GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

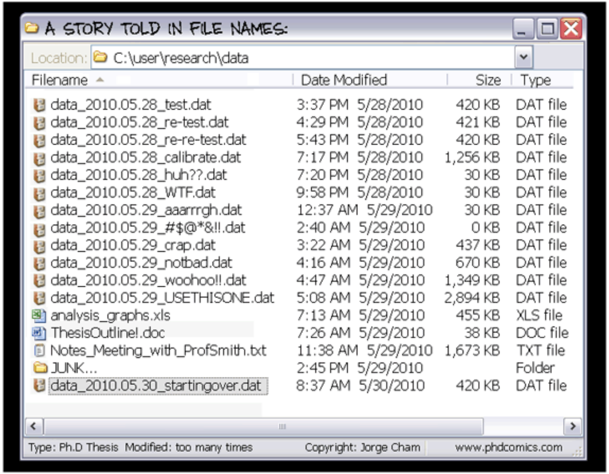
Data Science – Basics

Lecture 07 – Versioning

Fabian Sinz

27. May 2024

Institute for Computer Science – Campus Institute for Data Science (CIDAS)



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

[https://phdcomics.com/comics/archive_print.php?comicid=1323]

Reproducibility

Today and next time we are going to talk about **reproducibility** and tools that can help you achieve it.

1. Versioning (Git and GitHub)
2. Reproducible Python environments (`virtualenv`)
3. Containerization (Docker).

Why is reproducibility important?

- Professionalism: Your work should be reliable and verifiable (by others and you).
- Have piece of mind and reduce mental load.
- Debugging (it worked three days ago … what changed?)
- Share your work.

Versioning

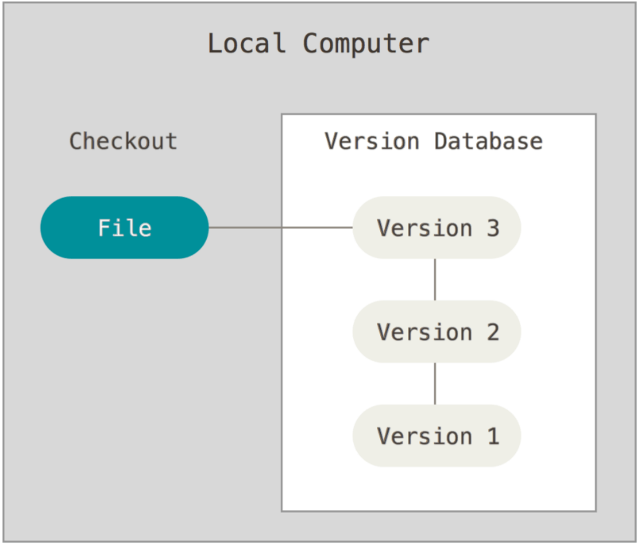
What is git?

- Git (British for “Idiot”) is a **distributed version control system**
- It is command line based (Shell)
- There are other similar systems (mercurial, bazaar) but git is currently the most widely used
- It allows you to
  - take snapshots of your project and save them.
  - go back to any snapshot.
  - work on different versions of the same project (branches).
  - merge different versions into a consistent one.
- It was written by Linus Torvalds from April to Juli 2005.
- It free and open source.

Linus Torvalds
Creator of Linux and Git

Local version control

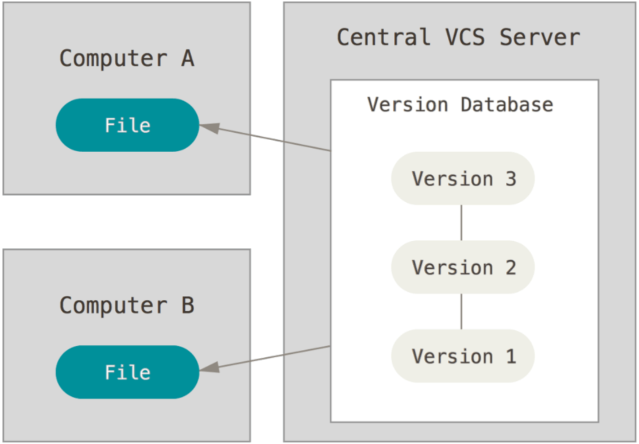
- A local version control system keeps the snapshots on the same computer.
- Access to the history is local.
- No central backup.



[https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control]

Centralized version control

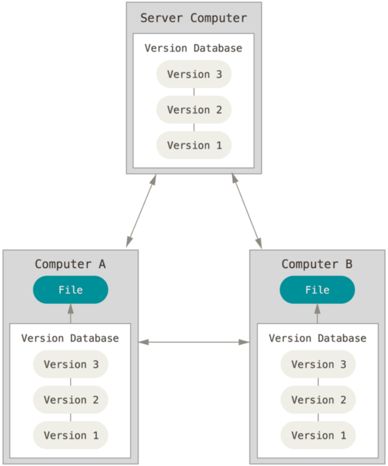
- One central server keeps the snapshots (e.g. CVS did that)
- To collaborate you always need access to the server
- Server admin has fine control of access rights



[https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control]

Distributed version control

- Git is distributed
- Each computer stores the full history.
- It's robust and redundant.
- Allows complex collaborations and workflows between different groups.

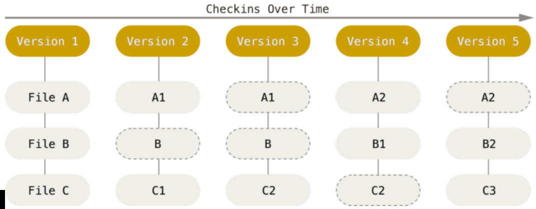


[https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control]

Git Basics

How does git track changes

- Git stores a series of snapshots of the tracked files.
- If a file has not changed, it just stores a reference the previous version.
- Git computes a checksum of the local files using a SHA-1 hash looking like `24b9da6552252987aa493b52f8696cd6d3b00373`
- Different snapshots (commits) are referred by their hash

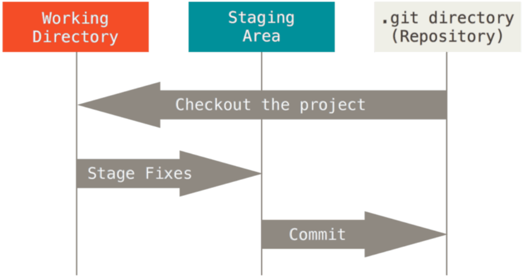


[https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F]

Three (four) stages of a file

A file can be

- untracked: Git ignores it
- modified: The file is changed but not in the local database
- staged: The current version of the file is marked to be stored in the local database
- committed: The file is safely stored in the local database



[https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F]

Getting a git repository

1. A repository can be **initialized** from an empty directory.
2. A repository can be **cloned** from another (local or remote) location.

How to initialize a project from an existing directory and add first files

```
cd my_project
git init
git add *.py
git add LICENSE
git commit -m 'Initial project version'
```

[https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository]

Getting a git repository

1. A repository can be **initialized** from an empty directory.
2. A repository can be **cloned** from another (local or remote) location.

How to clone a project from an existing repository

```
git clone https://github.com/sinzlab/datascience_git_demo.git
```

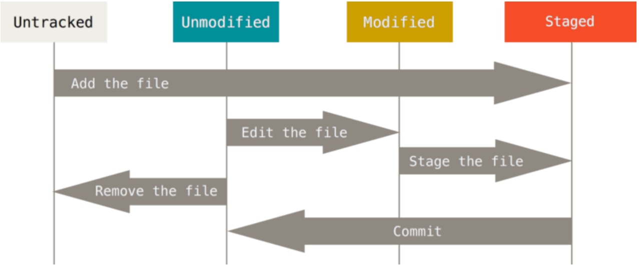
[https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository]

Life cycle of a file

```
touch solution.py # do something with files

git status # check status

git add solution.py # stage file
git commit -m "add solution" # commit
```



[https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository]

Git history and branches

Removing files

- `rm solution.py` removes the file from the working directory. It's modified but not staged.
- `git rm solution.py` removes and stages the file deletion
- `git rm --cached solution.py` stages the removal of the file from git's tracking but keeps the local copy

Git history

`git log` show the commit history and it’s hashes in reverse chronological order.

The output will look something like this

```
commit ba6642b0991038893ae07c52d0eb23c52e7f69ef (HEAD -> main)
Author: Fabian Sinz <sinz@cs.uni-goettingen.de>
Date:   Fri Aug 12 13:05:38 2022 +0200

  added solution

commit ba2cb8fc525d52753f91b6cd55fbb9a408feebdb (origin/main, origin/HEAD)
Author: Fabian Sinz <fabee@epagoge.de>
Date:   Fri Aug 12 12:53:34 2022 +0200

  Initial commit
```

[https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History]

Undoing things

- `git restore <file>` will revert the modifications of a file to the latest commit.
- `git restore --staged <file>` will unstage a staged file.
- `git reset --hard  <SHA1>` will reset your working directory to that hash an forget all changes (dangerous!)

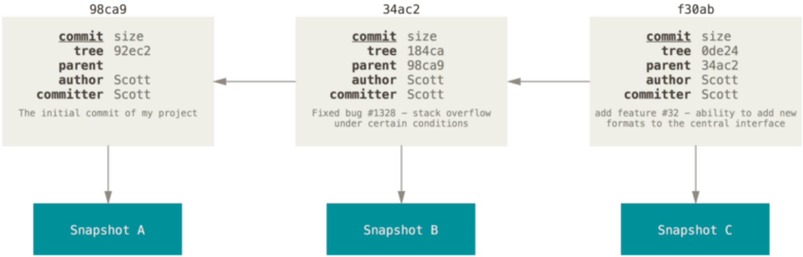
[https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things]

Looking back

- `git checkout <SHA1>` will temporarily checkout the commit with that hash.
- This will put you in a **detached HEAD** state.
- You can look at the current setup with `git log --graph --all`.
- `git checkout main` or `git checkout master` will bring you back to the latest version.

[https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things]

What is a HEAD? Commits, refs, and tags.



98ca9

```
   commit  size
     tree  92ec2
   parent
   author  Scott
committer  Scott
```

The initial commit of my project

34ac2

```
   commit  size
     tree  184ca
   parent  98ca9
   author  Scott
committer  Scott
```

Fixed bug #1328 - stack overflow under certain conditions

f30ab

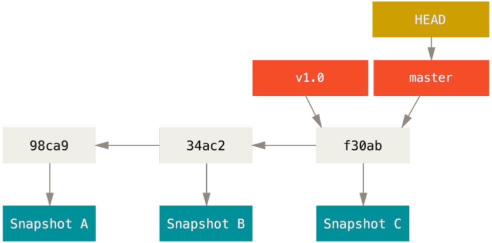
```
   commit  size
     tree  0de24
   parent  34ac2
   author  Scott
committer  Scott
```

add feature #32 - ability to add new formats to the central interface

- A git history is a series of commits (hashes) that know their parent commit.
- However, hashes are hard to remember so git can give them (moveable) names.
- These names are refered to as references.
- They can point to commits or other references.

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

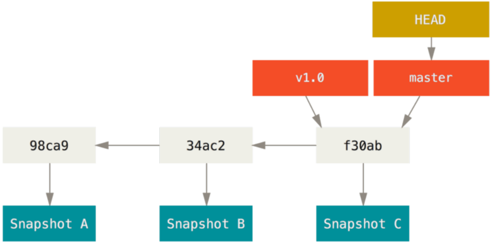
What is a HEAD? Commits, refs, and tags.



- One type of reference are branches that point to a particular commit.
- The default branch name in git is `master` or `main`.
- Every time you commit in a branch, this pointer moves ahead too.
- Branches keep track of the history tree when “branches” out.
- You can also name certain commits yourself using tags (like `v1.0`).
- You can manually inspect the references and branches by checking `.git/refs`.

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

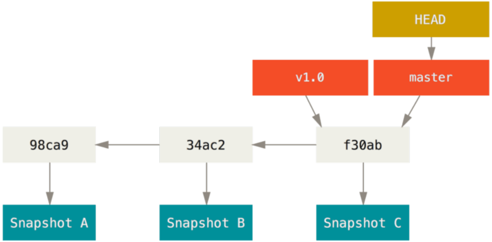
Commits, references, tags, branches



- One type of reference are branches that point to a particular commit.
- The default branch name in git is `master` or `main`.
- Every time you commit in a branch, this pointer moves ahead too.
- Branches keep track of the history tree when “branches” out.
- You can also name certain commits yourself using tags (like `v1.0`).
- You can manually inspect the references and branches by checking `.git/refs`.

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

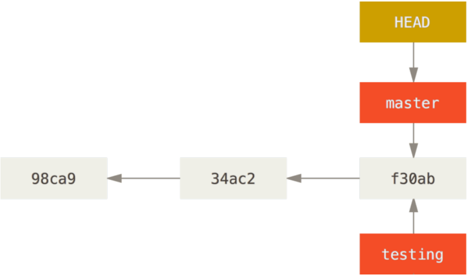
What is a HEAD? Commits, refs, and tags.



- git knows what you are currently working on trough the special reference called HEAD.
- HEAD points to the commit you are currently working on.

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

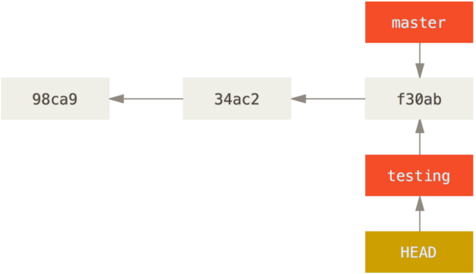
How to create a new branch?



- You can create a new branch by `git branch testing`
- `testing` is just a name, it can be anything

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

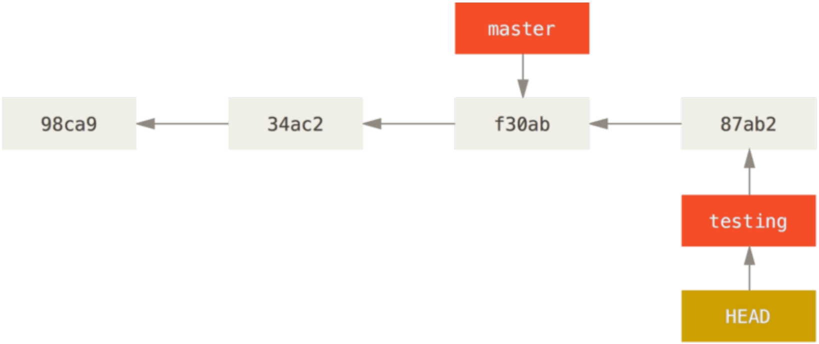
How to create a new branch?



- You can create a new branch by `git branch testing`
- `testing` is just a name, it can be anything
- This only **creates** the branch.
- If you want to contribute to it you need to move the HEAD to the new branch by `git switch testing`

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

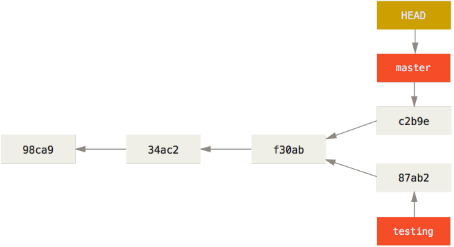
How to create a new branch?



- Now you can make new commits which will advance `testing`.
- To go back to the other branch you simply check it our `git switch master`

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

How to create a new branch?



- Now you can make new commits which will advance `testing`.
- To go back to the other branch you simply check it our `git switch master`
- You can then also make independent changes there.

[https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell]

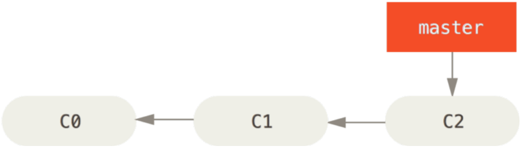
Merging branches

Illustratory example

1. In your software package you want to fix an issue.
2. You create a branch for that issue
3. Something urgent forces you to fix the master branch immediately.
4. You create a branch for that, fix it, merge it to the master.
5. Then you continue working on the issue.

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

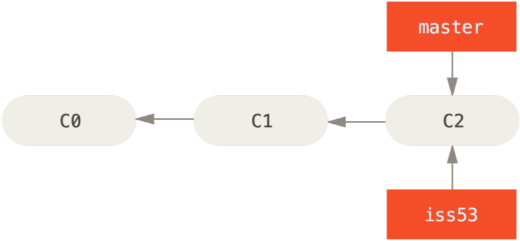
Create an issue branch



- To create a new branch for the issue and switch to it, you use `git switch -c iss53`

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Create an issue branch

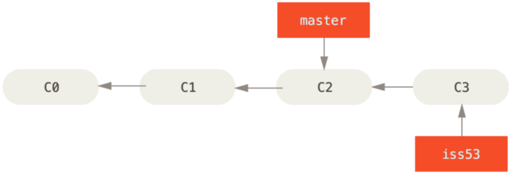


- To create a new branch for the issue and switch to it, you use `git switch -c iss53`
- This is a shortcut for

```
    git branch iss53
    git switch iss53
```

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

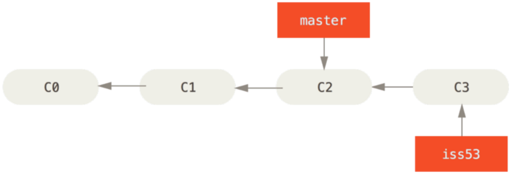
Create an issue branch



- Now you work on the issue and make commits (e.g. `git commit -a -m '<MSG>'`)

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

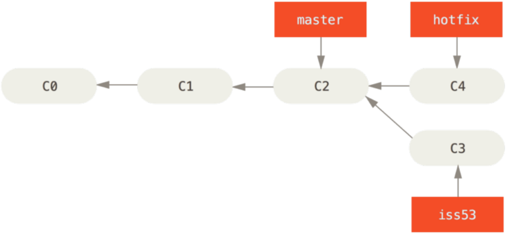
Create an issue branch



- Now you work on the issue and make commits (e.g. `git commit -a -m '<MSG>'`)
- Now you get a call that the master branch needs to be fixed.
- You commit everything on the issue branch and switch back to master with `git switch master`

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Create a hotfix branch

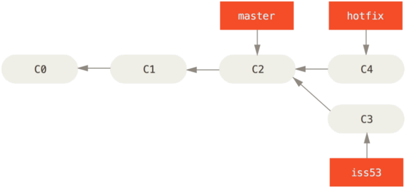


- On master, you create a new branch for the hotfix and start working on it

```
git switch -c hotfix
# ... some editing
git commit -m -a "<MSG>"
```

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Merge hotfix branch

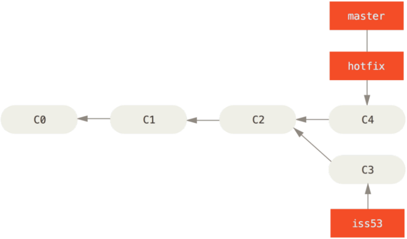


- After you have fixed the immediate problem, you go back to master and merge the `hotfix` branch into it

```
git switch master # go where you want to merge TO
git merge hotfix # say what you want to merge to your current HEAD
```

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Merge hotfix branch



- After you have fixed the immediate problem, you go back to master and merge the `hotfix` branch into it

```
git switch master # go where you want to merge TO
git merge hotfix # say what you want to merge to your current HEAD
```

- For git, merging is easy here since it only needs to move the reference `master` to `hotfix` (called fast-forward)

Merge hotfix branch

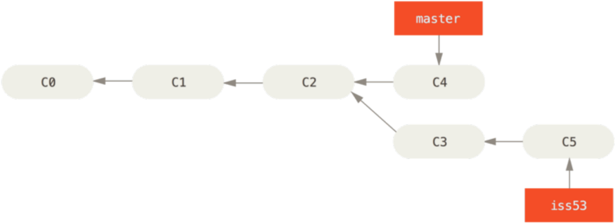
- After you have fixed the immediate problem, you go back to master and merge the `hotfix` branch into it

```
git switch master # go where you want to merge TO
git merge hotfix # say what you want to merge to your current HEAD
```

- For git, merging is easy here since it only needs to move the reference `master` to `hotfix` (called fast-forward)
- You can delete the `hotfix` branch afterwards `git branch -d hotfix`

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

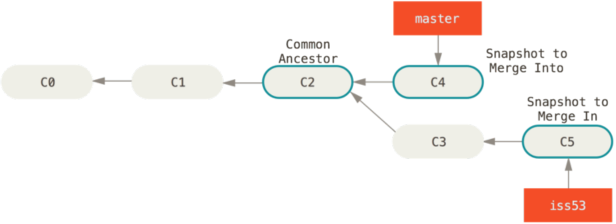
Merge issue branch



- Now you can go back to the issue via `git switch iss53` and finish your work there.

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Merge issue branch

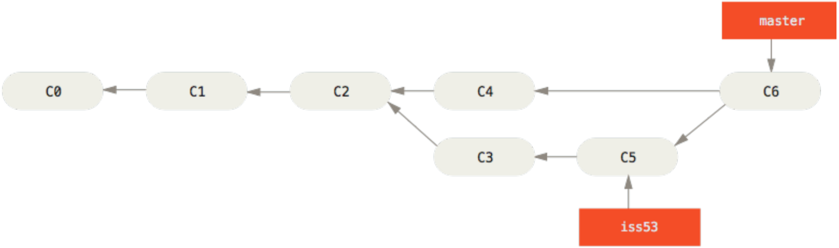


- Now you can go back to the issue via `git switch iss53` and finish your work there.
- You can do that just as before

```
git switch master # go where you want to merge TO
git merge iss53 # say what you want to merge to your current HEAD
```

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Merge issue branch



- However, now git needs to actually merge files.
- It does that by creating a merge commit.
- The merge commit is special because it has two parents.

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

What if git cannot merge automatically?

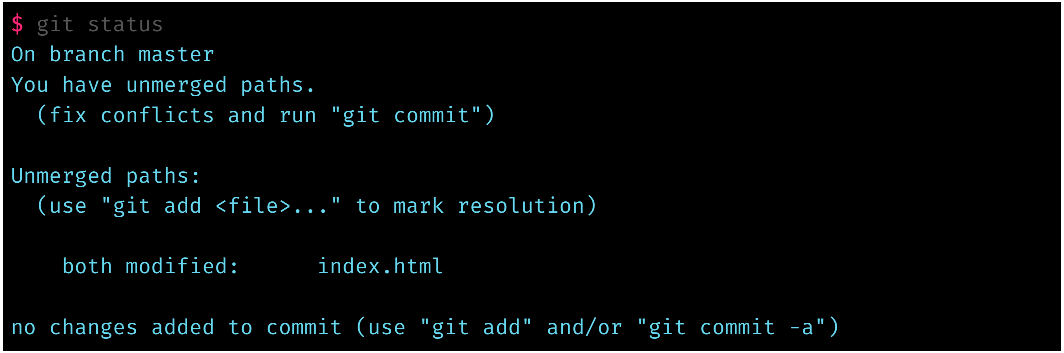
Sometimes things do not go smoothly and git tells you

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

What if git cannot merge automatically?

Git has not created a merge commit. It has halted and you need to fix things manually.



```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

What if git cannot merge automatically?

Git will point out the problems with markers like this

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

You simply go into the files, fix things manually and then `git commit` everything. This completes the merge.

[https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging]

Git remotes

Getting a git repository

Remember how we cloned our initial version from a remote repository (here: github)

```
git clone https://github.com/sinzlab/datascience_git_demo.git
```

Git also keeps a reference to what commit we got from the remote.

```
* 4144541 (HEAD -> main) add pandas
| * 6bfc8df (my_crazy_idea) add matplotlib
|/
* aad2927 (tag: v0.1) added imports
* ba6642b added solution
* ba2cb8f (origin/main, origin/HEAD) Initial commit
```

These are special branches that you cannot change.

[see also `https://git-scm.com/book/id/v2/Git-Branching-Remote-Branches`]

Remotes

- You can pull updates from the remote via `git pull <REMOTE> <BRANCH>`, e.g. `git pull origin main`.

Remotes

- You can pull updates from the remote via `git pull <REMOTE> <BRANCH>`, e.g. `git pull origin main`.
- Similarly, you can push your local changes to the remote `git push <REMOTE> <BRANCH>`, e.g. `git push origin main`.

Remotes

- You can pull updates from the remote via `git pull <REMOTE> <BRANCH>`, e.g. `git pull origin main`.
- Similarly, you can push your local changes to the remote `git push <REMOTE> <BRANCH>`, e.g. `git push origin main`.
- Sometimes, you will need to synchronize your local branches before you are allowed to do that.

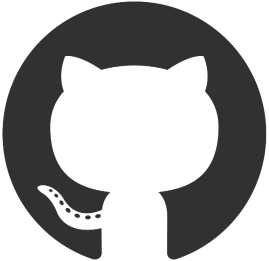
Remotes

- You can pull updates from the remote via `git pull <REMOTE> <BRANCH>`, e.g. `git pull origin main`.
- Similarly, you can push your local changes to the remote `git push <REMOTE> <BRANCH>`, e.g. `git push origin main`.
- Sometimes, you will need to synchronize your local branches before you are allowed to do that.
- You can look at all your remotes via `git remote -v`

Github

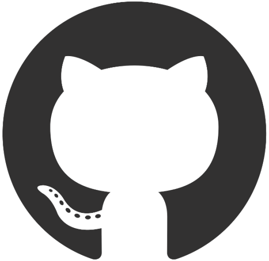
Github

- `github.com` is an online code platform that uses git
- It has free plans for students.
- It has many tools to collaborate on software projects.

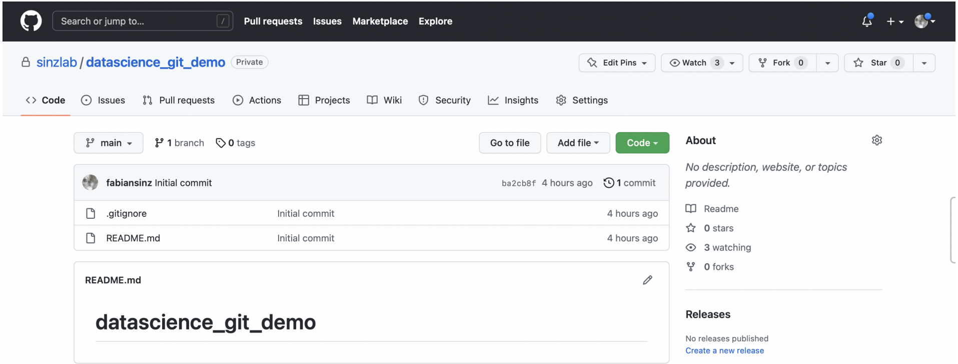


Github

- `github.com` is an online code platform that uses git
- It has free plans for students.
- It has many tools to collaborate on software projects.
- For your personal use, you can just use it as a remote.



Forking



fabiansinz Initial commit	ba2cb8f 4 hours ago	1 commit
.gitignore	Initial commit	4 hours ago
README.md	Initial commit	4 hours ago

README.md

datascience_git_demo

No description, website, or topics provided.

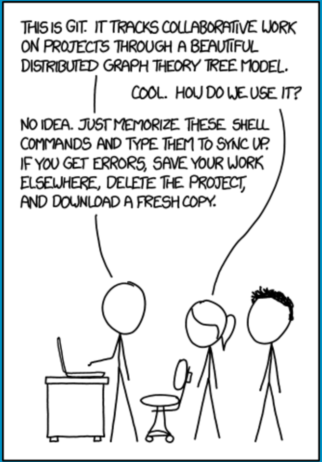
No releases published

- In a multi-person project everyone pushing and pulling to the same remote, will be confusing.
- To that end, github offers forks that let you clone the repository to your own github space.

Normal development process

1. You **fork** a project, you want to contribute to, to your local account.
2. You **clone** the fork to you local machine via `git clone ...`
3. You work on the code locally using commits to save your work
4. You **push** your local changes back to your account: `git push origin main`
5. Once you are happy with a contribute, you make a **pull request** via github. This will request to merge your changes into the version you forked from.
6. **Other persons than you** review this request and either approve it or request changes.
7. Once they are happy with it, they merge the pull request. Until it is merged, you can still add to it with more pushes to you local version.

- Git is a distributed code versioning system
- It is very versatile and powerful.
- Start using it by just using basic functionality.
- This saves your code.
- If you are in trouble you can resolve your problems with googeling or `https://ohshitgit.com`
- Start using github and do use mutual code review


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

[https://xkcd.com/1597/]

Thanks for listening.

Questions?