CHALMERS
UNIVERSITY OF TECHNOLOGY

Git Tutorial

Filip Buric
January 2020

www.sysbio.se

Overview

- version control
- basic git   (command line)
- exercises!

- Ask whenever confused

Perfect Reproducibility

I have:

results = program(data, parameters)

And data never changes.

Question: What do you need from me to get the same results?

Perfect Reproducibility

results = program(data, parameters)

A given set of results is **determined uniquely** by

- the program code
- parameter values

Source data should never be altered.

Real World

- Code is very fluid
- Actual results reflect some intermediate (“slightly different”) code version

Show of hands

1. If you use Google Drive, Dropbox, or Box

2. If you use any backup software/method for your personal computer

Version Control

- **Issue**: Files with long, complicated history.
  Want to keep different versions:

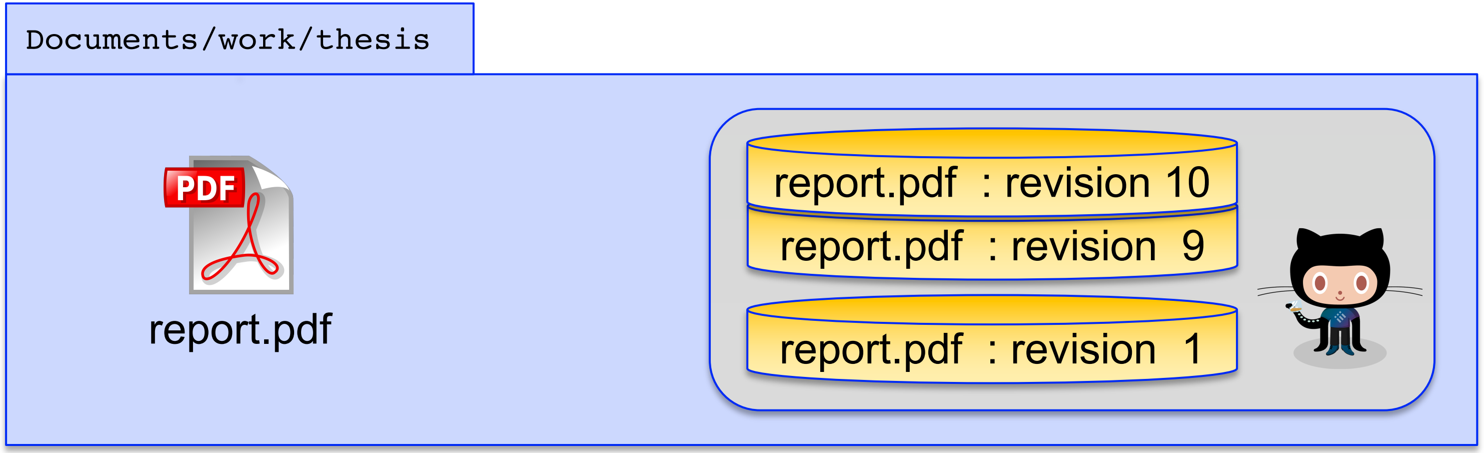
```
Report_v3_comments_2018_01_05.docx
experiment_pipeline_10_2017_11_05.sh
```

- **Compound issue:** Other people work on them too

- Programs like **git** (version control systems) keep track of changes made by different people

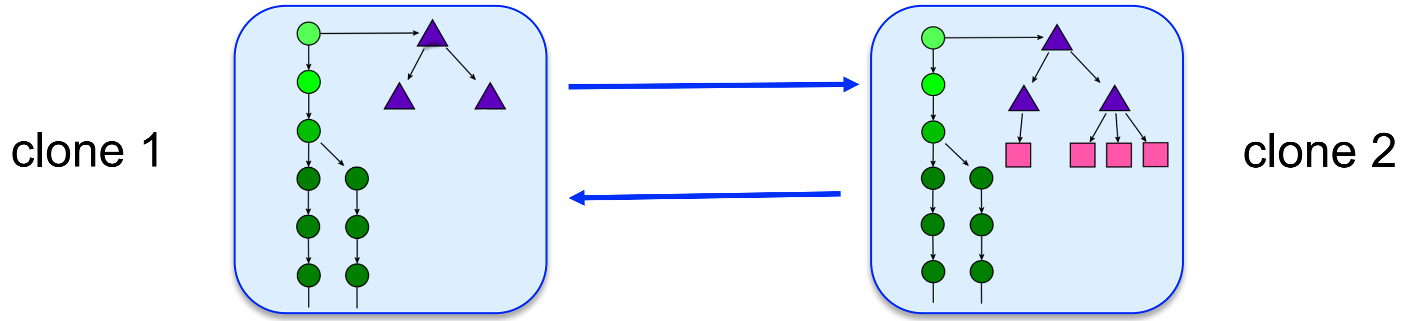
Git Concepts

- a git project is called a ***repository*** or ***repo*** = directory with history
- a repo contains a collection of snapshots (called ***revisions***) of the directory:



Git Concepts

- revisions are connected in ***branches***, reflecting file evolution



Original image © Bunyk / Wikimedia Commons / CC-BY-SA-4.0

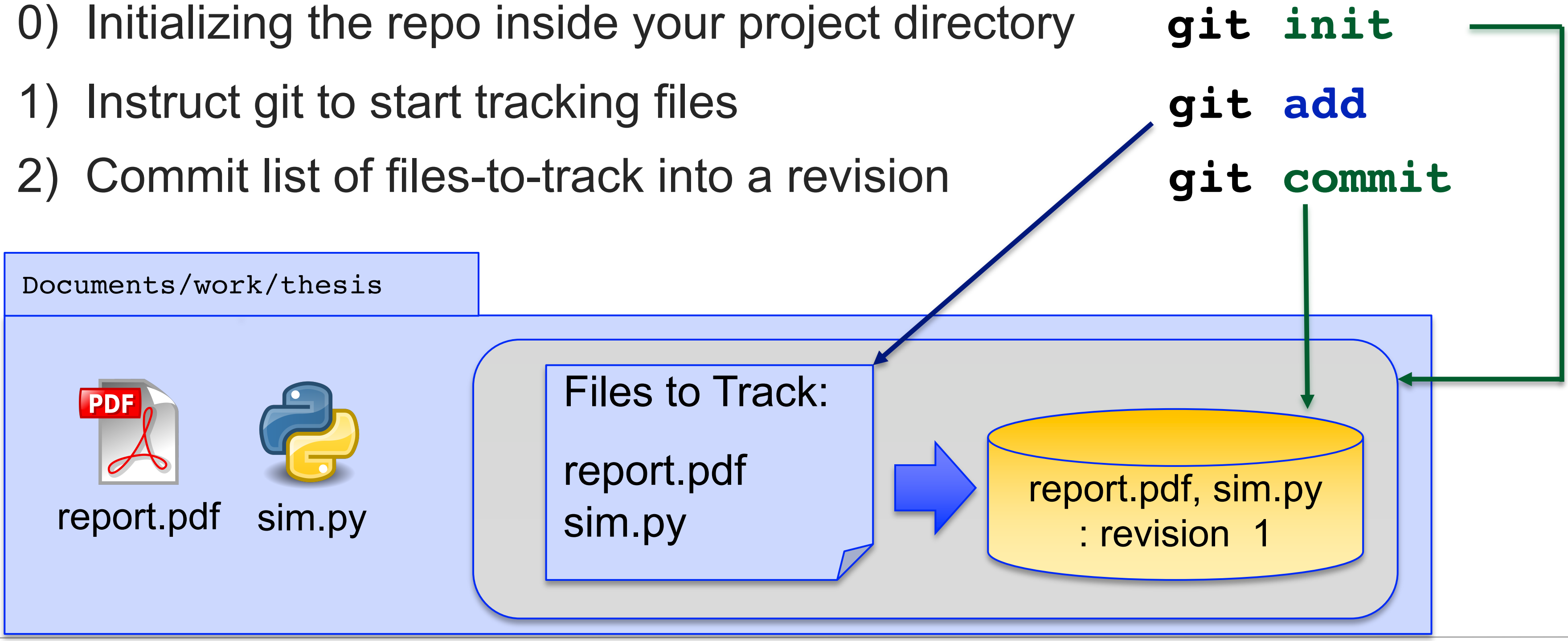
- repos are *decentralized*
  - Each ***clone*** contains everything (all revisions + history)
  - Changes can be passed between clones

Creating a Repo and Recording Changes

0) Initializing the repo inside your project directory `git init`

1) Instruct git to start tracking files `git add`

2) Commit list of files-to-track into a revision `git commit`

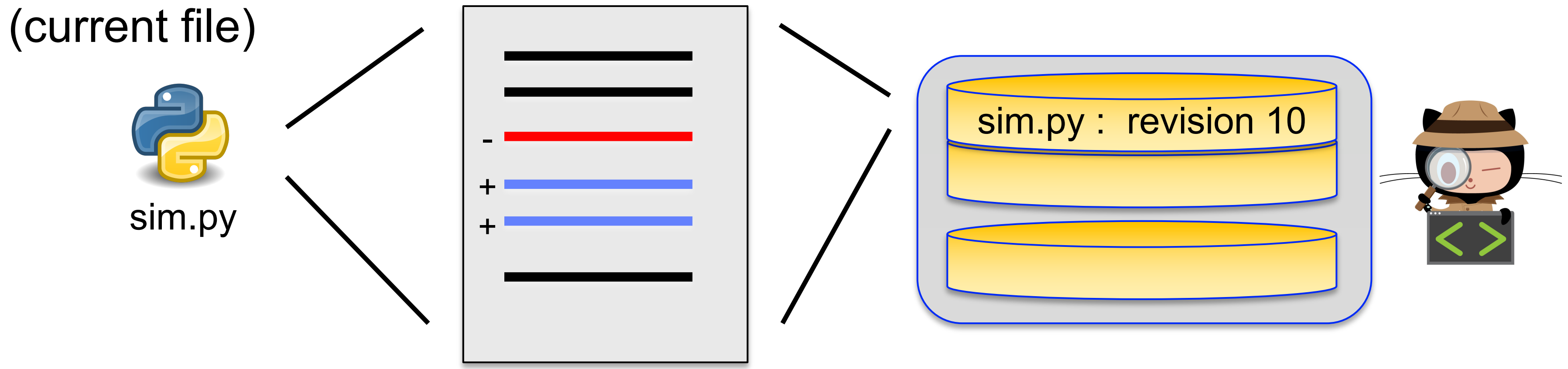


Exercise 1

- **10-15** minutes
- Go to `https://mpbio-bbt015.github.io/`
- If you need to, read "How to connect to remote accounts"
- Notes are good-to-know info only

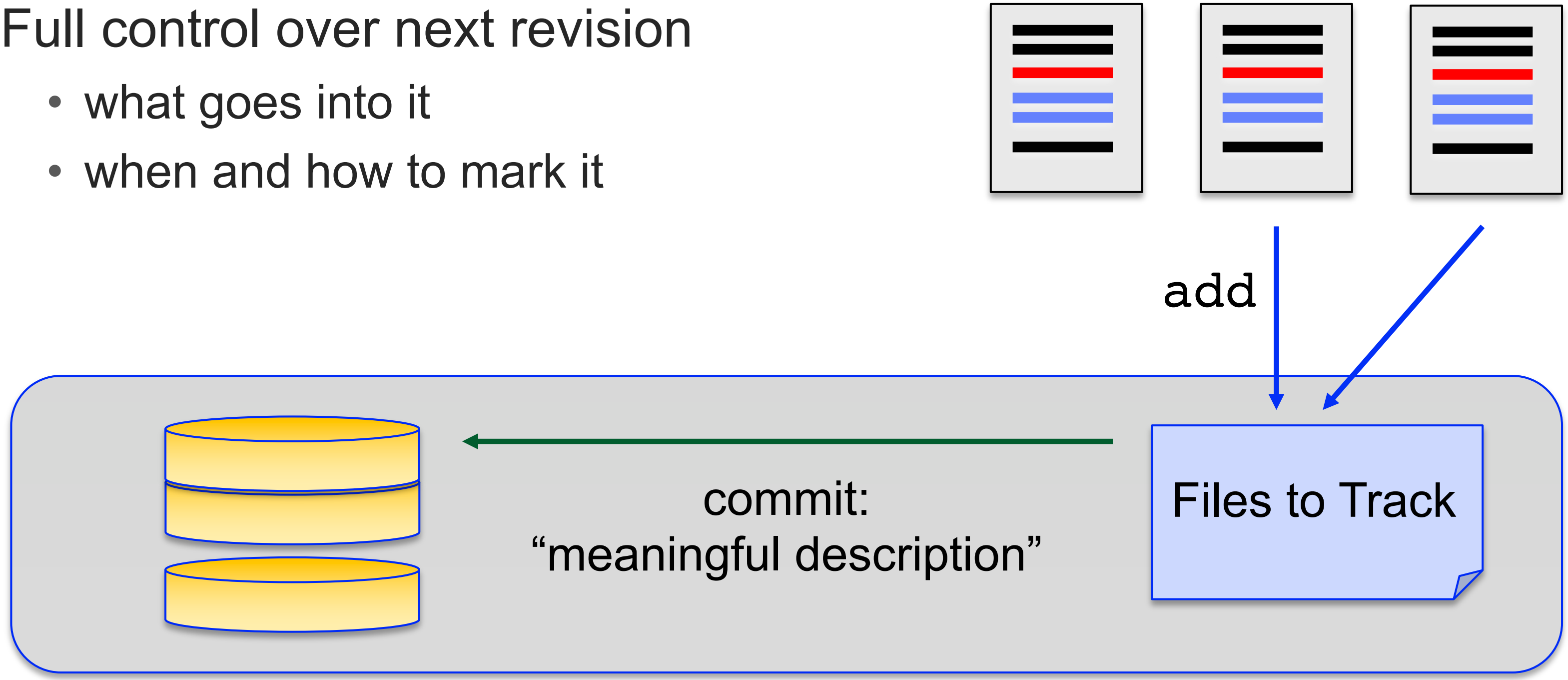
Making and Committing Changes

- Git reports what changed since latest revision: `git status`
- Differences can be inspected: `git diff`



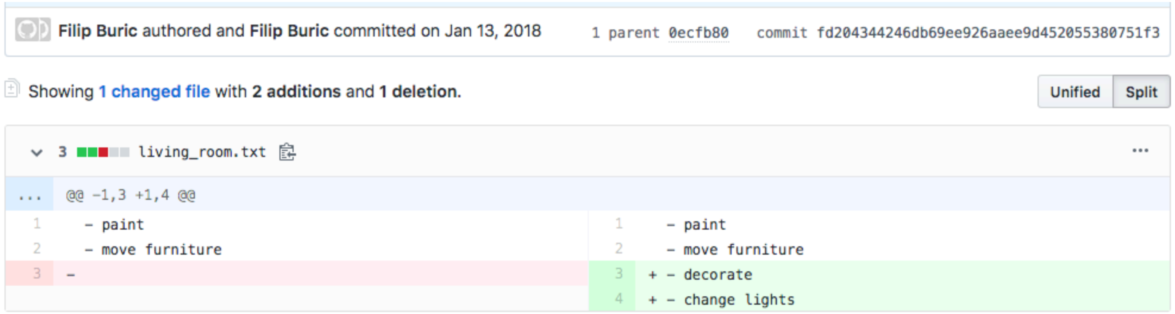
Making and Committing Changes

- Full control over next revision
  - what goes into it
  - when and how to mark it



Viewing Differences

- Web (GitHub) or GitHub Desktop (= graphical frontends to the git program)
  - Easier to use
  - Limited functionality

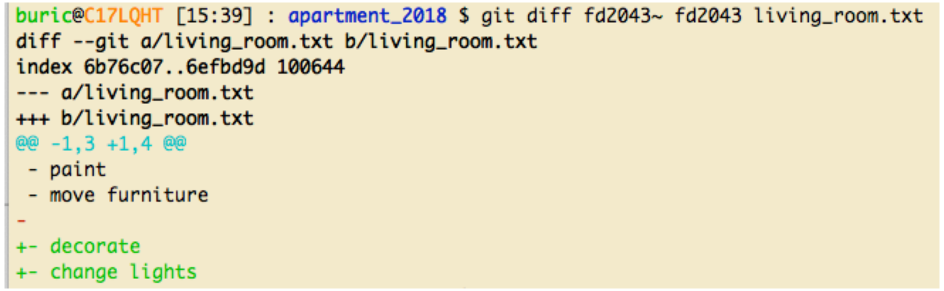


Filip Buric authored and Filip Buric committed on Jan 13, 2018   1 parent 0ecfb80   commit fd204344246db69ee926aaee9d452055380751f3

Showing 1 changed file with 2 additions and 1 deletion.

```
3 living_room.txt
...  @@ -1,3 +1,4 @@
1    - paint                         1    - paint
2    - move furniture                2    - move furniture
3  -                                 3  + - decorate
                                     4  + - change lights
```

- Command line:
  - More cumbersome
  - Far more flexible
  - Always available



```
buric@C17LQHT [15:39] : apartment_2018 $ git diff fd2043~ fd2043 living_room.txt
diff --git a/living_room.txt b/living_room.txt
index 6b76c07..6efbd9d 100644
--- a/living_room.txt
+++ b/living_room.txt
@@ -1,3 +1,4 @@
 - paint
 - move furniture
-
+- decorate
+- change lights
```

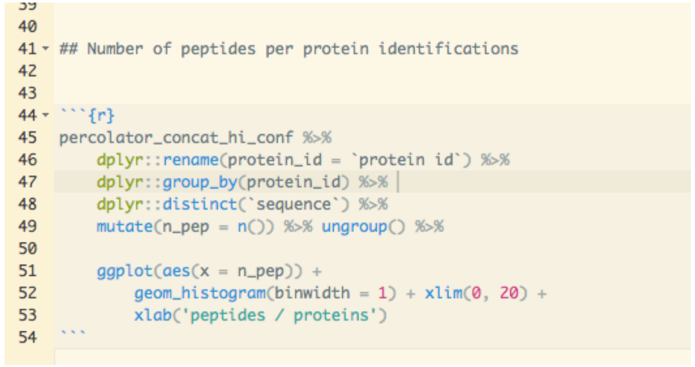
Git is meant for text files

- Tool for tracking source code (= text / “low level”)
- Can track any type of file BUT can’t see diffs (without extra plugins)

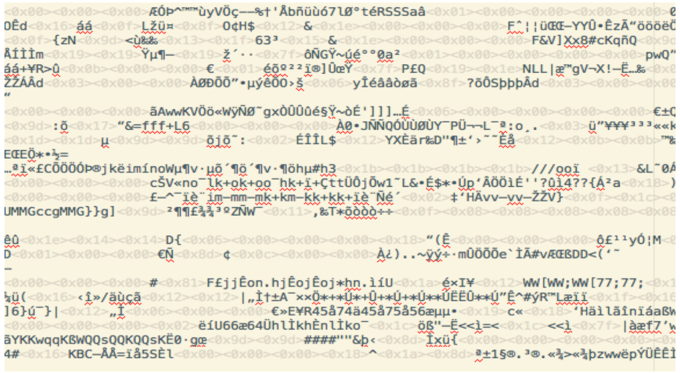
`report.md` → `report.pdf`

Markdown: text, readable by any program

PDF: “binary” format, encodes graphics, needs dedicated decoder (Adobe, etc)



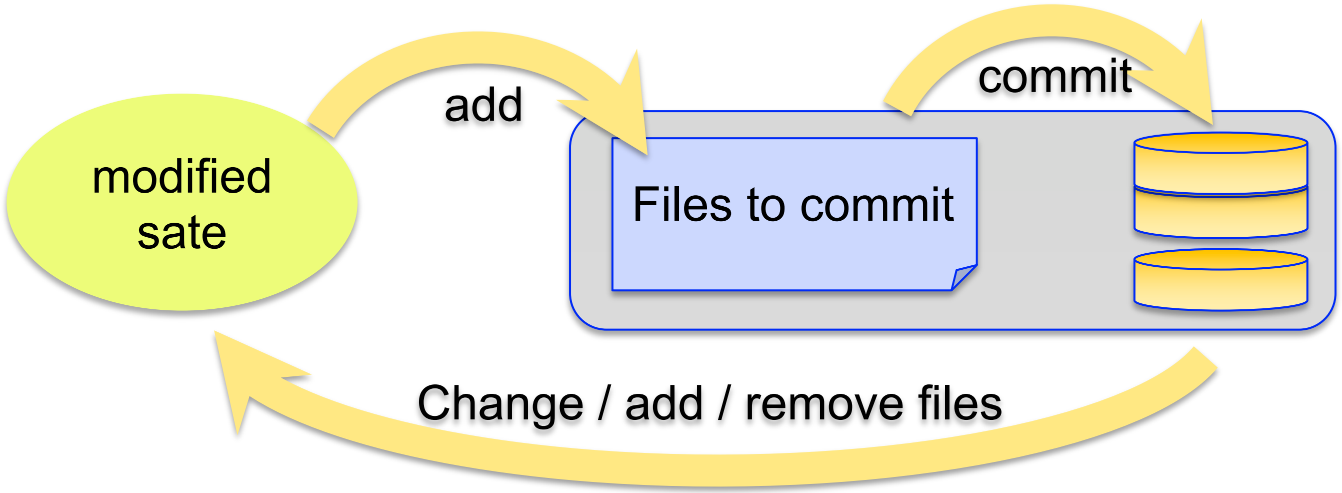
~~~
## Number of peptides per protein identifications

```{r}
percolator_concat_hi_conf %>%
    dplyr::rename(protein_id = `protein id`) %>%
    dplyr::group_by(protein_id) %>%
    dplyr::distinct(`sequence`) %>%
    mutate(n_pep = n()) %>% ungroup() %>%

    ggplot(aes(x = n_pep)) +
        geom_histogram(binwidth = 1) + xlim(0, 20) +
        xlab('peptides / proteins')
```
~~~



(only need to track this)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Typical Work Loop



Pop Quiz! (yaaay…)

- Go to `socrative.com` > Student Login
- Room number: BBT045

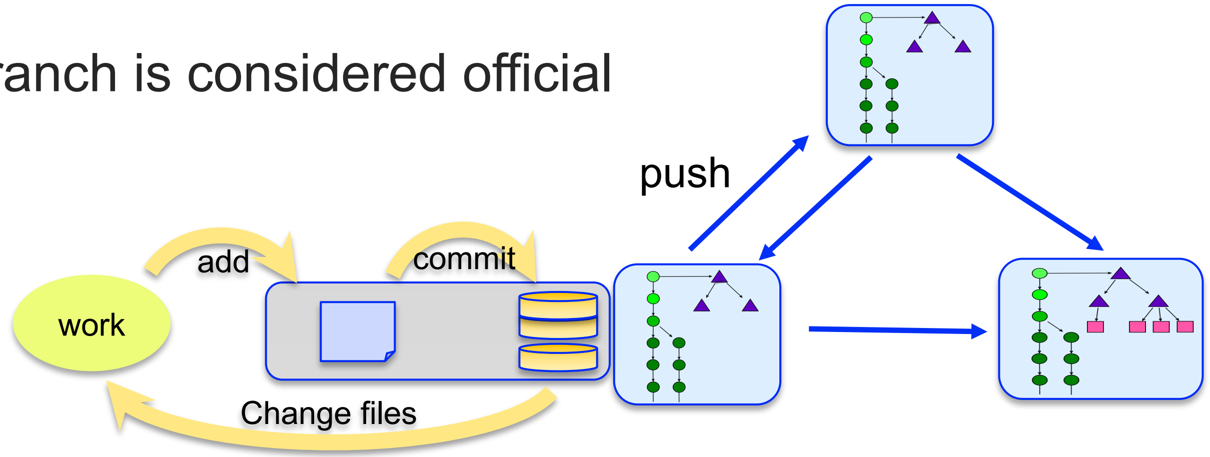
Exercise 2

- **15** minutes

Collaborating

- Convention:

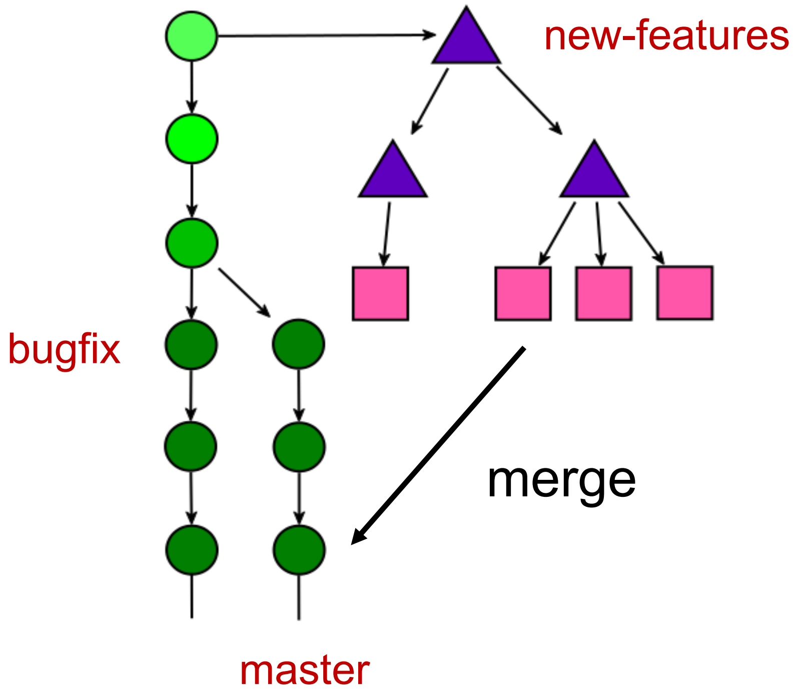
  one repo and one branch is considered official



- Collaborators:
  - clone from this repo
  - work
  - ***push*** their contributions to it

Collaborating

- Work usually done on branches:
  - maintain separation of interest
    (e.g. ”development” vs “bug fixing”)
  - isolate changes
    (e.g. “experimental” branch)



Some real examples

Using GitHub to

- inspect files
- change file

Reproducibility with Version Tracking

results = program(data, parameters)

You must track	You must share
• source code • a list of package versions • parameter values *	• source data • repo version for each result set • parameter values

* Note: Tracking parameter values in a publication repo is more stringent.
It gives a complete "snapshot" of the conditions in which results were generated.
It also gives a more comprehensive history of the project

May be omitted if the repo is a generic (multi-use) software packages but **must** be reported in any study.

Wrap-up

- Do use git to track your work – even if working alone
- Don’t be afraid to break things!   Almost always possible to recover.
- Complex tool but daily routine involves only a handful of commands

Thank you!