Version control management and research collaboration using git and github

An introduction

APSIS group

MCC Berlin

July 11th, 2019

What are git and GitHub and why should I use them?

How can I use git and GitHub?

What is version control?

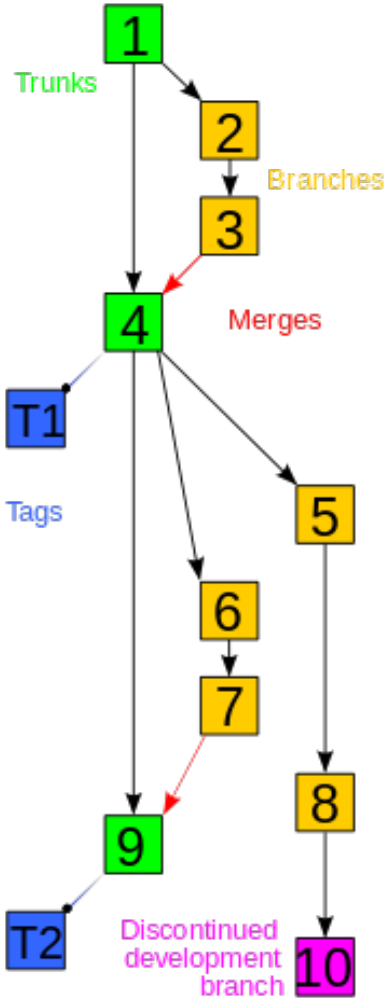
A software (component) to keep track of the history of revisions and/or different versions of files, e.g. documents or programs

Revision are associated with a person and timestamp

Dependencies between versions as a graph

Version control system: standalone software to record changes to files

Most used systems: git, Apache Subversion (SVN), CVS, other commercial systems



Source: wikipedia.org

What is git?

An open source version control system designed for distributed software development

Created in 2005 by Linus Torvalds for his work on the Linux kernel

Repository: directory managed with git, contains the entire version history in hidden .git folder

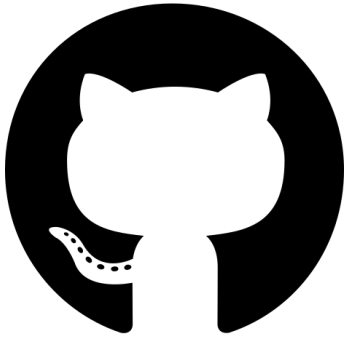
What is GitHub?

Hosting provider of public and private remote repositories

Remote repository: shared, central repository that collaborators can compare their local repositories to

Provides also functionality for documentation (wiki) and bug reporting and tracking

Alternative providers: GitLab, BitBucket, SourceForge, Launchpad, etc.



Why use version control in research?

Get some order in the mess of projects by routinely keeping track of changes in

- data
- software code/scripts
- manuscripts for papers

Share and publish code, data and documents

Make collaboration easier and attribution of work transparent

Example 1: https://github.com/mcc-apsis/git-intro

mcc-apsis / **git-intro**

Watch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Security | Insights

Dismiss

Join GitHub today

GitHub is home to over 36 million developers working together to host and review code, manage projects, and build software together.

Sign up

This is a repository holding material for a introductory git workshop

17 commits | **1** branch | **0** releases | **3** contributors

Branch: **master** | New pull request | Find File | Clone or download

mcallaghan add git documentation — Latest commit `4eb6b71` 23 hours ago

slides	add git documentation	23 hours ago
.gitignore	add tex auxiliary files to gitignore	3 days ago
README.md	update preparation description	3 days ago
github-git-cheat-sheet.pdf	initial commit	9 days ago

README.md

git-intro

This repository holds material for an introductory git workshop.

Example 2: https://github.com/mcc-apsis/plpr-scraper/

mcc-apsis / **plpr-scraper**
forked from Datenschule/plpr-scraper

Watch 2 | Star 0 | Fork 8

Code | Pull requests 0 | Projects 0 | Security | Insights

Dismiss

Join GitHub today

GitHub is home to over 36 million developers working together to host and review code, manage projects, and build software together.

Sign up

Parser für die Plenarprotokolle des Bundestags https://www.bundestag.de/plenarprotok...

164 commits | **6** branches | **0** releases | **7** contributors | MIT

Branch: **master** | New pull request | Find File | Clone or download

This branch is 99 commits ahead of Datenschule:master. | Pull request | Compare

yuantinglee removed test notebook — Latest commit 49d0dd7 3 days ago

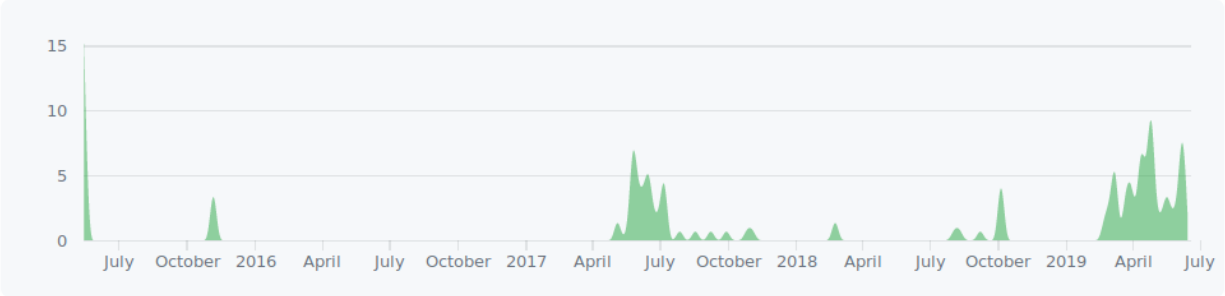
data	new files added	3 months ago
scraper	removed test notebook	3 days ago
tests	make installable and testable	last month
.gitignore	update gitignore	last month
.travis.yml	add basic test and run in Travis CI	2 years ago
Dockerfile	wp19 scraper	4 months ago
LICENSE	wp19 scraper	4 months ago
README.md	wp19 scraper	4 months ago
akoma_convert.py	wp19 scraper	4 months ago

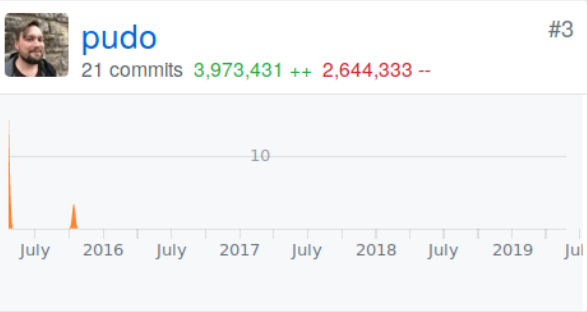
Example 2: https://github.com/mcc-apsis/plpr-scraper/

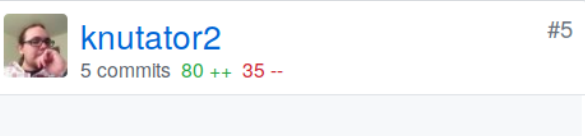
Jun 7, 2015 – Jul 11, 2019

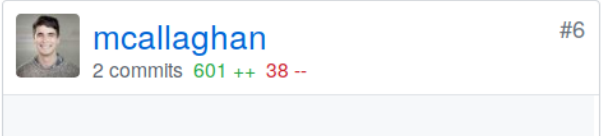
Contributions: **Commits**

Contributions to master, excluding merge commits









Example 2: https://github.com/mcc-apsis/plpr-scraper/

master origin/master fix missing text passages	fmhansen	6 days ago
fix missing text passages	fmhansen	6 days ago
parser updates for agenda points and repeated utterances	yuantinglee	7 days ago
fix repeated utterances, speakers in correct order	yuantinglee	Jul 3, 16:53
remove frozen cell	yuantinglee	Jul 2, 16:14
update syntax	yuantinglee	Jul 2, 11:14
testing new parser format	yuantinglee	Jul 1, 17:20
testing removing repeating utterances	yuantinglee	Jun 28, 17:39
testing removing repeating utterances	yuantinglee	Jun 27, 16:54
minor syntax update	yuantinglee	Jun 27, 13:58
update regular expressions	yuantinglee	Jun 21, 16:09
minimalise imports	yuantinglee	Jun 21, 12:13
capture Dr. from speaker names	yuantinglee	Jun 11, 17:50
rename old scraper	yuantinglee	Jun 11, 17:46
Merge branch 'master' of https://github.com/mcc-apsis/plpr-.	yuantinglee	Jun 11, 16:42
update speakers	yuantinglee	Jun 11, 16:41
Merge branch 'master' of https://github.com/mcc-apsis/plp	fmhansen	Jun 11, 16:37
make installable and testable	fmhansen	Jun 11, 16:37
update regex to capture some speakers	yuantinglee	Jun 6, 16:07
update gitignore	yuantinglee	Jun 4, 13:38
bug fix	fmhansen	Mai 29, 13:42
modifacations for period 13	fmhansen	Mai 28, 17:37

fmhansen <finnmh@posteo.de>
05.07.2019 11:35:27 +0200

fix missing text passages

cdd42a6c946ebfc5c3fa5ffec4719242738c3dde

```
252  265                                ut.save()
     266 +                              if self.v > 1:
     267 +                                  print("tag = name", speaker)
253  268
254  269                        elif uts.tag == "p" and uts.get("klasse") in spe
```

What are git and GitHub and why should I use them?

How can I use git and GitHub?

Git Workflow (simplest)

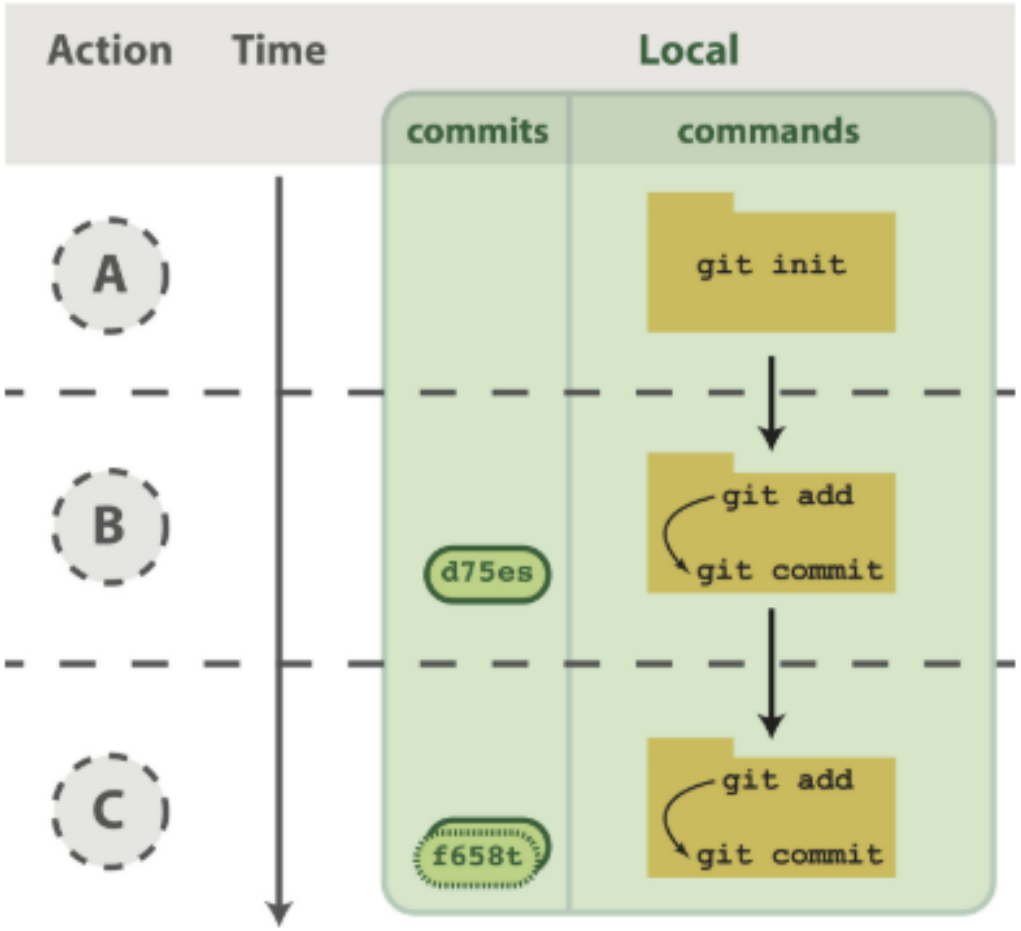


Figure: Blischak et al. (2016)

- Keep track of changes in a folder on your computer

Git Workflow (simplest)

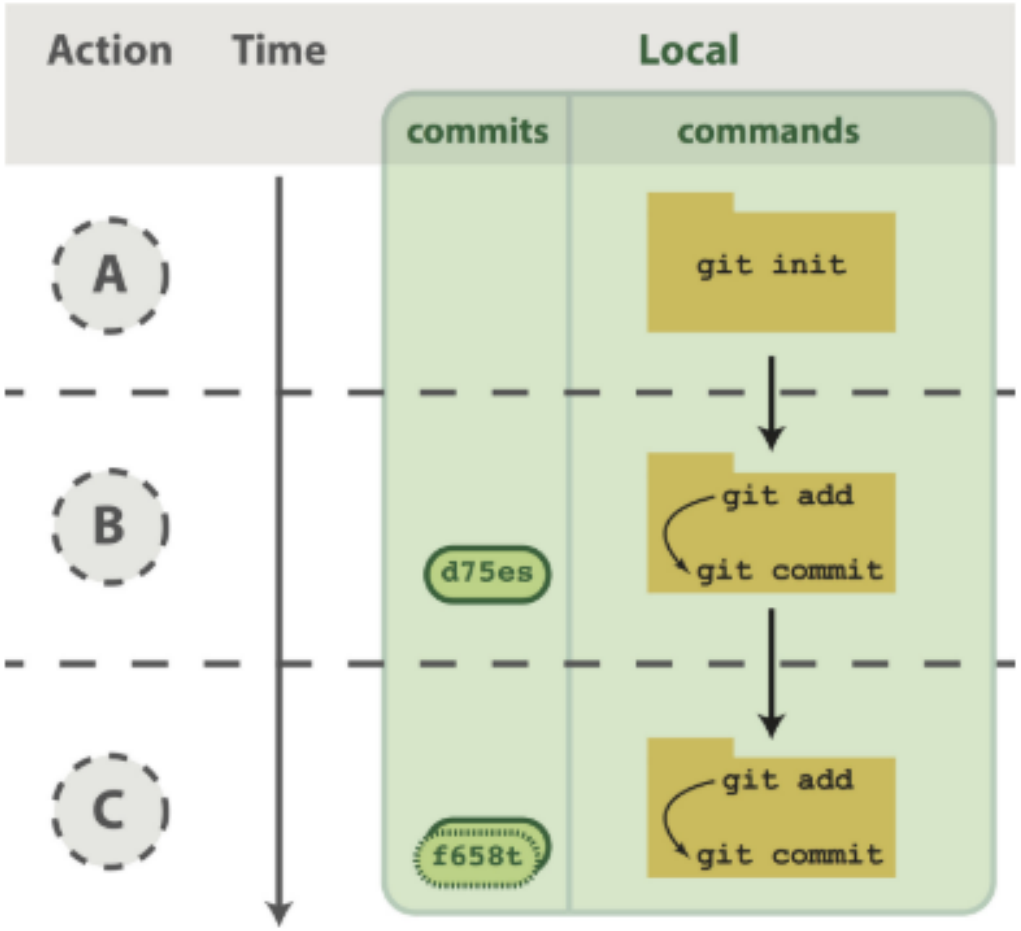


Figure: Blischak et al. (2016)

- Keep track of changes in a folder on your computer
- Changes are stored as lines added and removed

Git Workflow (simplest)

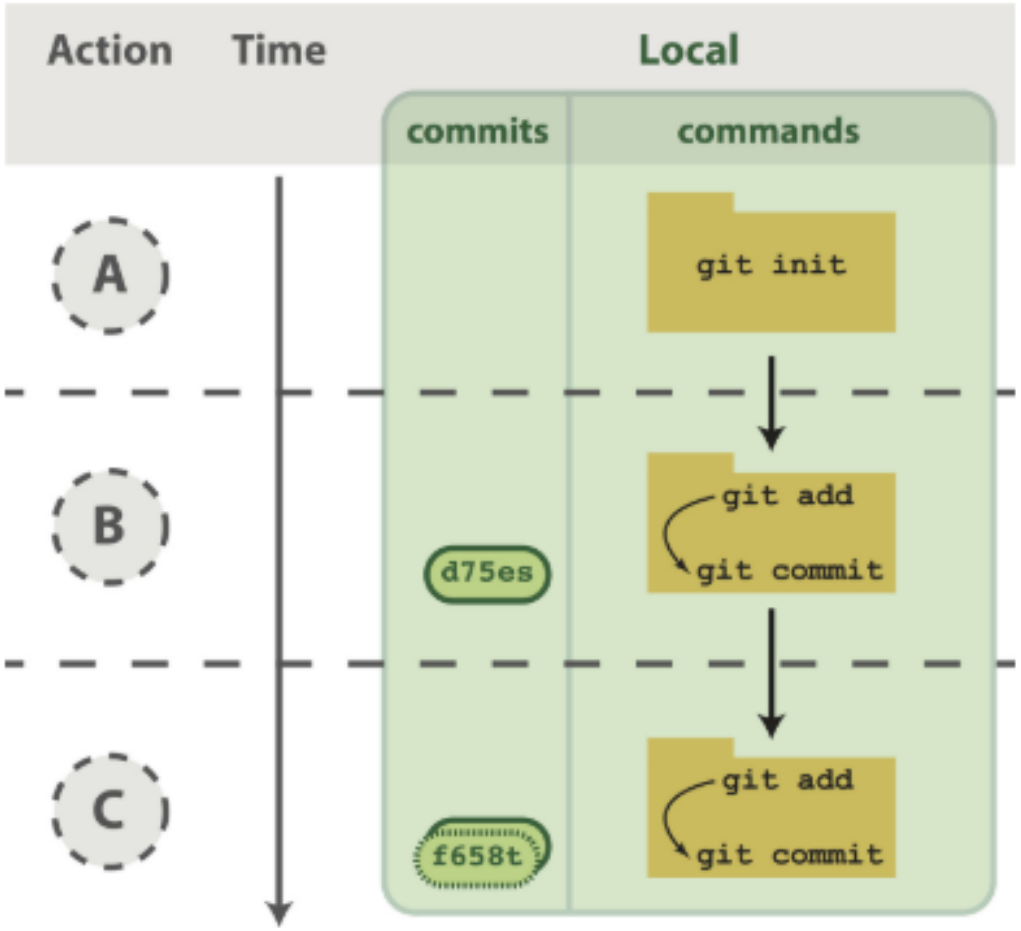
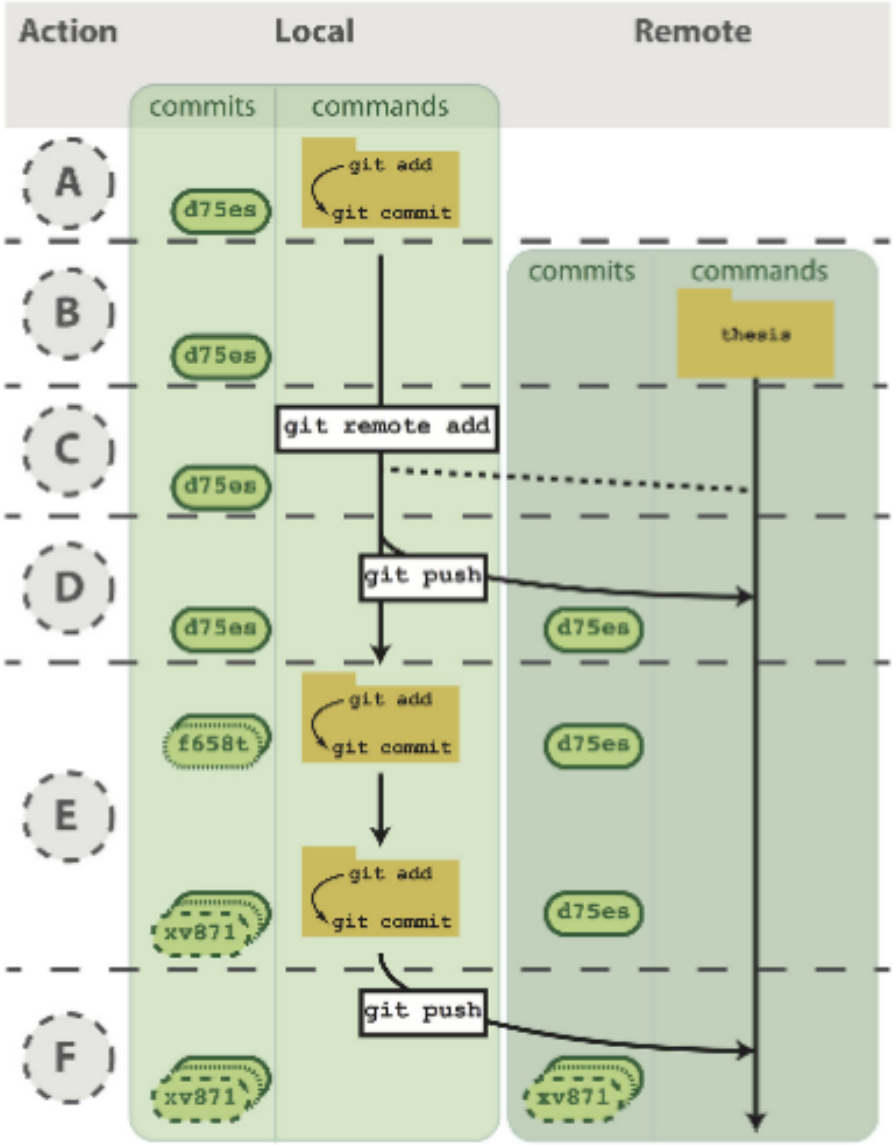


Figure: Blischak et al. (2016)

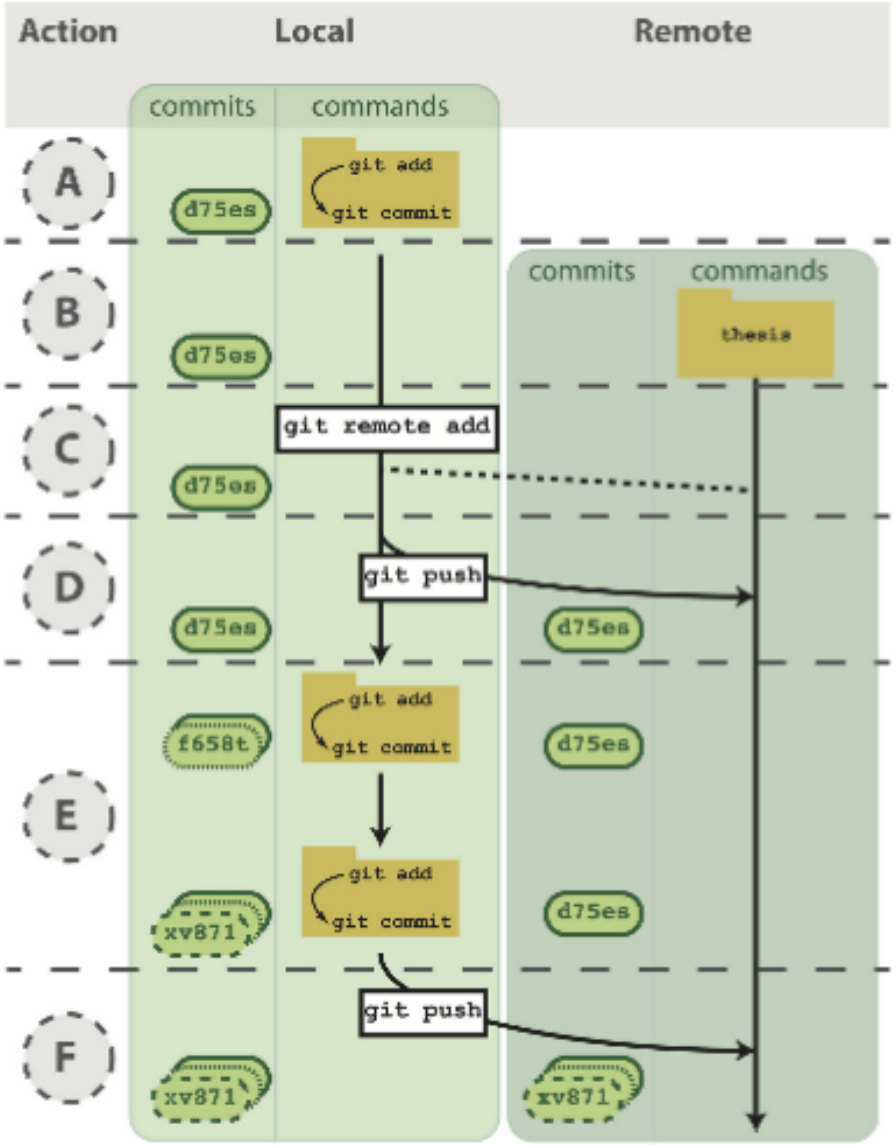
- Keep track of changes in a folder on your computer
- Changes are stored as lines added and removed
- No need to save multiple versions of the same file; you have recorded all changes and can view or revert these at any time

Git + Github Workflow (simplest)



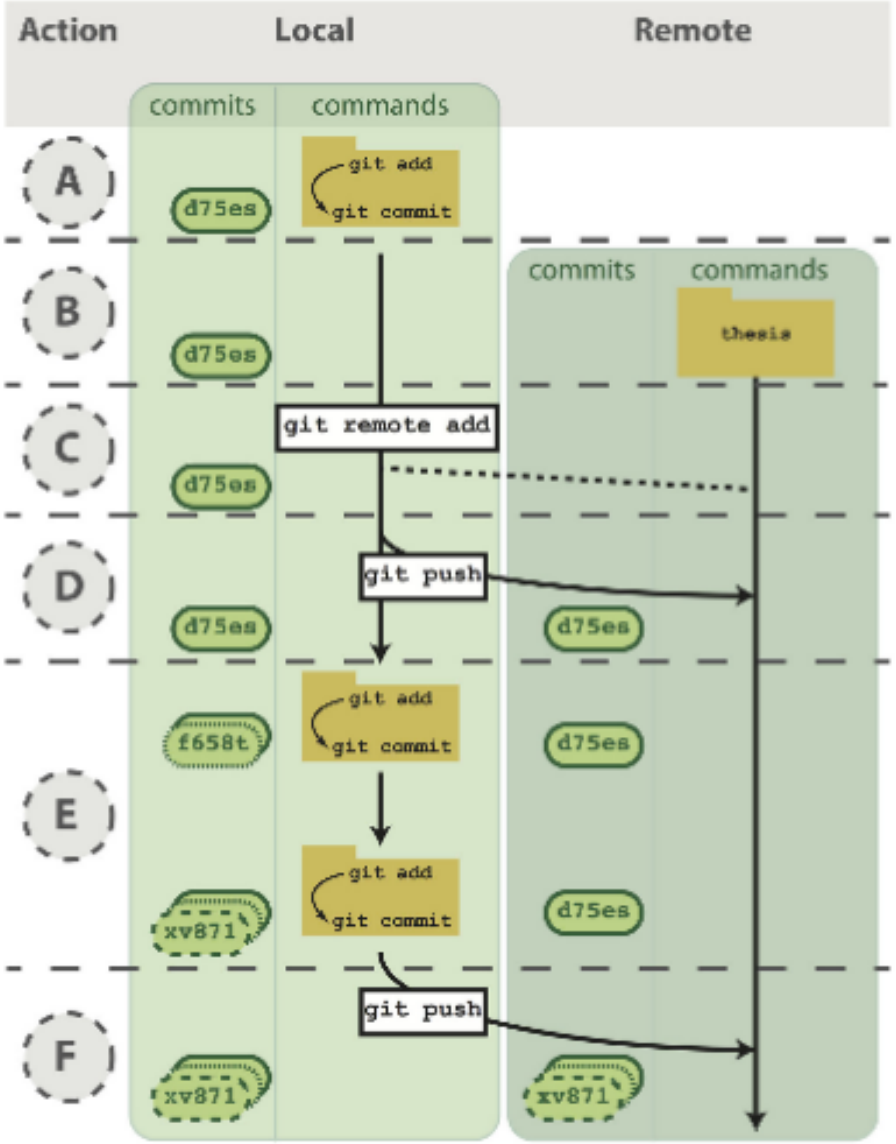
- Attach your repository to a remote version

Git + Github Workflow (simplest)



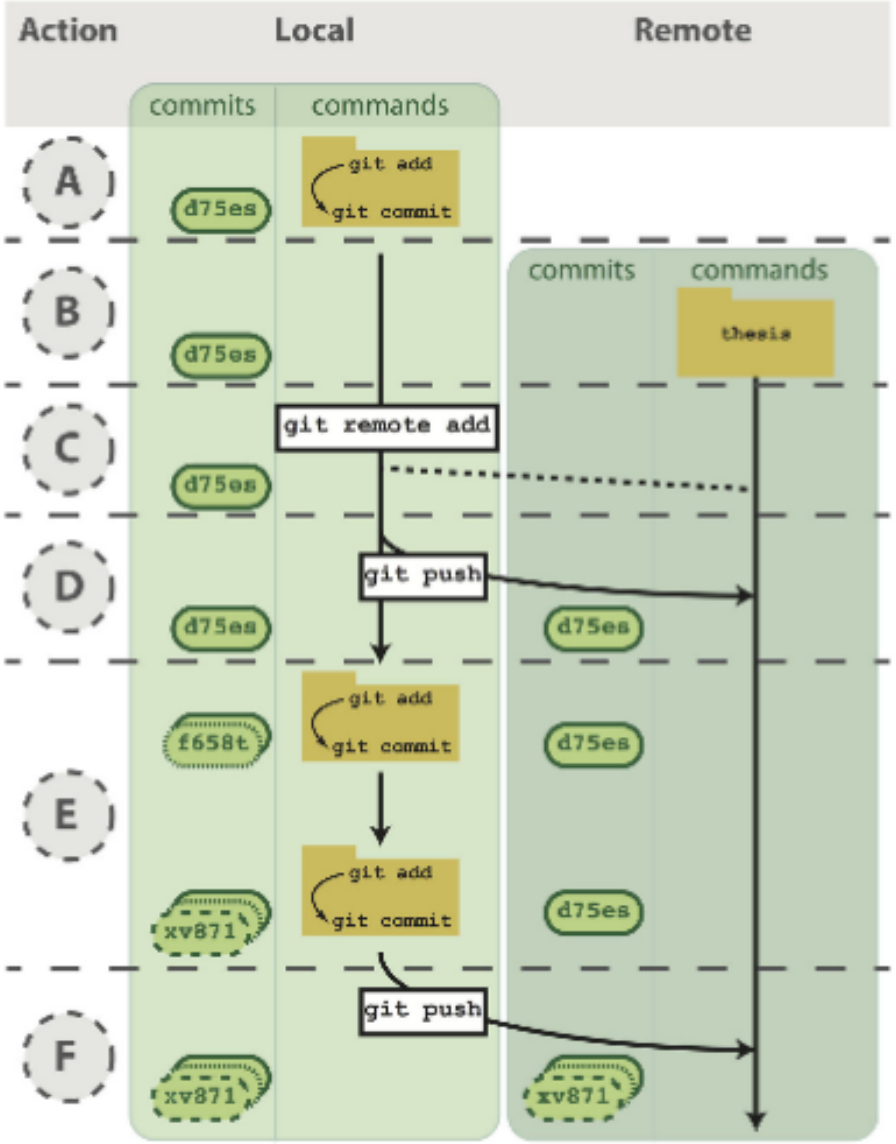
- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (**`clone`**) on their machine

Git + Github Workflow (simplest)



- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (**`clone`**) on their machine
- By both using **`pull`**, you can keep up to date with each others' changes

Git + Github Workflow (simplest)

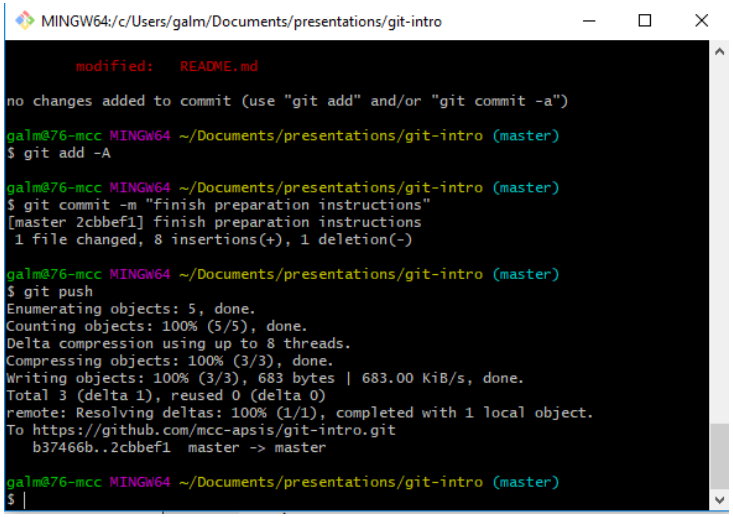


- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (**`clone`**) on their machine
- By both using **`pull`**, you can keep up to date with each others' changes
- For more complicated workflows, especially where maintaining a working version is critical, check out branching `https://guides.github.com/`

Tools

Command Line

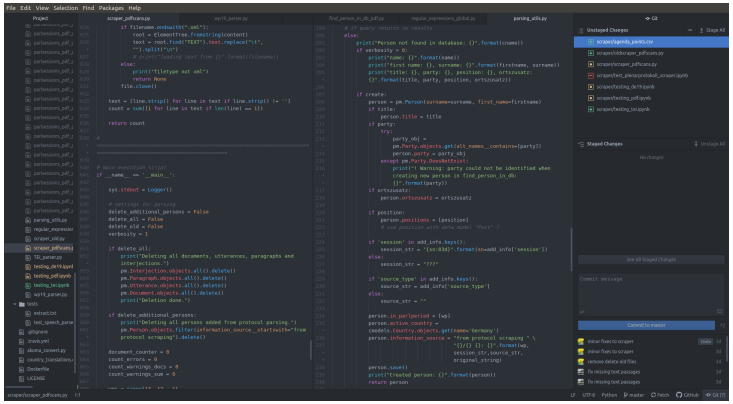
- Easy to document/explain



Steeper learning curve, but more flexible and harder to do things unintentionally

GUIs

- Easy to use



Often there are integrations in development environments, e.g. RStudio, Atom

Starting a repository

To start working with a repository, either turn an existing folder into a git repository

```
git init
```

or copy an existing remote repository into a folder on your machine

```
git clone
```

Editing a respository

- Edit files (write some new code or a nice new paragraph)

Editing a respository

- Edit files (write some new code or a nice new paragraph)

- Stage changes (tell git about the changes you want record)
  - **`git add -A`**
  - Or add only certain files using patterns or exact file names

---

Editing a respository

- Edit files (write some new code or a nice new paragraph)

- Stage changes (tell git about the changes you want record)
  - **`git add -A`**
  - Or add only certain files using patterns or exact file names

---

- Commit changes (make a timestamped version of the repository, recording all the changes you have told git about)
  - **`git commit -m "made a cool new graph"`**
  - It's best if each commit describes a discrete change, and has an interpretable name.

Managing the repository

Where are we?

`git status` tells us which files have changed and are staged or unstaged:

```
galm@76-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   slides/presentation.pdf
        modified:   slides/presentation.tex

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

        modified:   slides/images/git-terminal.png
```

What's changed?

`git diff` lets us know the difference between the files we could stage, and the staged version of them

Managing the repository

Where are we?

`git status` tells us which files have changed and are staged or unstaged:

What's changed?

`git diff` lets us know the difference between the files we could stage, and the staged version of them

```
$ git diff slides/presentation.tex
diff --git a/slides/presentation.tex b/slides/presentation.tex
index 5d65679..3c1a020 100644
--- a/slides/presentation.tex
+++ b/slides/presentation.tex
@@ -274,14 +274,18 @@ or copy an existing repository into a folder

 \medskip

-\code{git status}
+\code{git status} tells us which files have changed and are staged or unstaged:
+
+\begin{figure}
+       \includegraphics[width=\linewidth]{images/git-status}
+\end{figure}

-\medskip

+\medskip

 \textbf{What's changed}

-\code{git diff}
+\code{git diff} lets us know the difference between the files we could stage, and the s
taged version of them
```

Managing the repository

Where are we?

`git status` tells us which files have changed and are staged or unstaged:

What's changed?

`git diff` lets us know the difference between the files we could stage, and the staged version of them

`git diff` can also tell us about the difference between variously specified versions of files

Navigating different versions

`git log` shows us a list of all the commits that have been made.

```
galm@76-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git log
commit 2cbbef100ff738f182676f7bfe420691340a23be (HEAD -> master, origin/master, or
EAD)
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date:   Mon Jul 8 15:14:06 2019 +0200

    finish preparation instructions

commit b37466bf9a88bd481887769415375f1e95343f3a
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date:   Mon Jul 8 15:03:56 2019 +0200

    preparation instructions in README.md

commit 569f525ab48e1fb922495b204ab1b91d07229b40
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date:   Mon Jul 8 13:50:34 2019 +0200

    updated slides
```

`git checkout` takes you to another commit, or another branch of the repository

`git checkout master` takes you back to the most recent commit on the master branch

Interacting with remote repositories

If you started a repository on your computer, you can associate it with an *empty* GitHub repository. Follow the instructions on Github:

```
git remote add origin
https://github.com/mcallaghan/blablabala.git
```

```
git push -u origin master
```

Now your local repository exists online, where it's current status and entire history is reflected.

N.B. when dealing with the public domain, don't forget licenses (of your and other people's work).

Interacting with remote repositories

`git pull` downloads all of the changes (commits) that are published in the remote version, and incorporates them (as long as they do not clash with your changes) into your local copy
Merging is normally done automatically, but you may need to choose between different versions if they have both changed the same lines of code

---

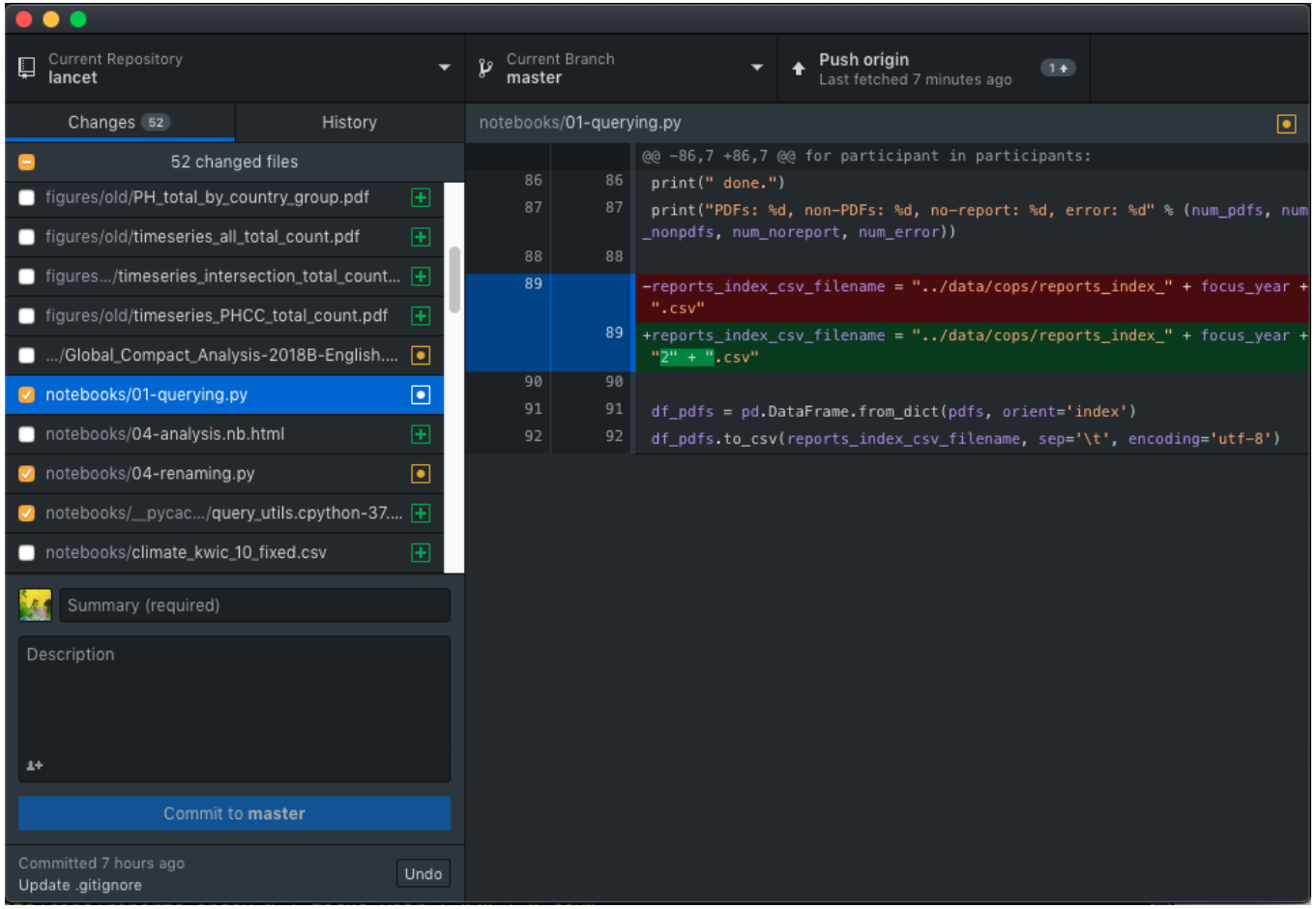
`git push` adds all of your committed changes to the remote version (as long as your version contains all of the changes in the remote - you need to **`pull`** first)

You can only **`push`** to repositories you have the correct permissions for, but you can make a **`pull request`** (a suggestion that your changes be merged into the master) for any repository. This allows changes to be reviewed and tested before incorporation.

Version control using GUIs

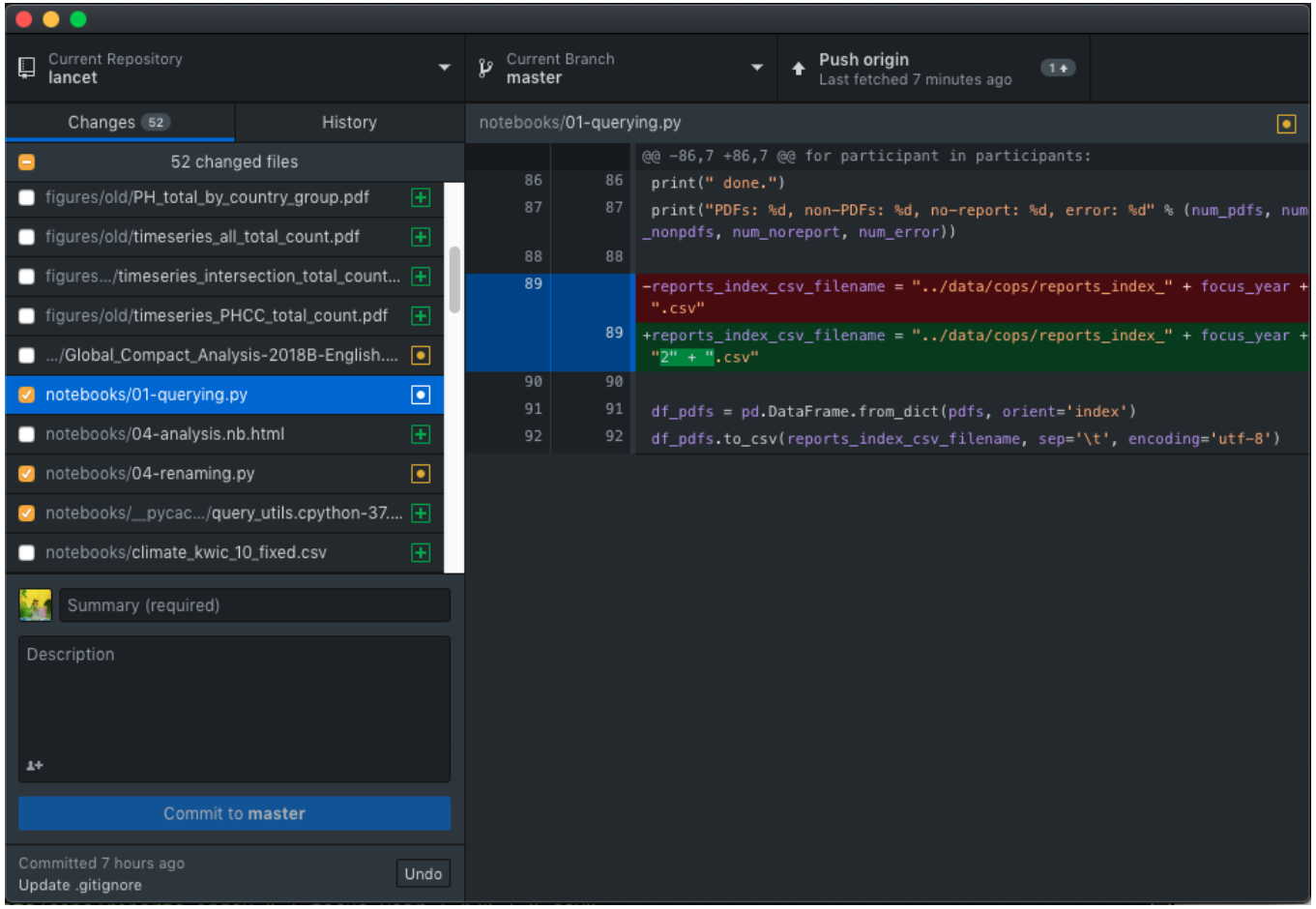
You can also employ version control using a GUI such as Github Desktop, or within code editors e.g. Atom

Version control using GUIs



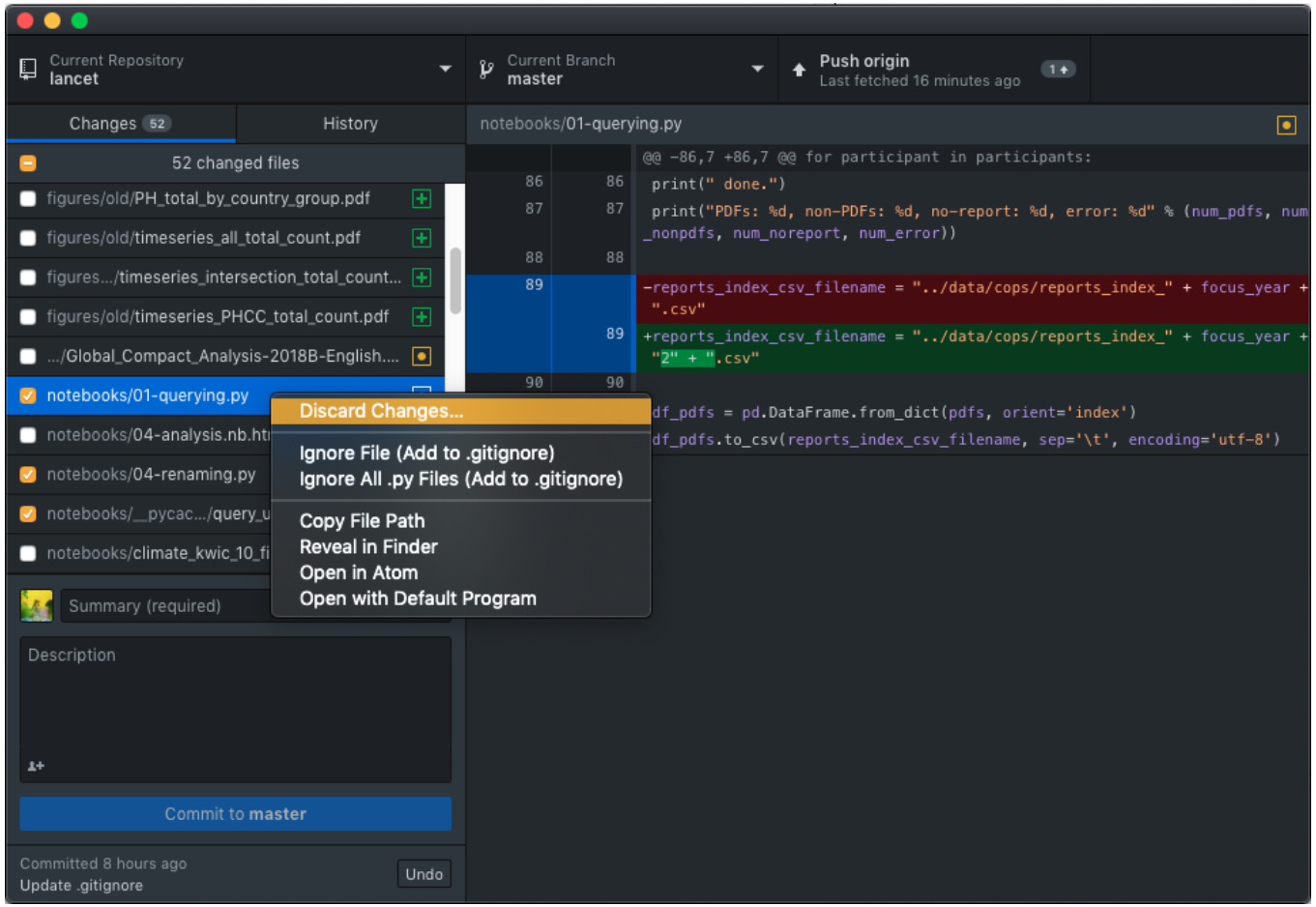
The sidebar shows an equivalent of **`git status`** - you can see which files have been created or changed, and whether they have been added or staged for commit or not

Version control using GUIs



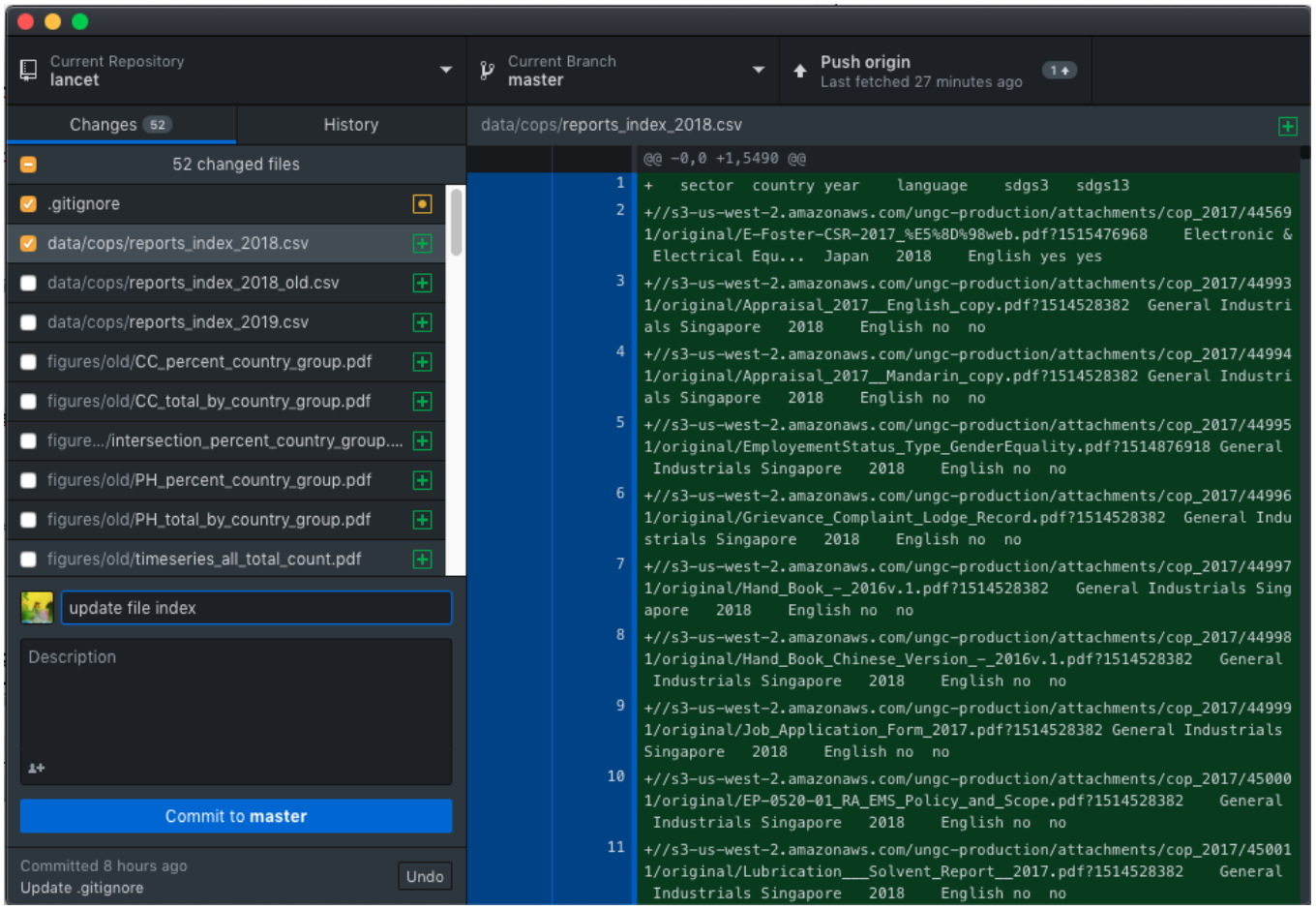
You can also see the changes you've made to files, similar to **`git diff`**

Version control using GUIs



There are also options for you to stage or discard your changes, or add it to a list of files where changes are ignored

Version control using GUIs



You can then review your changes and commit them, then push to the remote from the interface

Further useful commands and tools

- **`.gitignore`** - A list of files, or patterns for git to ignore
- Zenodo lets you get a citable DOI for your code/data
- GitHub Pages allows you to quickly turn your repository into a static website
- Blame lets you trace, line by line, who contributed which parts of the code, and when

Github also has nice tools for making wikis and tracking issues, among other things, and can act as cloud storage, and a convenient way to browse the history of your project.

N.B. Sometimes Git is confusing! If you are about to do something you don't quite understand, you can copy the whole folder into another location for safe keeping. It's hard to lose data you have committed, but it is possible to lose changes you haven't committed.

Questions?

Practice / task

- Make a new project folder, and initialise a git repository with **`git init`**
- Add a README.md file with some text in it.
- Stage and commit your changes with **`git add`**, and **`git commit`**
- Make a new empty repository on Github `https://github.com/new`
- Follow the instructions to add this remote to your local repository with **`git remote add origin [link]`** and push your changes **`git push -u origin master`**
- Find a partner, add each other as collaborators, edit each other's repositories, and push changes.

References

Blischak, J. D., Davenport, E. R., and Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology*, 12(1):e1004668.

Git is abundantly documented `https://git-scm.com/doc`

Most "How do I?" git queries can be answered via a google search