C++

Achievements:

Organizations:

Contribution grid:

330 contributions in the last year

Contribution settings ▾

Less More

Learn how we count contributions

Let's create a GitHub repo

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *  Center-for-Health-Data-Science /

Repository name *

Great repository names are short and memorable. Need inspiration? How about [refactored-sniffle?](#)

Description (optional)

 Public
Anyone on the internet can see this repository. You choose who can commit.

 Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

README file

File shown by default when someone visits the repository – good for an introduction to what the repository is about. Git markdown format.

.gitignore

Hidden file containing specifications of files/types that should be ignored at all times by git. There are some default choices.

LICENSE

Creates a LICENSE file that contains the license of the content of your repository.

<https://choosealicense.com>

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

{ Which of the following best describes your situation? }



I need to work in a community.

Use the [license preferred by the community](#) you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to [add a license](#).



I want it simple and permissive.

The [MIT License](#) is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

[Babel](#), [.NET Core](#), and [Rails](#) use the MIT License.



I care about sharing improvements.

The [GNU GPLv3](#) also lets people do almost anything they want with your project, except distributing closed source versions.

[Ansible](#), [Bash](#), and [GIMP](#) use the GNU GPLv3.

git Syntax

We will use the terminal command git. Syntax:

```
git <command> [<options>] [<args>]
```

Args are usually names or files or directories.

Individual commands have help pages:

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

Examples:

```
git add file.txt
```

```
git commit -m "changes were done"
```

```
git branch -d name
```

Getting started (first time user)

Configure git with your identity:

```
git config --global user.name <username>  
git config --global user.email <example@example.com>
```

This is important if you want to associate your local changes with your GitHub account. If you want to keep your email private see this guide: <https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address>

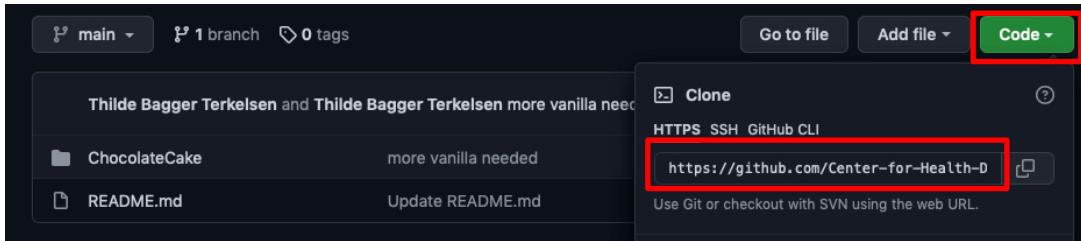
Change the default text editor used by git (for example to Emacs):

```
git config --global core.editor emacs
```

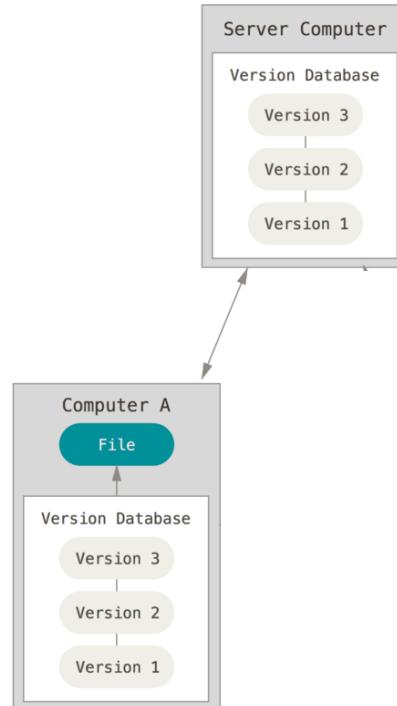
On a Windows system you must specify the full path to the text editor's executable file.

Cloning a GitHub repository

We need a local copy of our repository :



```
git clone <GitHub link>  
cd <your repo>
```



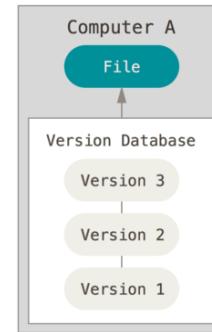
Creating a git repository

You could create it from scratch as well:

```
mkdir my-repo; cd my-repo  
git init
```

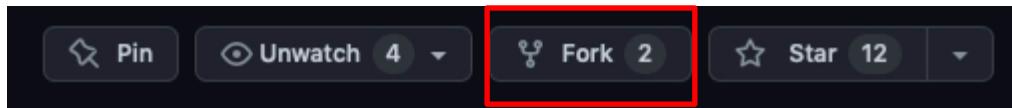
N.B: that in this case we would not have a remote! Creating a repository on GitHub and cloning it ensures that:

- We have a remote repository
- Remote and local copy are linked



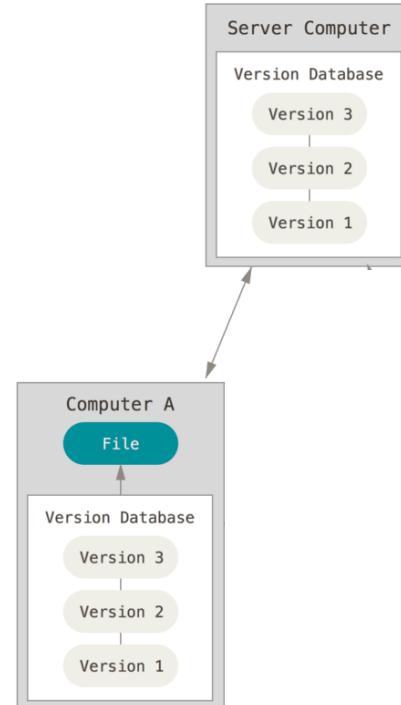
Forking a GitHub repository

Create your own copy of someone else's repository



Forks are most commonly used to:

- Propose changes to another person's repository
- Use another person's repository as a starting point for own project



Checking status of a git repo

- Check the current state of the repository:

```
git status
```

- See the name and location of the remote repository:

```
git remote -v
```

Exercise 1

Exercise

1

- 1. Configure git with username and email if you have not done so before.
- 2. Create a new public repository on GitHub to hold your favourite cooking recipes.
- 3. Create a local copy of your GitHub repository by **cloning** it in a folder on your computer. Make note of the location of the directory that you choose to host your local copy.
- 4. Move to the local repository. Check the status of this repository (don't worry if all the information does not make sense yet). Check name and location of the remote repository.

Useful concepts

Commit:

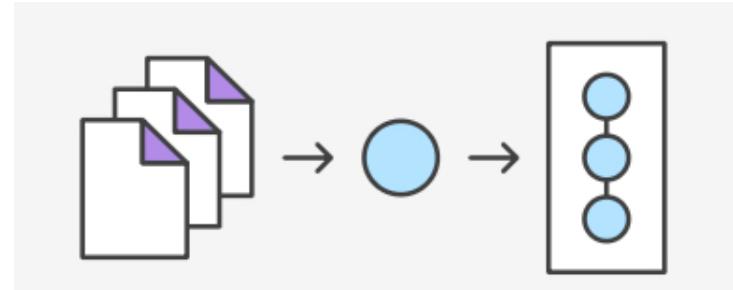
Snapshot of a particular state of a file tree.

Stage:

Selection of changes made on a repository to be added to a commit.

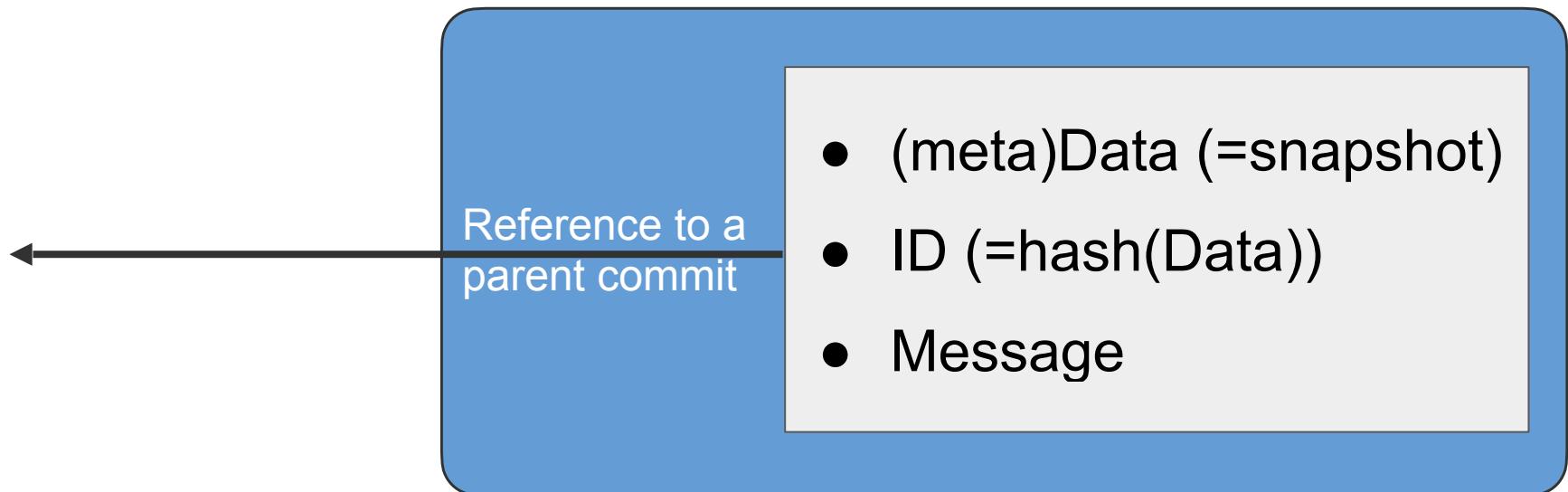
head & branch:

Reference (“pointer”) to a particular commit.



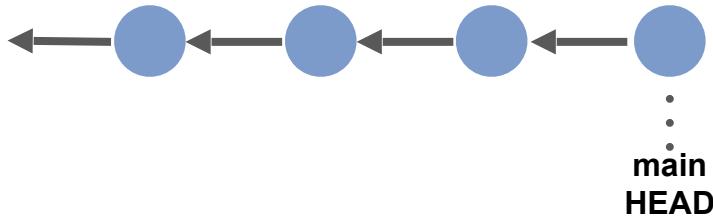
Useful concepts

Anatomy of a commit



Useful concepts

Anatomy of a commit



origin = original remote repository

main = main branch (local and remote)

HEAD = current commit you are viewing

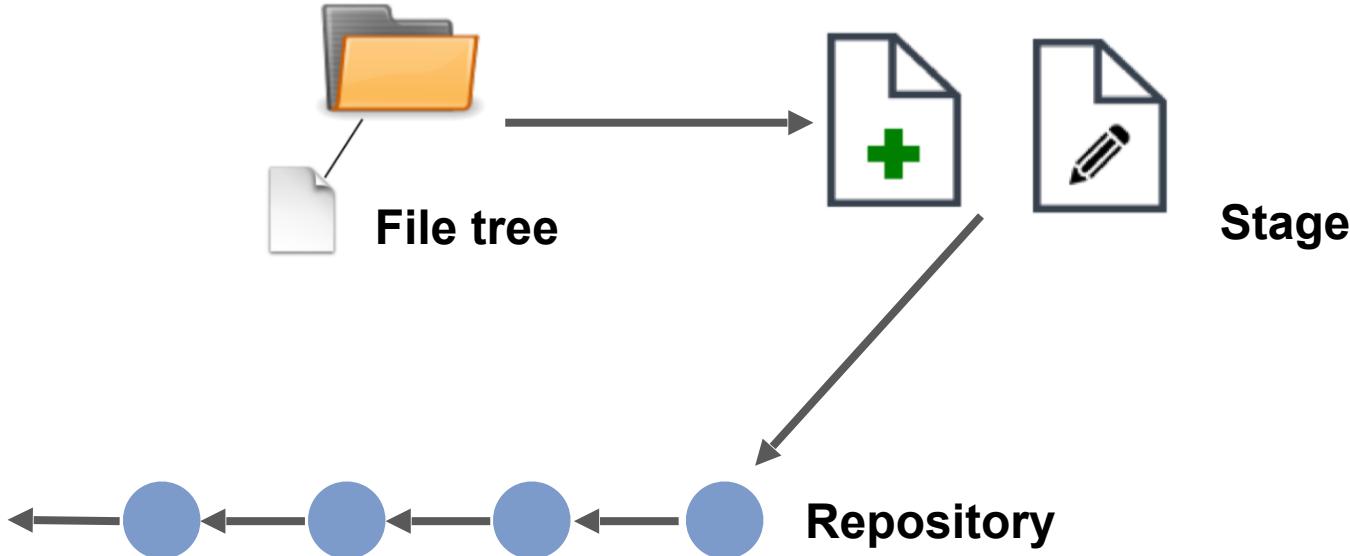
Be aware of the “**HEAD**” head!

It is the head referring to the commit currently checked out in the file tree

master = previous name for remote main (changed in 2020)

Useful concepts

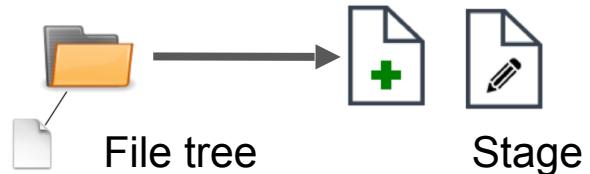
Workflow of a commit



Adding files to stage

Files are added to stage using the command **add**

```
git add <filename>
```



- Allows to stage files or even directories (if they contain files!)
- Option `-A` stages all changes if no arguments are given
- More than one filename are allowed including wildcards

```
git rm <filename>
```

```
git mv <filename>
```

When you're happy with your changes and have add all the files to stage the next step is to commit the changes:

```
git commit
```

```
git commit -m "Commit Message"
```

- Use `-a` to automatically stage files that have been modified and deleted. **New files you have not told Git about are not affected.**

```
git commit -a -m "Commit Message"
```

First Commit

Good Practices

- **Commits** should be:
 - Atomic and self-contained (or WIP), but not excessively pedantic
 - Done often (try not to have gigantic blob commits)
- **Commit messages** should:
 - Always have a short descriptive comment (no “bugfix”, “save”, etc.)
 - Written to be understandable by someone else >1 year down the line, including you :)

Checking history

To check the history of a repository:

```
git log
```

A few useful options:

- To see only the commits of a certain author

```
git log --author="SomeAuthor"
```

- To see a very compressed log where each commit is one line

```
git log --oneline
```

- To see a more complete history in a graph form:

```
git log --all --decorate --graph --oneline
```

Checking differences

Check

- not staged changes since the last commit:

```
git diff
```

- staged changes since the last commit:

```
git diff --cached
```

- differences between two commits:

```
git diff <ID1> <ID2>
```

Exercise 2

Exercise

2

- 1. Add a recipe to your repository by creating an *ingredients.txt* and a *recipes.txt* file (can contain anything you would like) in the repository folder.
- 2. Check the status of the repository. Stage the new files using **add** and check the status. **Commit** the staged files and check the status. What did you see change?
- 3. Create a directory with the recipe name and move your files to it; create a commit with your changes.
- 4. Add or remove one step in *recipes.txt*; create a commit with your changes.

Exercise 2 cont.

Exercise

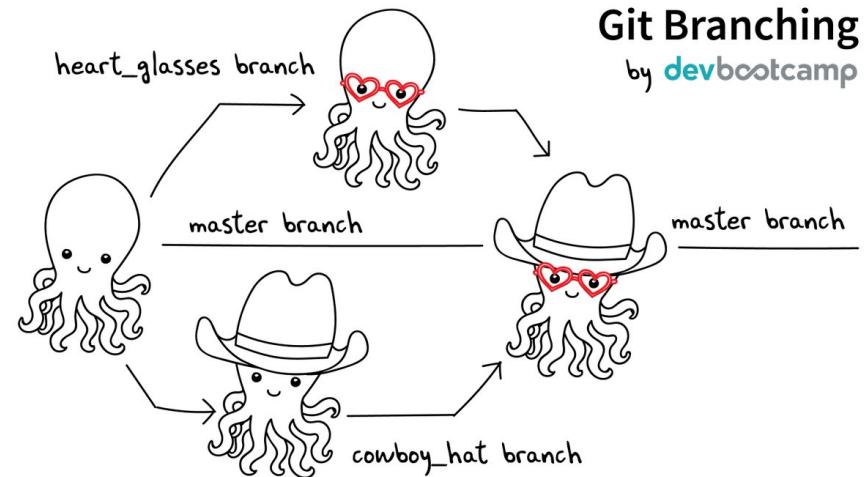
2

- 5. Check the commit history. Can you identify the author, ID, time and date of your commits? Try experimenting with the different ways of showing the history.
- 6. Check the difference between different commits. Can you identify the filename(s) and changes?

Branching

A **branch** represents an independent line/track of work on your repository.

- Keep a branch that is in working order (main branch).
- Isolate experiments from the main branch.
- Work on more than one feature at the same time, independently.
- Merge changes we like into the “main” branch.



- List of all the branches in your repo

```
git branch synonymous to
```

```
git branch --list
```

- Create a branch named <branch>

```
git branch <branch>
```

- Switch to a branch

```
git checkout <branch>
```

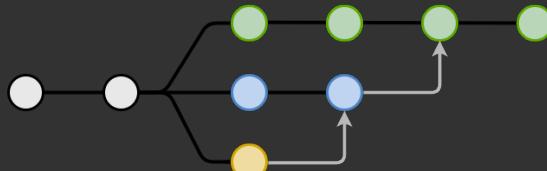
- Safe deletion: prevents from deleting
the branch if it has unmerged changes

```
git branch -d <branch>
```

- Force delete

```
git branch -D <branch>
```

Branching

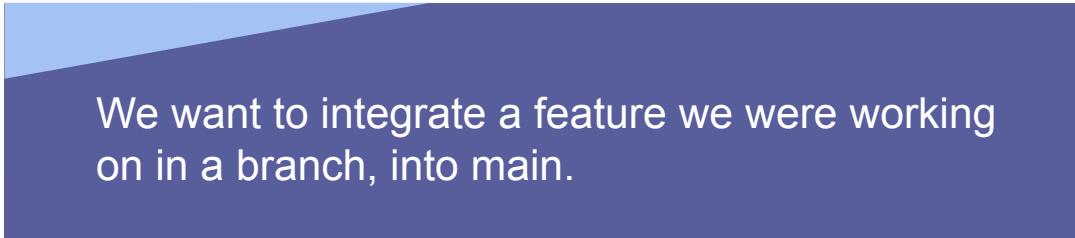


Good Practices

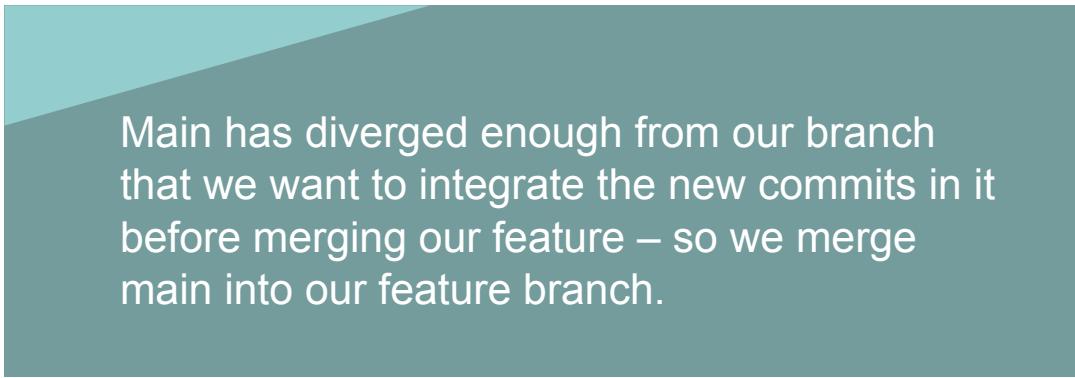
- Branches should:
 - Have descriptive names.
 - Done every time you want to experiment or add a new feature, bugfix etc.
 - Have a single purpose (one feature, one bug, one change).
 - Derived from a sensible commit (often the latest commit in the main branch)

Merging

Two main cases for merging



We want to integrate a feature we were working on in a branch, into main.



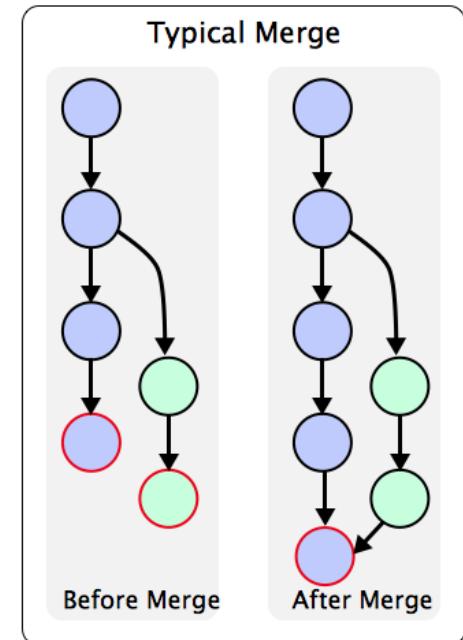
Main has diverged enough from our branch that we want to integrate the new commits in it before merging our feature – so we merge main into our feature branch.

Merging

Merging incorporates changes made in another commit/branch **into the current/active branch**.

- **Switch** to the branch you want to change (not the one you want to merge!)
- **Merge** incorporates the changes made by the other branch **into the current/active branch**.

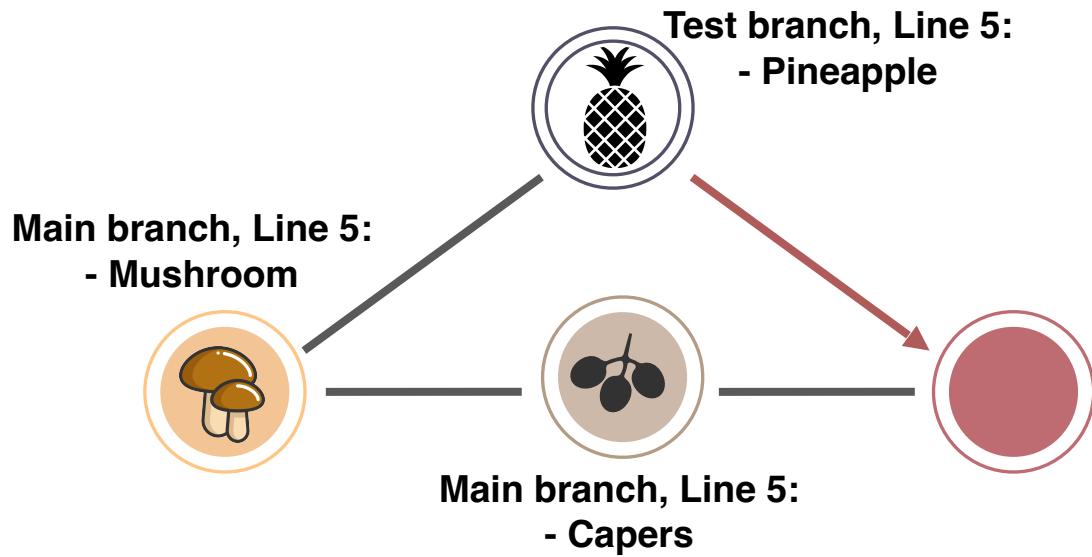
```
git checkout <active branch>  
git merge <branch to merge>
```



Conflicts

A merge **conflict** happens when trying to merge branches that have changes in the same files and in the same lines

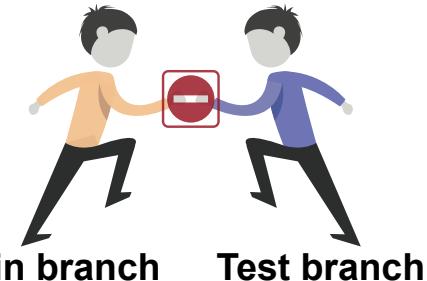
Merge test into main: which version should we keep?



Conflicts

```
(base) Matteos-MacBook-Pro-2:test matteotiberti$ git merge test
Auto-merging recipe.txt
CONFLICT (content): Merge conflict in recipe.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Check which file(s) have conflicts
- Modify files by hand to achieve the desired result.
- Add all files once they are ready and commit.
This solves the merge in a commit.
- **Conflict markers!** Remember to remove all markers and change your code in the final version you want.



```
- tomato
- mozzarella
- flour
- water
<<<<< HEAD
- capers
=====
- pineapple
>>>>> test
```

Exercise 3

Exercise

3

- 1. Create an **experimental** branch and switch to it. Change an ingredient in *ingredients.txt* and store the change as a commit. Try to get a list of the branches in the repository.
- 2. Go back to your main branch and add a step in *recipes.txt* and store the change as a commit. Check the difference between the two branches (use `git diff`).
- 3. Merge your **experimental** branch into your **main** branch (it will ask you to write a commit message). Check the history (try using the graph mode).

Exercise 3 cont.

Exercise

3

- 4. Switch to your **experimental** branch and change the first line in one of your files, and commit.
- 5. Switch to your **main** branch and change the same first line of the same file in a different way, and commit.
- 6. Try to merge your **experimental** branch into your **main** branch; you should get a conflict. Check the status of the repository.
- 7. Edit the conflicting file and solve the conflict. Stage and commit to fix it.

Back to the past

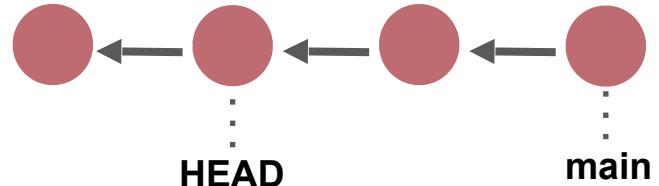
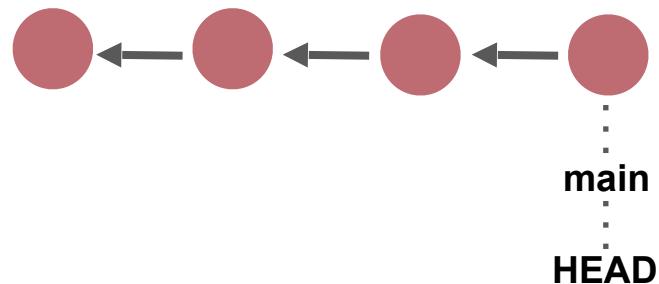
We can move around the repository using the command **checkout**

```
git checkout <identifier>
```

Identifier can be:

- A commit **ID**
- A branch **Name**

Changes the local content of the file tree to that of the reference - which means, moves **HEAD** to identifier.



Back to the past

Rewriting history is (usually) a bad idea

Every change you do in git is done with a forward in time perspective - the past should be changed only in some specific occasions!

- Changes (even to something done a long time ago) need to be done in commits added to the HEAD
- To restart from an earlier commit use branching/merging

I've botched staging or commit!

How to fix the last commit message:

```
git commit --amend -m "new message"
```



I've botched staging or commit!

`reset` is useful to “undo” some of the changes:

`git reset`

- Removes all staged changes.

`git reset <filename>`

- Removes file from staged area.

`git reset --hard <ID>`

- Moves reference to that commit (can be HEAD) and resets the working directory to match that commit.



Note that these are potentially **destructive operations**. Only on local branches.

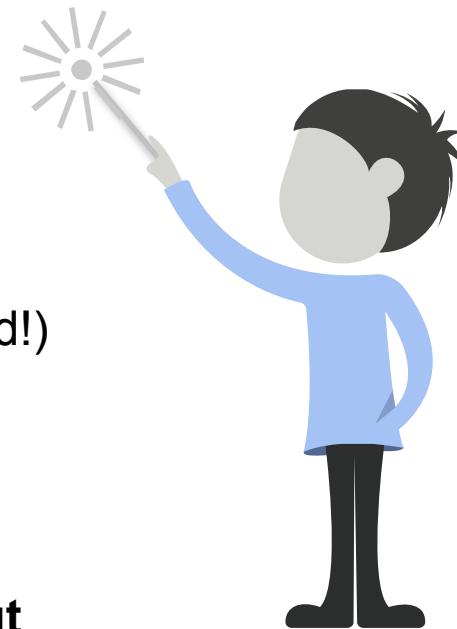
I've botched staging or commit!

`revert` is used to add commits that “reverse” the changes done by older commits:

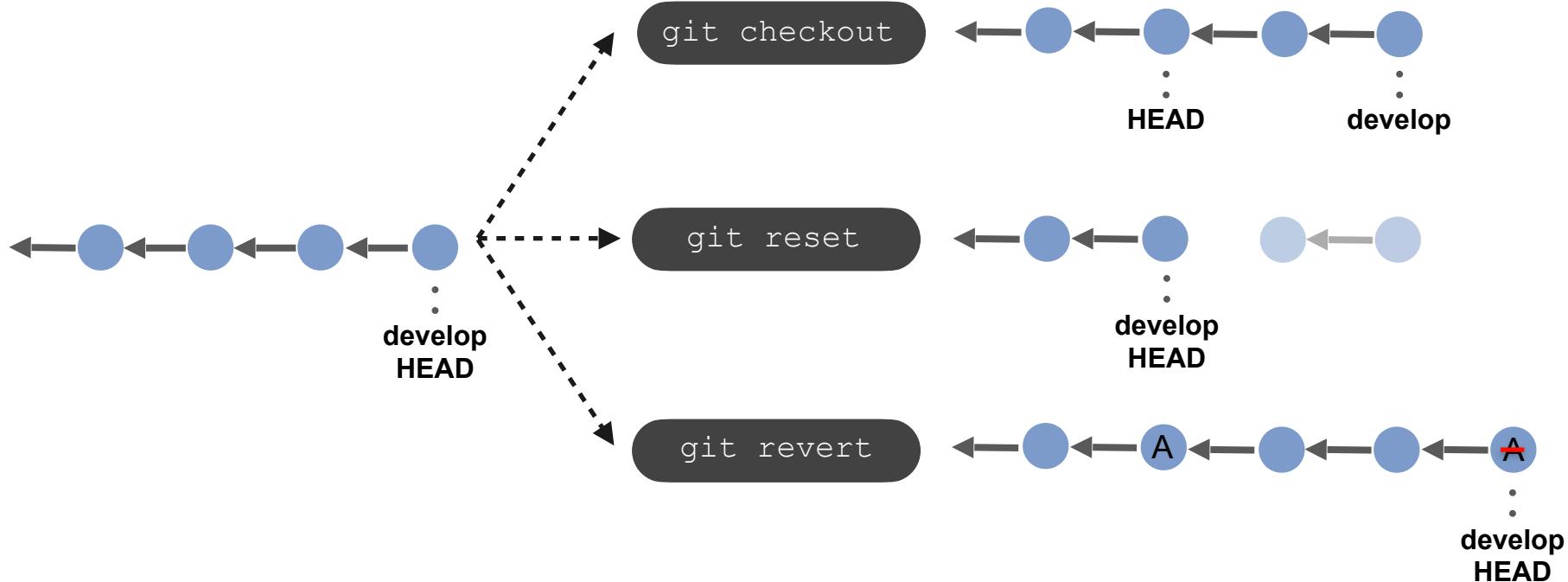
```
git revert <ID>
```

```
git revert <ID1>..<ID2>
```

- Non-destructive operation (history is maintained!)
- Here “ID” is the commit that you want to revert.
- History is polluted with commits that just revert – not the cleanest option
- **Better than reset for shared repositories, but still only on local branches.**



Git checkout, reset & revert



Exercise 4

Exercise

4

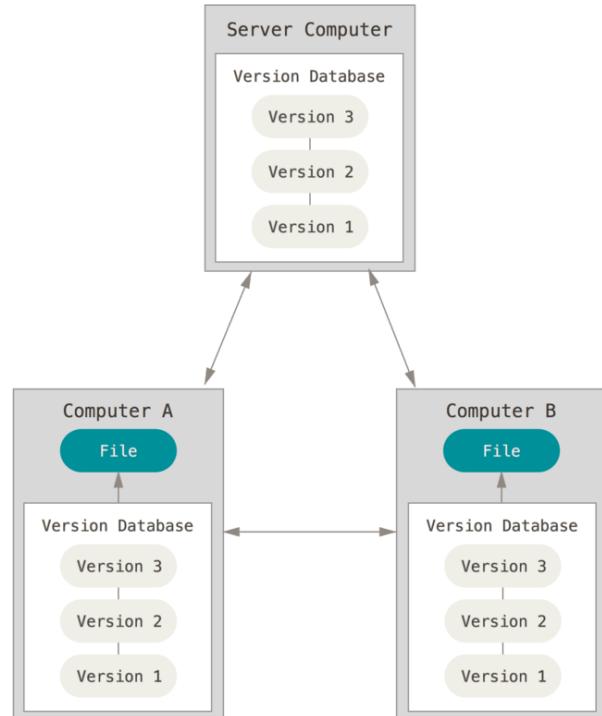
- 1. Switch to the **main** branch and change the last commit message.
- 2. Checkout an earlier commit. Check the content of the files. Check the history and status of the repository (do not commit). What do you notice? Checkout the last commit (use **main** as identifier) to get back to the **HEAD**.
- 3. Switch to the **experimental** branch. Add a step in *recipes.txt* and add the changes to stage, but **do not commit**. Check the status. Do a hard reset of **HEAD** and check the files and status. What do you notice?
- 4. Add a new ingredient to *ingredients.txt* and commit the changes. Add a new step to *recipes.txt* and commit the changes.
- 5. **Revert** the first of the two new commits (it will ask you to write a commit message). Check the files and history. What do you notice?

Interacting with a remote

A **remote** is a repository that is linked to our local (or in our case, our local repository started out as a copy of a remote repository).

We can create a local copy of our repository, make our changes as commit, and submit those to remote

We should take into account the changes that happened in the meanwhile on remote (other people's)



Personal Access Token (PAT)

Personal access token to use with the command line.

Token is used to “communicate” between local repo and remote GitHub repo.



Advantage of PAT over password:

- **Unique:** Can be generated per use or per device.
- **Revokable:** Can revoke access to each one at any time.
- **Secure:** Random strings of characters that cannot be attacked by brute force i.e. less hackable than a password.
- **Quantity:** Any number of access tokens can be created, compared to only one password per user.

Interacting with a remote

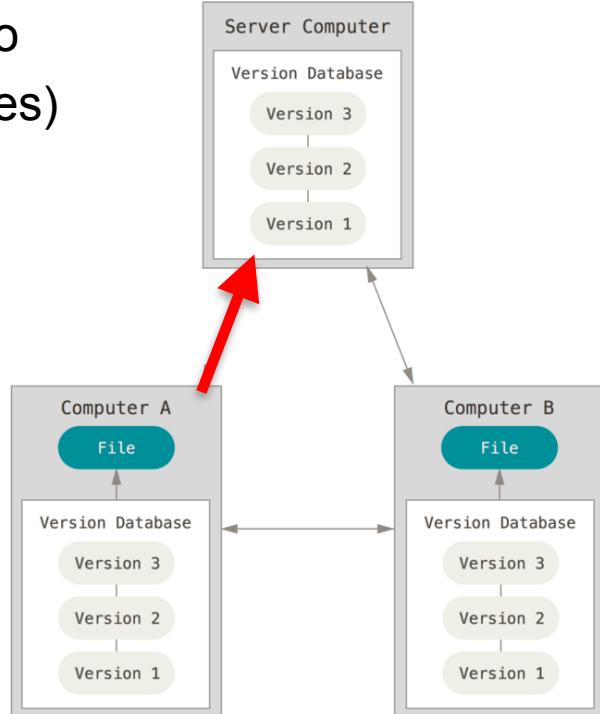
- Use **push** to upload local repository content to remote (i.e. export commits to remote branches)

```
git push <remote> <branch>
```

If this is your first time pushing to GitHub you need to login with your username and PAT.

- The default remote branch for the current local branch can be set when pushing (set upstream):

```
git push -u <remote> <branch>
```



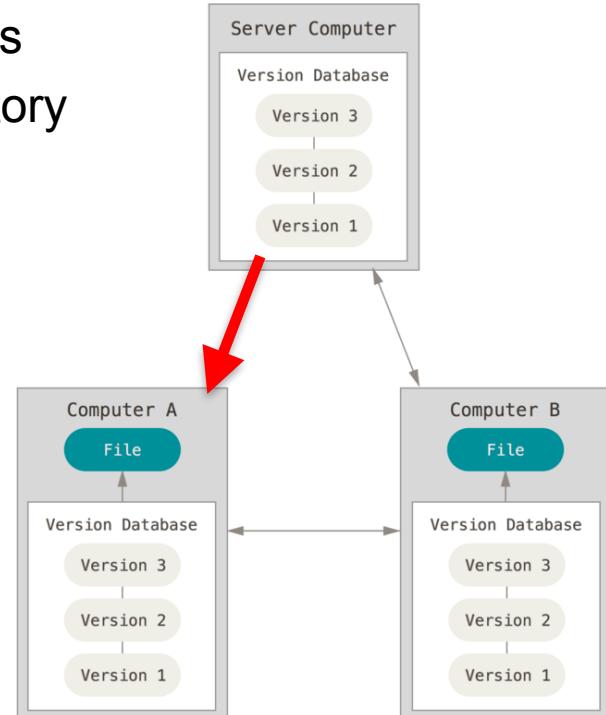
Interacting with a remote

- Use **pull** to download and apply the changes that were done in remote to your local repository (i.e. apply commits by other people):

```
git pull <remote> <branch>
```

- Use **fetch** to download without merging (remote fetched branches are named as “`<remote>/<branch>`”):

```
git fetch <remote>
```



Exercise 5

Exercise

5

- 1. Go to your GitHub account and set up your personal access token (if you have not already done it). Follow the step-by-step below:

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>
- 2. Copy your access token and note your GitHub username as you will need both the first time you are pushing data to GitHub and for RStudio part. If you have pushed to GitHub before you might not need to use it.

Exercise 5 cont.

Exercise

5

- 3. Push your changes to the remote repository for your **main** branch. Visit your remote repository on github.com. Check if your changes are there.
- 4. Push the **experimental** branch to your repository. Check the changes on GitHub. What do you notice?
- 5. Merge the **experimental** branch into **main**. Push your changes and check them on GitHub. What have changed?

Exercise 5

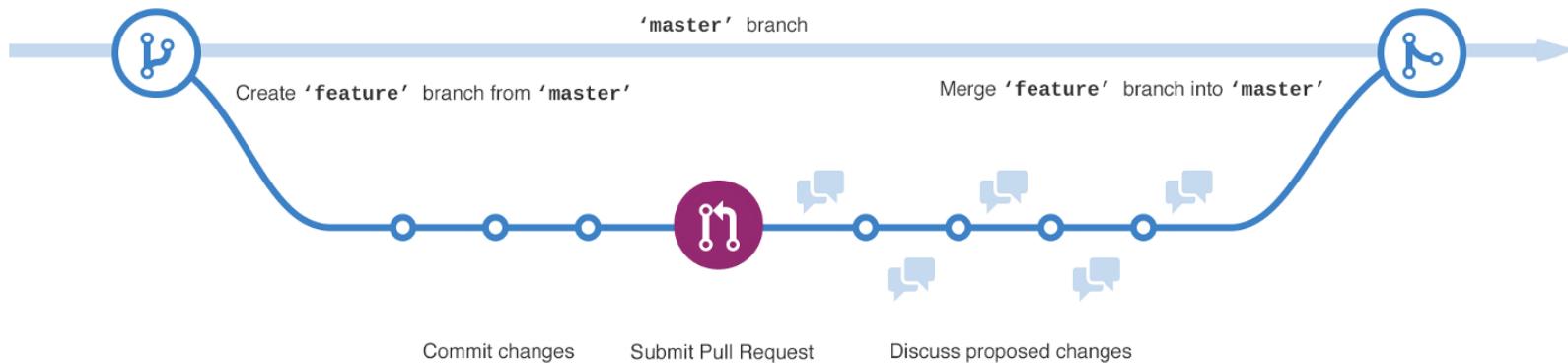
Exercise 5 cont.

- 6. Edit the **README file on GitHub**. Create a new commit on **main** and try to push it. What happens?
- 7. **Fetch** the changes from the remote repository. Note that the remote version of the **main** branch is named **origin/main**. Look at the difference between the local and remote version of **main**.
- 8. Merge remote version **origin/main** into the local version of **main** and push the changes to the remote repository on GitHub. Visit GitHub and check if your changes are there.

A deeper look at GitHub

Contributing and interacting with other users

- Projects on GitHub often follow a pattern of development and communication.
- The main branch is considered the “stable” branch and is not changeable directly - only through merges and pull requests.
- Contributions happen in branches that are merged into the main.
- Releases happens once in a while from the main branch.



A deeper look at GitHub

Interacting with other users: README



Read the documentation of the project you want to contribute to.

README.md



git-GitHub-workshop-2023

This repository is used for the HeaDS git and GitHub workshop 2023. Workshop participants should clone the repo after which they can use it to complete workshop exercises associated. The shared repo is made for hosting cooking recipes which are made locally by participants and then pushed to remote for everyone to view.

HeaDS workshop contacts:

A deeper look at GitHub

Interacting with other users: Issues



Check *Issues* if you have a problem or question to see if others have already asked the same. If needed, open a new issue.

The screenshot shows the GitHub Issues page for the repository ELELAB/mutateX. The page has 11 issues listed:

- #98: Wild type residues types should be checked when using a position list (enhancement)
- #95: Mutation runs are getting terminated before completing
- #93: Add individual data points in histogram and box plots
- #92: Unrecognised residues from FoldX (enhancement)

At the top of the page, there is a modal window titled "Label issues and pull requests for new contributors" with a "Go to Labels" button. The page also features standard GitHub navigation elements like Code, Issues (11), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

A deeper look at GitHub

Interacting with other users: Branches



Create a local copy of the repository.

- **Clone** if you have write access to the repository (direct access as *collaborator* or part of the repo *organization*). Or if you are only planning on using the method and not contributing to its development.
- **Fork** if you have no affiliation to the repository or you want your own copy to experiment with.



Create a **new branch** from the main branch. Add changes as commits. Push them so they are present on the remote (GitHub).

A deeper look at GitHub

Interacting with other users: Pull requests



Create a *Pull Request (PR)* from the new branch.

- Your code might be subject to code review. If so, the maintainers will decide if/when your code is good enough to be merged.
- Push new changes to the branch to update the PR
- Create links to issues that are related to the fix in the PR. You can use keywords (e.g. *resolves*) to automatically close an issue if the PR is merged into main: *This resolves issue #40.*

A deeper look at GitHub

Interacting with other users: Pull requests

The screenshot shows a GitHub pull request page for a repository. The top navigation bar includes links for Issues (2), Pull requests (selected), Actions, Projects, Security, and Insights. A green "New issue" button is located in the top right corner.

The main title of the pull request is "Update Catch2 to newer version #41". A purple "Merged" button indicates the status, followed by the merged user "jonassibbesen" and the commit message "merged 1 commit into master from update-deps". The merge occurred 3 weeks ago.

Below the title, there are tabs for Conversation (0), Commits (1), Checks (0), and Files changed (1). To the right, a review summary shows +1 and -1 reviews.

The commit history is as follows:

- jonassibbesen commented 3 weeks ago:

Updated the Catch2 submodule to a more recent version. Please run `git submodule update --init --recursive` to update the submodules before compiling. This resolves #40
- o- Update Catch2 to newer version 920efb0
- o- jonassibbesen merged commit d0478d0 into master 3 weeks ago
- o- jonassibbesen deleted the update-deps branch 3 weeks ago

On the right side, there are sections for Reviewers (No reviews), Assignees (No one assigned), Labels (None yet), and Projects (None yet).

Exercise 6

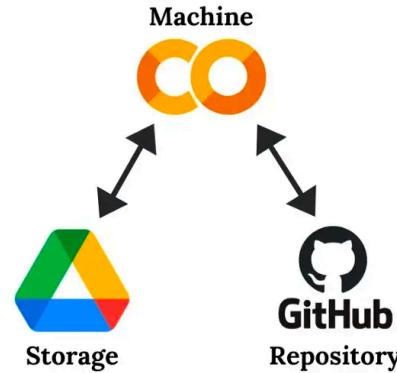
Exercise

6

- 1. Go to the web page of the shared-recipes repository for which you have become a collaborator.
- 2. Open an issue about a recipe you think is missing.
- 3. **Clone** your local copy of this shared repository. Create a new branch. Add your recipe. Commit and push your changes. Try to pull **main** and merge if needed before pushing.
- 4. Go back to the repository webpage and open a **Pull request (PR)** with your new branch. Link to your *issue* in the PR message. You might need to re-merge **main** because of changes made by other people.
- 5. Wait for one of us to approve the PR. Pull from the repo and check our shared recipe book.

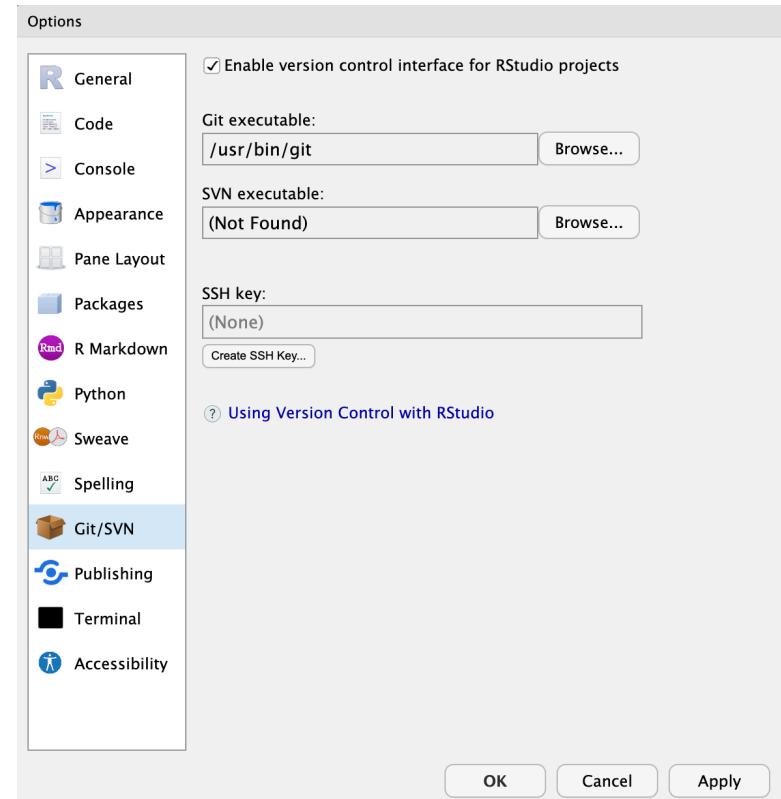
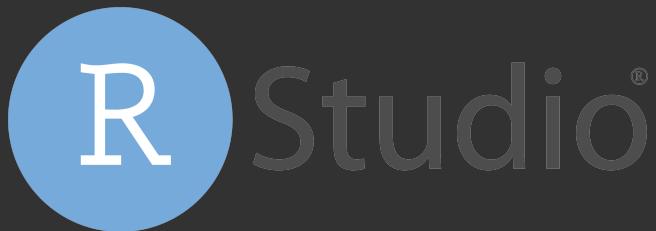
Git & GitHub with Code Editors

- You can work with **git/GitHub** from within code-editors, including Rstudio (R), PyCharm, Visual Studio etc. (Python).

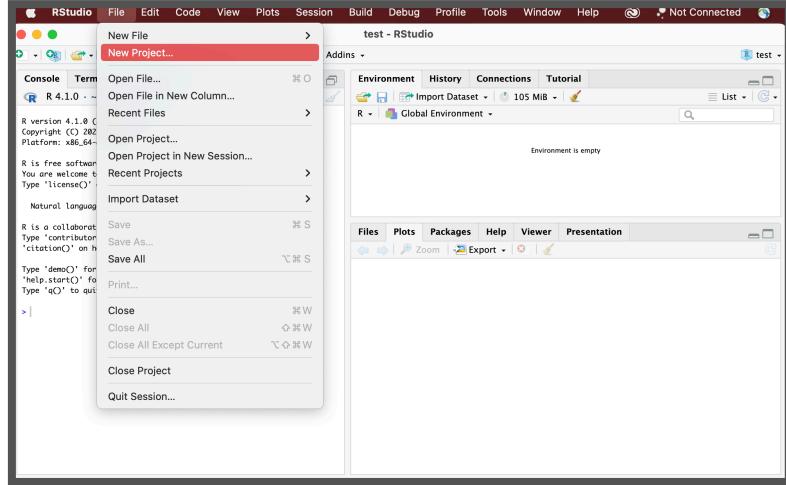


Configure RStudio

- Make Sure RStudio Knows about Git
- Go to the Git/SVN section. Then enable the version control interface and make sure RStudio knows where to find git.



Git & RStudio

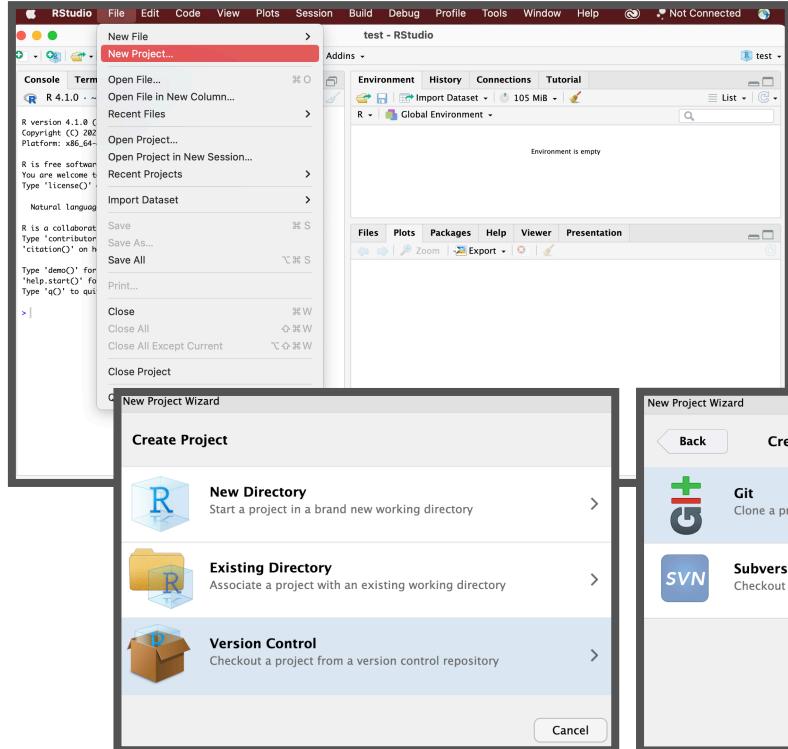


Researchers working with R benefit from using version control.

Notice! – where the Project will be saved locally.

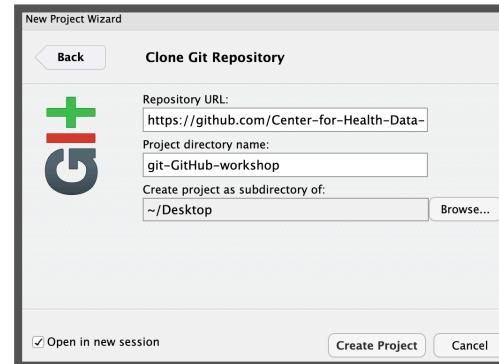
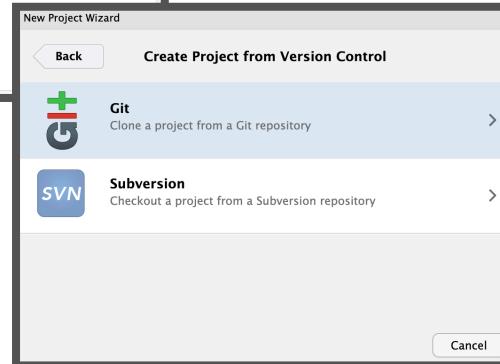


Git & RStudio

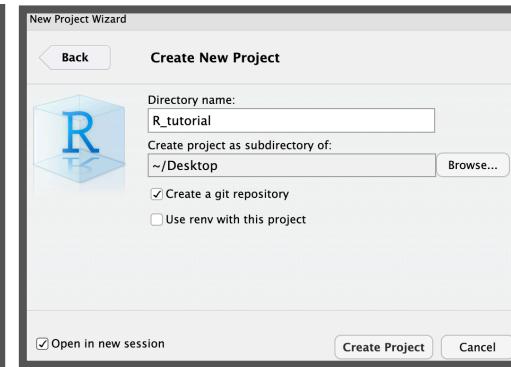
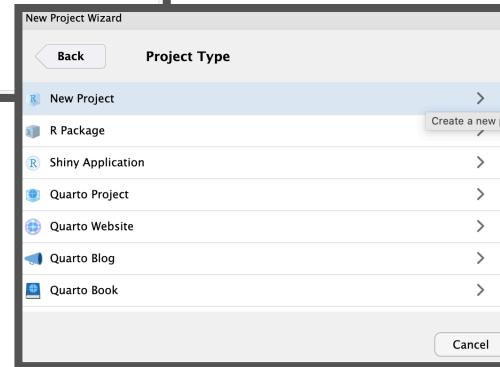
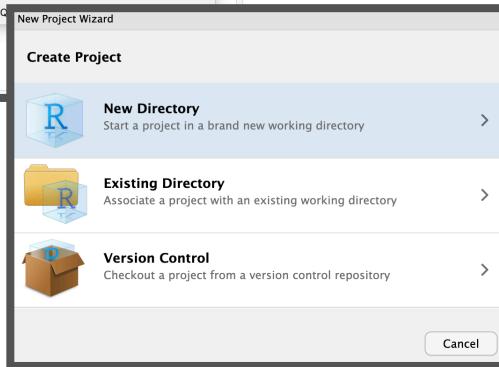
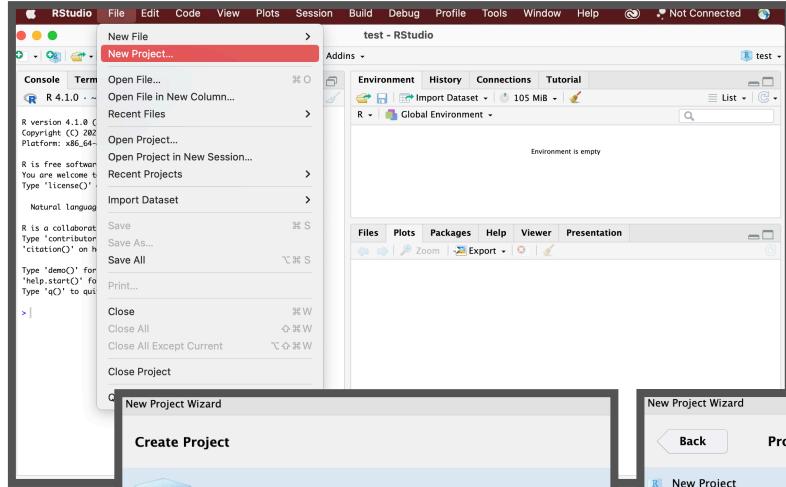


1 - New R project from the GitHub repository

- Copy the GitHub repository HTTPS url
- In Rstudio, File > New Project..., select Version Control, choose Git
- Provide the repository HTTPS link you just copied



Git & RStudio



2 - Local R project with Git

- Then, you can use the `use_github()` function, which will create a GitHub repo and connect it to your current RStudio project.

Configure Git

- You want git to know who you are so it can associate your changes with you. If you haven't configured Git already you can do it from within Studio.
- Via the usethis package in R:

```
>install.packages("usethis")
>library(usethis)
>usethis::use_git_config(user.name="Jane Doe", user.email="jane@example.org")
```

- `edit_git_config()` function will open your `gitconfig` file. Add your name and email and close this.

```
>edit_git_config()
```

Personal Access Token & RStudio

- **How to Connect RStudio Projects with Git & GitHub Repositories?**

Your Personal Access Token will need to be stored in a way RStudio can access it to connect to your GitHub account.

- There are a couple ways to get your PAT into the Git credential store:
 - Call an R function to explicitly store (or update) your credentials.
 - Do something in command line Git or RStudio that triggers a credential challenge.

Personal Access Token & RStudio

- **"gitcreds" package**

If you've installed "usethis", you will already have gitcreds installed

- Call `gitcreds::gitcreds_set()`.

If you don't have a PAT stored already, it will prompt you to enter your PAT. Paste!

If you already have a stored credential, it will reveal this and will let you inspect it.

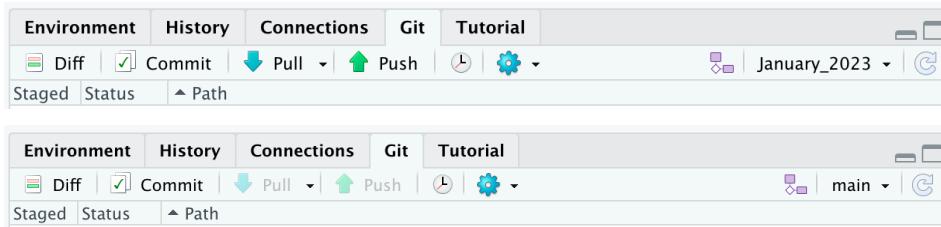
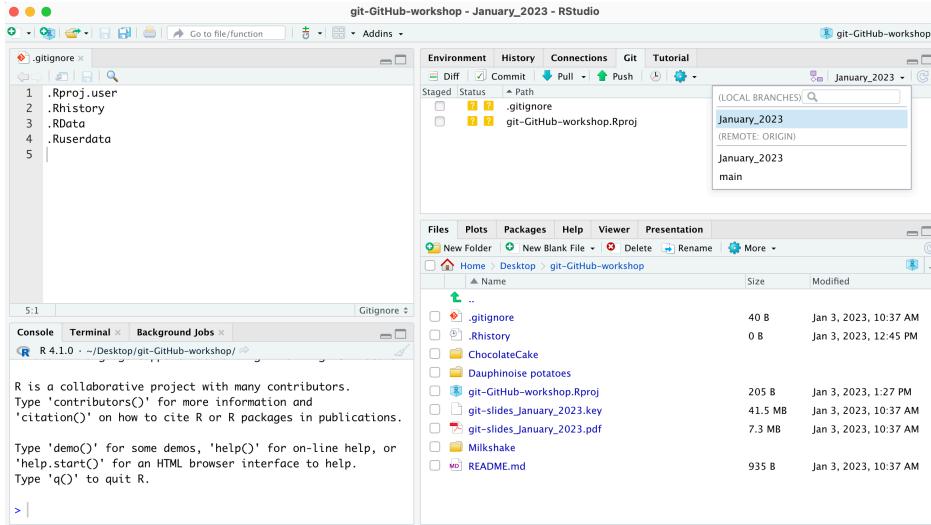
- You'll enter your GitHub username and the Personal Access Token as your password (NOT your GitHub password).

```
>install.packages("gitcreds")
>library(gitcreds)
>gitcreds_get()
#You can check that you've stored a
#credential with gitcreds_get():
```

```
>gitcreds::gitcreds_set()

? Enter password or token:
ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx
-> Adding new credentials...
-> Removing credentials from
cache...
-> Done.
```

Git & RStudio

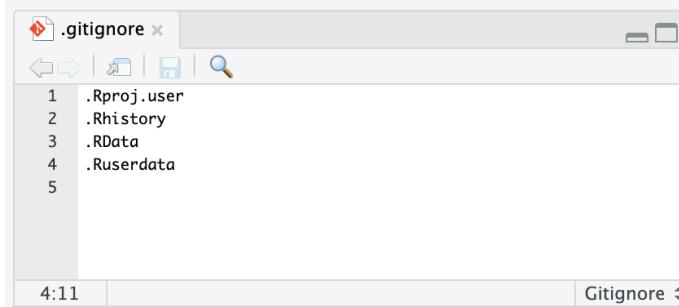


RStudio's Git pane exposes a specific subset of commands from Git.

You can use more complex commands from Shell or terminal

- Make local changes, stage and commit
- Create and switch between branches
- Changes to scripts can be reverted, and pushed to remote (e.g. GitHub)
- Click the green “Push” button to send your local changes to GitHub.

Ignore files from version control



A screenshot of the RStudio interface showing the contents of a .gitignore file. The window title is ".gitignore". The file contains the following text:

```
1 .Rproj.user
2 .Rhistory
3 .RData
4 .Ruserdata
5
```

The status bar at the bottom shows "4:11" and "Gitignore".

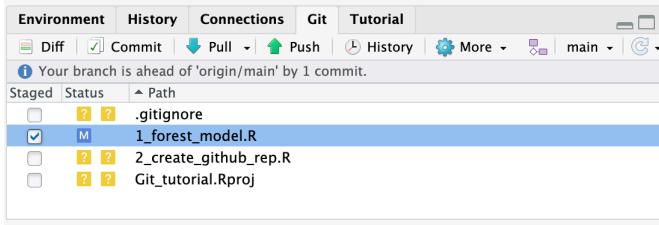
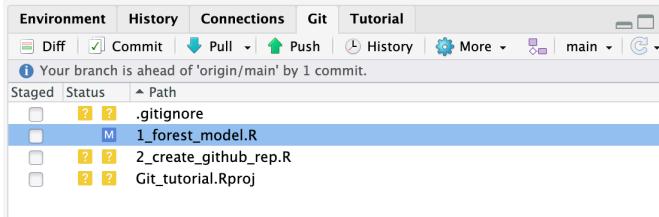
When starting a new project in RStudio, it will always add a file .gitignore if it does not already exists and add some initial files to ignore.

Adding file names or folder names to .gitignore will ignore files that should not be taken into account by Git

Some other examples of files you probably want to ignore:

- Sensitive information (passwords,...)!
- Binary files. Git works very well with text files, but not with binary files such as .Rdata.
- Files > 50MB. Git is specifically made for code (e.g. .R) and does not intend to track all changes in large data files.
- A temp/ folder inside your project with 'disposable' content, e.g. draft, test or temp.

Stage and Commit

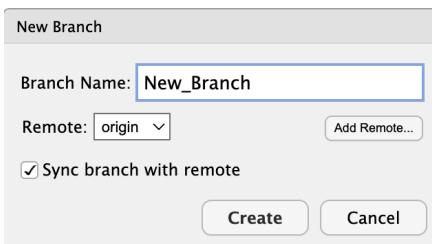
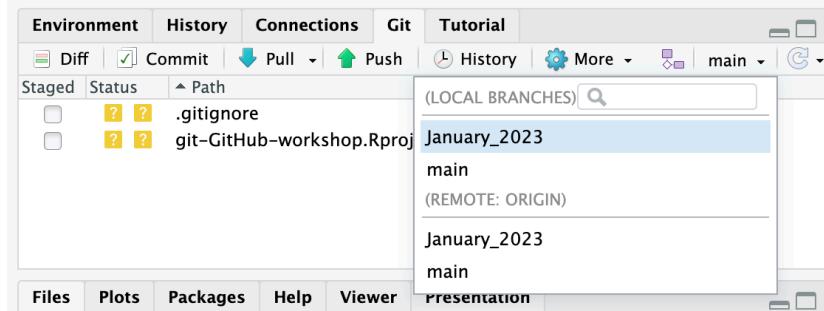


The status of the new files/changes is indicated with the yellow question mark defining that the file is yet unknown to Git.

A screenshot of the RStudio interface showing the "Review Changes" dialog. The "Changes" tab is selected. A message says "Your branch is ahead of 'origin/main' by 1 commit." The "Staged" section lists ".gitignore" (yellow question mark), "1_forest_model.R" (blue checked box with a yellow checkmark icon), "2_create_github_rep.R" (yellow question mark), and "Git_tutorial.Rproj" (yellow question mark). The "Commit message" field contains "Remove boxplot". Below it is a checkbox for "Amend previous commit" and a "Commit" button. The code editor below shows R code for a forest model and ggboxplot.

By checking the box and clicking commit, we add the file and are able to commit this with a commit message.
THEN FINALLY WE PUSH TO GITHUB!

Branching



Subject	Author	Date (UTC)	SHA
HEAD => refs/heads/main	origin/main add fetcjonassibesens <j.a.si	2022-02-24	c20693cb
Merge branch 'main' of https://github.com/ Thilde Bagger Terkels	2022-02-24	e3c44996	
introslides edited	Thilde Bagger Terkels	2022-02-24	4a01a785
Update README.md	Thilde Bagger Terkels	2022-02-24	616a7b3e
Update README.md	Thilde Bagger Terkels	2022-02-24	abfa6aa9
updated personal repos	Thilde Bagger Terkels	2022-02-24	edfbec37
readme updated with links to personal repo	Thilde Bagger Terkels	2022-02-24	180f26f1
slidehows added, pdf and .key	Thilde Bagger Terkels	2022-02-24	87cdfe8a
Merge pull request #1 from Center-for-Hea	Diana Andrejeva <70	2022-02-21	bf96b33a
add milkshake recipe to drinks branch	andrejeva_d <andrejk	2022-02-21	f472f13a
Revert "add sugar to Dauphinoise potatoes i	andrejeva_d <andrejt	2022-02-21	286754a6
add sugar to Dauphinoise potatoes ingredie	andrejeva_d <andrejk	2022-02-21	3d4bb625
add salt to the Dauphinoise potatoes ingred	andrejeva_d <andrejk	2022-02-21	7bda8972
Dianas recipe	andrejeva_d <andrejk	2022-02-21	68e57c53
more vanilla needed	Thilde Bagger Terkels	2022-02-07	6a4bdb5d
Chocolate cake recipe added	Thilde Bagger Terkels	2022-02-07	15564f9d
Chocolate cake ingredients added	Thilde Bagger Terkels	2022-02-07	cba36d7e
Update README.md	Thilde Bagger Terkels	2022-02-07	023313f7
Initial commit	Thilde Bagger Terkels	2022-02-07	e7579038

RStudio: Review Changes

Changes	History	main	(all commits)	Search	Pull
Subject	Author	Date (UTC)	SHA		
HEAD -> refs/heads/main	origin/main	add fetc jonassibbesen <j.a.si	2022-02-24	c20693cb	
Merge branch 'main' of https://github.com/ Thilde Bagger Terkels	Thilde Bagger Terkels	2022-02-24	e3c44996		
introslides edited	Thilde Bagger Terkels	2022-02-24	4a01a785		
Update README.md	Thilde Bagger Terkels	2022-02-24	616a7b3e		
Update README.md	Thilde Bagger Terkels	2022-02-24	abfa6aa9		
updated personal repos	Thilde Bagger Terkels	2022-02-24	edbfece37		
readme updated with links to personal repo	Thilde Bagger Terkels	2022-02-24	180f26f1		
slidehows added, pdf and .key	Thilde Bagger Terkels	2022-02-24	87cdef8a		
Merge pull request #1 from Center-for-Hea	Diana Andrejeva <70	2022-02-21	bf96b33a		
add milkshake recipe to drinks branch	andrejeva_d <andrejek	2022-02-21	f472f13a		
Revert "add sugar to Dauphinoise potatoes i	andrejeva_d <andrejek	2022-02-21	286754a6		
add sugar to Dauphinoise potatoes ingredie	andrejeva_d <andrejek	2022-02-21	3d4bb625		
add salt to the Dauphinoise potatoes ingred	andrejeva_d <andrejek	2022-02-21	7bda8972		
Dianas recipe	andrejeva_d <andrejek	2022-02-21	68e57c53		
more vanilla needed	Thilde Bagger Terkels	2022-02-07	6a4bdb5d		

README.md

README.md

View file @ abfa6aa9

```
@@ -1,4 +1,4 @@
1 # git-GitHub-workshop-2022
1 ## git-GitHub-workshop-2022
2 2
3 3 This repository is used for the HeaDS git and GitHub workshop 2022.
4 4 Workshop participants should clone the repo after which they can use it to complete workshop
   exercises associated. The shared repo is made for hosting cooking recipies which are made
   locally by participants and then pushed to remote for everyone to view.
@@ -14,7 +14,7 @@ josi@di.ku.dk
14 Diana Andrejev, Data Scientist
15 andrejeva@sund.ku.dk
16
17 For the workshops last exercise we will be working with three private repositories to learn to
   work with requests:
17 # For the workshops last exercise we will be working with three private repositories to learn to
   work with requests:
18 18
```

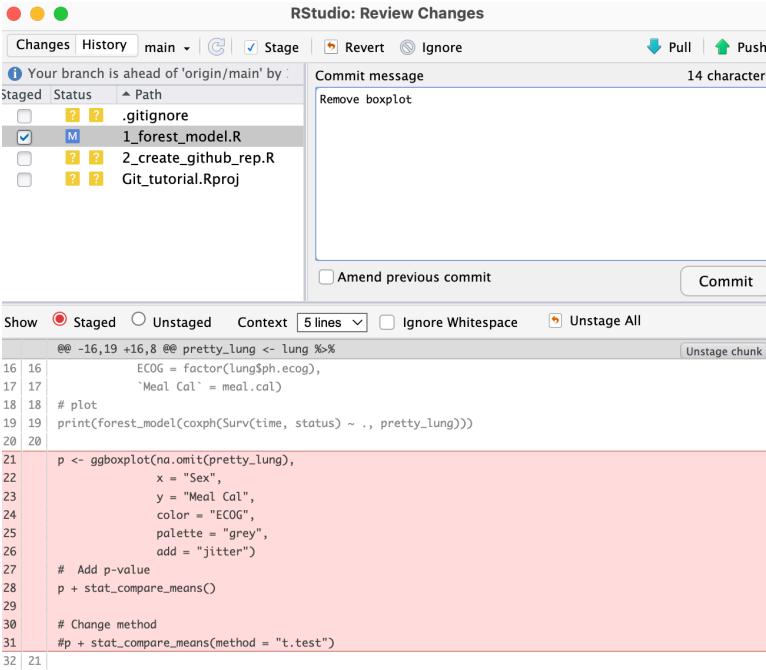
History

You can check the history (the series of commits) online on GitHub, terminal as well as in Rstudio.

In Rstudio, click on history in the Git pane:

Notice the presence of History twice in the previous image: Git history vs R command history

Interact with GitHub



RStudio: Review Changes

Changes History main Stage Revert Ignore Pull Push

Your branch is ahead of 'origin/main' by :

Staged	Status	Path
<input type="checkbox"/>	<input type="checkbox"/> ?	.gitignore
<input checked="" type="checkbox"/>	<input type="checkbox"/> M	1_forest_model.R
<input type="checkbox"/>	<input type="checkbox"/> ?	2_create_github_repos.R
<input type="checkbox"/>	<input type="checkbox"/> ?	Git_tutorial.Rproj

Commit message
Remove boxplot 14 characters

Show Staged Unstaged Context 5 lines Ignore Whitespace Unstage All

```
@@ -16,19 +16,8 @@ pretty_lung <- lung %>%  
16 16 ECOG = factor(lung$ph.ecog),  
17 17 `Meal Cal` = meal.cal)  
18 18 # plot  
19 19 print(forest_model(coxph(Surv(time, status) ~ ., pretty_lung)))  
20  
21 p <- ggboxplot(na.omit(pretty_lung),  
22 x = "Sex",  
23 y = "Meal Cal",  
24 color = "ECOG",  
25 palette = "grey",  
26 add = "jitter")  
27 # Add p-value  
28 p + stat_compare_means()  
29  
30 # Change method  
31 #p + stat_compare_means(method = "t.test")
```

Pull updates from GitHub

- Using the down arrow button, RStudio goes to the GitHub repo, grabs the most recent code and brings it into your local editor.
- Pulling regularly is extremely important if you're collaborating.

Push changes to GitHub

- Rstudio gives you a warning about the status of your local commits versus those stored on GitHub.
- After committing, we now have a push button (the up arrow) on RStudio.



Thank you for today
See you at HeaDS

IN CASE OF FIRE 🔥

git commit

git push

git -tf out