

Git Tutorial

Filip Buric
January 2020



Octocat logos © 2013 – 2017 GitHub, Inc.

Overview

- version control
 - basic git (command line)
 - exercises!
-
- Ask whenever confused

Perfect Reproducibility

I have:

$$\text{results} = \text{program}(\text{data}, \text{parameters})$$

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

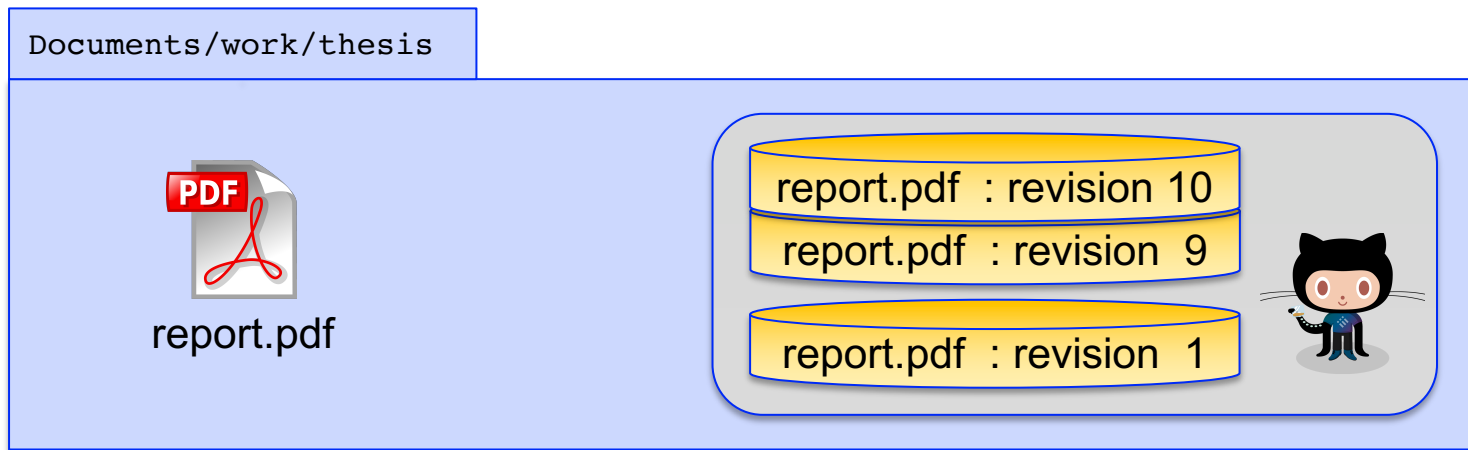
- **Issue:** Files with long, complicated history.
Want to keep different versions:

```
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

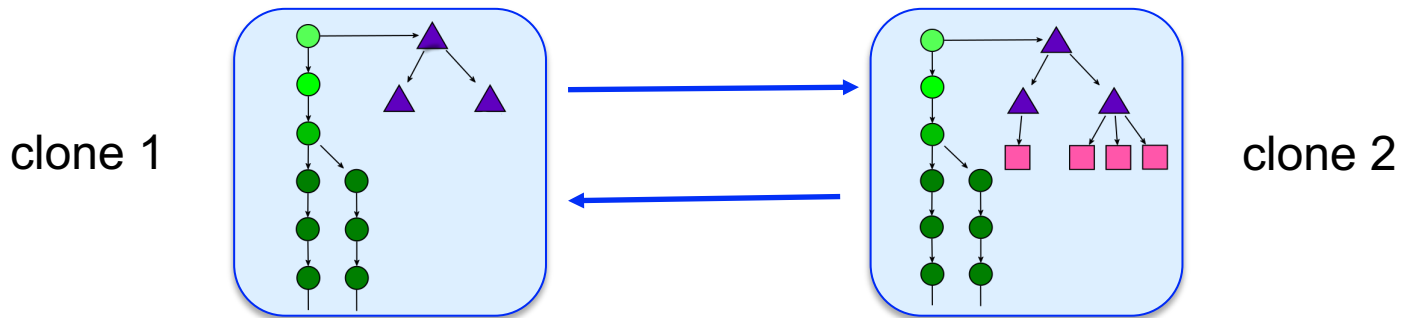
Git Concepts

- 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

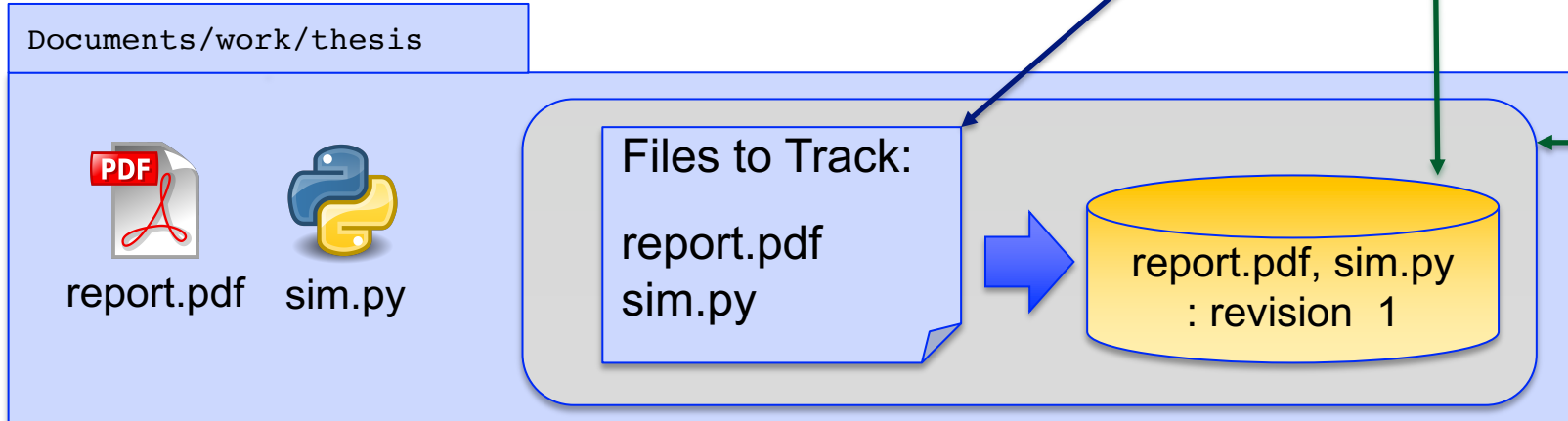
Creating a Repo and Recording Changes

- 0) Initializing the repo inside your project directory
- 1) Instruct git to start tracking files
- 2) Commit list of files-to-track into a revision

git **init**

git **add**

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

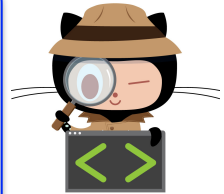
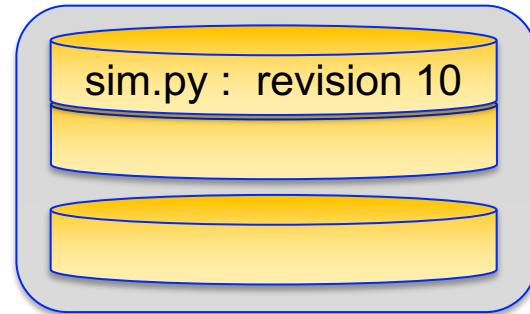
Making and Committing Changes

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

(current file)

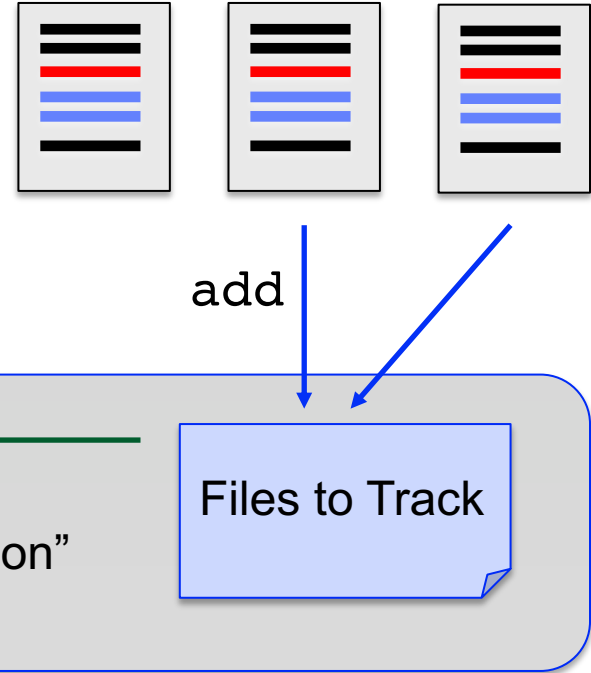


sim.py



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

Filip Buric authored and Filip Buric committed on Jan 13, 2018 1 parent 0ecfb80 commit fd204344246db69ee926aee9d452055380751f3

Showing 1 changed file with 2 additions and 1 deletion. Unified Split

3 living_room.txt

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

```
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

Markdown: text, readable by any program



report.pdf

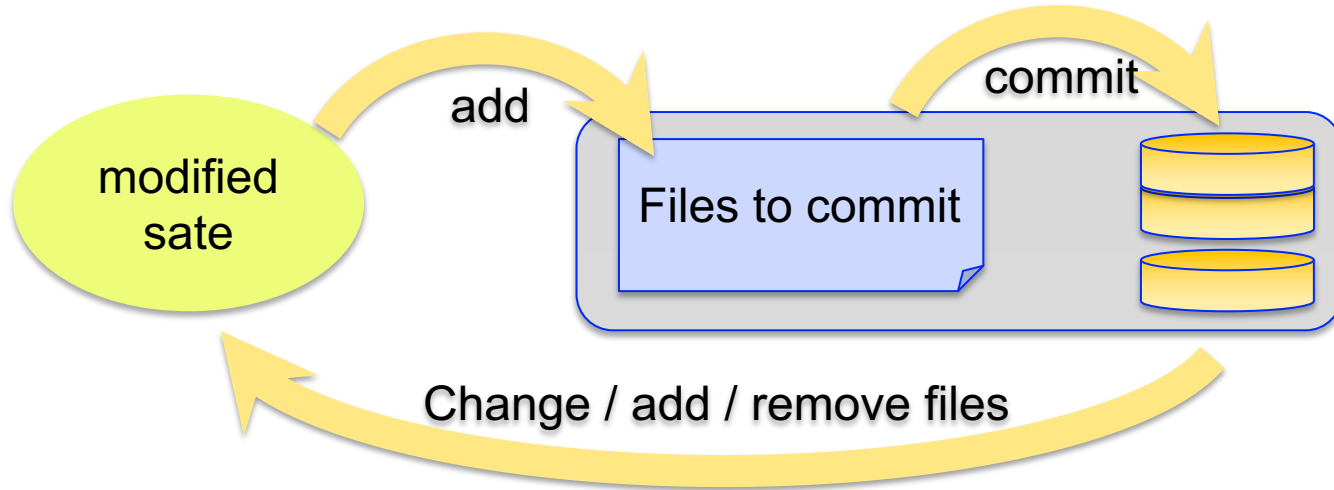
PDF: “binary” format, encodes graphics, needs dedicated decoder (Adobe, etc)

```
33
40
41- ## Number of peptides per protein identifications
42
43
44- ```{r}
45 percolator_concat_hi_conf %>%
46   dplyr::rename(protein_id = `protein id`) %>%
47   dplyr::group_by(protein_id) %>% |
48   dplyr::distinct(`sequence`) %>%
49   mutate(n_pep = n()) %>% ungroup() %>%
50
51   ggplot(aes(x = n_pep)) +
52     geom_histogram(binwidth = 1) + xlim(0, 20) +
53     xlab('peptides / proteins')
54 ```
```

(only need to track this)

```
DEd<0x00><0x00><0x00>Æ0b~üyV0ç~%t'Äbhüü67L0*téRSS5aâ<0x01><0x00><0x00><0x00>e
0E0<0x16>ââ<0x0f>Lzû=<0x8f>0cH$<0x12>6<0x1e><0x00><0x00><0x00>F_||ü0E-YVÜ-EzA"öööeC
<0x0f>{zN<0x9d><0x00><0x13><0x1f>63<0x15>6<0x1e><0x00><0x00><0x00>F5V|Xx8#cKqH0<0x9d
Ä11im<0x19><0x19>Ym1-<0x19>Z'..<0x7f>ÖNGY~üé"0a2<0x01><0x00><0x00><0x00><0x00>pwQ"
ââ+WR<0x0b><0x00><0x00>€<0x01>60022i0j0mY<0x7f>PEQ<0x19><0x00><0x00><0x00><0x00><0x00>
ZZAAd<0x03><0x00><0x00>A0000"~üy000.5<0x00>y1éââ00â<0x8f>70S0ppbAd<0x03><0x00><0x00>
"
<0x00><0x00><0x00>âAwKV00=WyN0"gx0000ésÿ~0É'j]l.É<0x00><0x00><0x00><0x00><0x00>€±C
<0x0d>:0<0x17>"6=fff+L6<0x00><0x00><0x00><0x00>Ä0•JNN000U00Y"PU~L"0.o.<0x03>ü"VYV333u0k
<0x1d><0x1d>ü<0x9d><0x00>0j0":<0x02>Éfll$<0x12>YXEärkD"q±',"É0<0x12><0x00><0x00><0x00>
EE00++±=
_0i=EC00000p0jkëimInoWmYv.ü0"qiv.ñ0hμ#h3<0x1b><0x1b><0x1b><0x1b>///go1<0x13>6L"0â
<0x00><0x00><0x00>cSV<no"lk+ok+oo"hk+i+Ctt00j0w1"L6•É$*•üp'A001É'"70147?{A?â<0x18>
<0x00><0x00><0x00>E~"ie"im-mm-mk-kk+kk+ie"Ne"<0x02>+*HÄvv-yv-ZZV}<0x0f><0x1c><0x00>
UMMGccgMVG}g}<0x9d>~q1E±±±±ZNV<0x11>~<0x00>~<0x0f><0x00><0x00><0x00><0x00><0x00>
00
â0<0x1e><0x14><0x14>D{<0x05><0x00><0x00><0x00><0x18>"(É<0x00><0x00><0x00><0x00>âÉ"1y0|M=
D<0x01><0x00><0x00>EN<0x8d>€<0x0c><0x00><0x00><0x00>ÄZ)..~ÿ~.m0000e'IA#v0ESDD<{~
-
<0x00><0x00><0x00>#<0x81>FEjjEon.hjEojEoj*hn.ifu<0x1a>éxIV<0x12>WW[WW;WW[77;77;<0x1
hü(<0x16>{i>/âüçâ<0x12><0x12>|„It±A"x0+++U*+U*+U*+UEEU*U"É~#YR"læ1i<0x1f><0x16
16jü"}|<0x12>~I<0x00><0x00><0x00>€>EYR45â74â45â75â56âµµ<0x19>c<0x18>'Hällä1äinää8w
<0x00><0x00><0x00><0x00>éIU66æ64ÜhlikhEnliko"<0x1c>00"É<0x1c><0x1c><0x1c>7f|âat7'x
âYKKWaqK8WQ0S0QKQ0SKE0.g0<0x9d><0x9d><0x9d>####"6p<0x8d>Ixü{<0x00><0x00><0x00><0x00><0x00>
4#<0x10>KBC-ÄÄ=IâSSEl<0x00><0x00><0x00><0x10>^<0x1a><0x00>±1$0.±0.€±pzw0pYÜÉ1
```

Typical Work Loop



Pop Quiz!

(yaaay...)

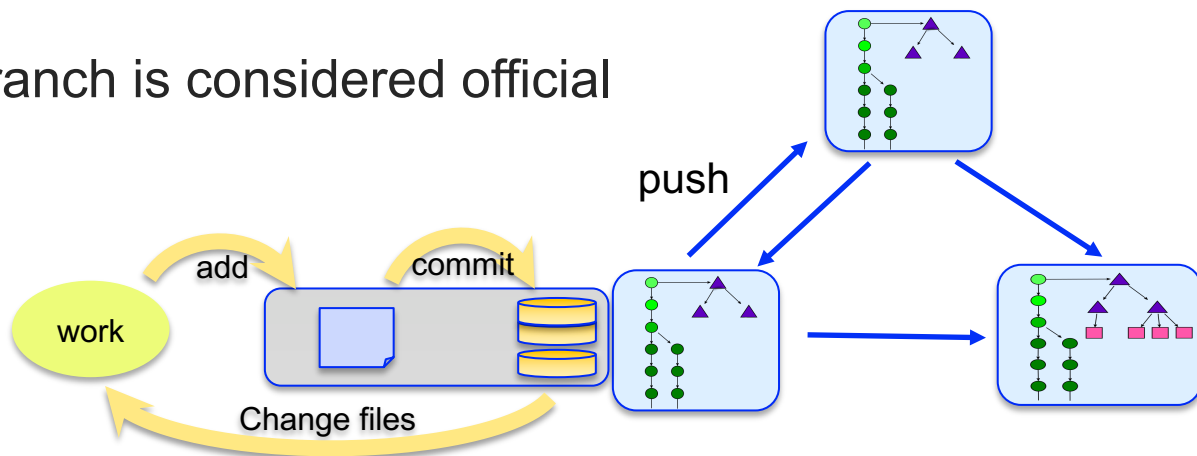
- Go to **socrative.com** > **Student Login**
- Room number: **BBT045**

Exercise 2

- **15 minutes**

Collaborating

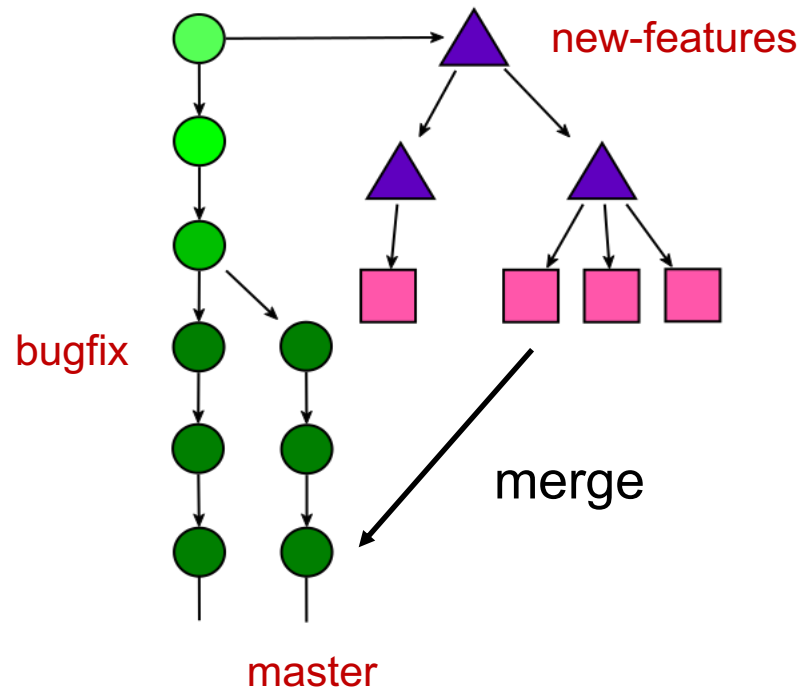
- Convention:
one repo and one branch is considered official



- 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

- source code
- a list of package versions
- parameter values *

You must share

- 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!

