Git crash course

A quick introduction to Git versioning

Jessen V. Bredeson
2017-10-16

Why is version control (software) important?

1. **It's better programming**
    1. Promotes self-documentation
    2. Allows previous version retrieval
    3. Organized and transparent collaboration
    4. Easily distribute changes/updates
    5. Robust to data loss/corruption

2. **It's better science**
    1. Encourages open source
    2. Others can review your methods/code
    3. Promotes reproducibility/replicability

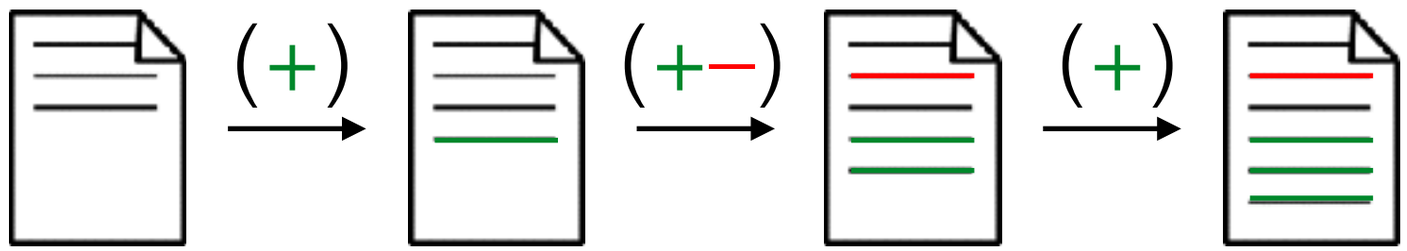
More uses for version control

Track changes in your:

1. Config files
2. Your analysis/project "notebook"
3. Adapter and primer sequences, etc.
4. Gene/genome sequences and annotations
5. Any manually curated files
6. Key results files
7. Paper figures
8. Manuscripts

How does it work?

Git tracks whole files and their (not always) linear history (branching)



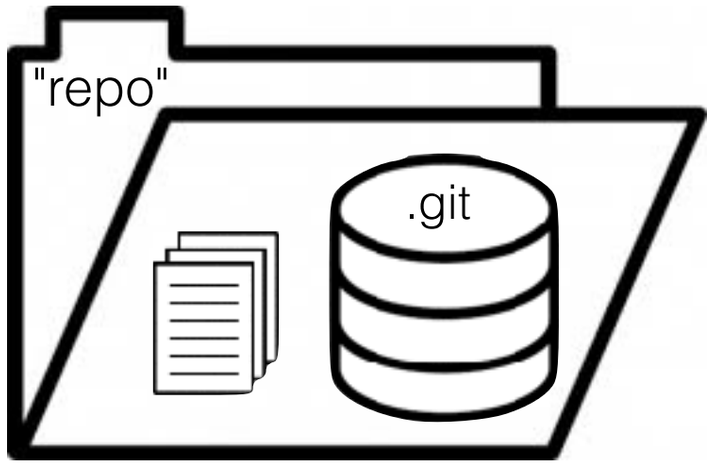
All changes are stored, forever*
(you can go back later and retrieve them at a later date, if necessary).

*You *can* delete code/changes from the repo

Create empty code repository

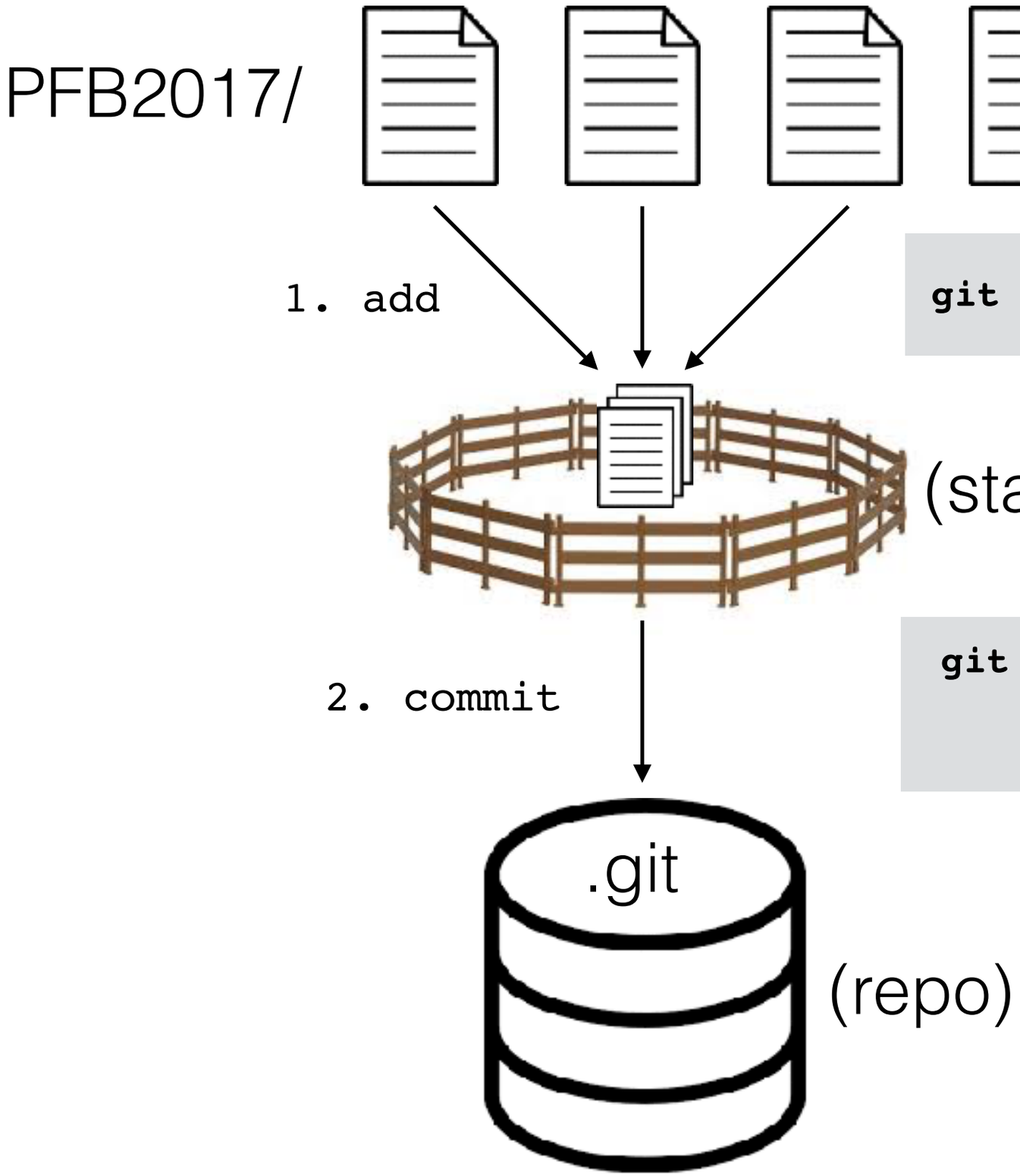
```
> git init repo
Initializing empty Git repository in /path/to/repo/.git/

> cd ./repo
> pwd; ls -AF
/path/to/repo/
.git/
```




```
> git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track")
```

Add and commit changes to the repository




```
git add file1 file2 file3 ...
```

```
git commit -m "Apply this \
        descriptive message to \
        the commit log"
```

Add (stage) changes to the staging area

```
> git status
# On branch master
#
# Initial commit
#
# Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#
#  correct.py
nothing added to commit but untracked files present (use "git add" to track)
```

```
> git add correct.py
> git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#  (use "git rm --cached <file>..." to unstage)
#
#   new file:   correct.py
#
```

Remove changes from the staging area

Remove add'd file from staging

```
> git reset HEAD ./correct.py
Unstaged changes after reset:
M       correct.py
```

Commit changes to the repository

Basic commit

```
> git commit -m 'Initial correct.py'
[master (root-commit) 504bf54] Initial correct.py
 1 files changed, 75 insertions(+), 0 deletions(-)
 create mode 100755 correct.py
```

Stage (add) and commit simultaneously (only currently tracked files)

```
> git commit -a -m 'Cleaning up cruft and adding helpful comments'
[master 4007b53] Cleaning up cruft and adding helpful comments
 1 files changed, 1 insertions(+), 2 deletions(-)
```

Don’t be afraid to commit

Change your previous commit message (to something more descriptive!)

```
> git log --oneline
9cbb6243792eb0d63966d68ffa69b4c43dea3a6f fixed bug
ac4a801319e50d9e3d3c2bcbad1ee618b9d37493 Adding bug
4007b5328f5eb3cfe1acb636c11ca1530ebbae75 Cleaning up cruft and adding helpful ...
504bf54b8745c9baae3de9718f3b20bda3ec224f Initial correct.py

> git commit --amend -m 'Fixed bug which caused a failure to parse the VCF GT'
[master 11902d9] Fixed bug which caused a failure to parse the VCF GT field
 1 files changed, 1 insertions(+), 1 deletions(-)

> git log --oneline
11902d925fd501573e5ee76759e55182dd3120ef Fixed bug which caused a failure to ...
ac4a801319e50d9e3d3c2bcbad1ee618b9d37493 Adding bug
4007b5328f5eb3cfe1acb636c11ca1530ebbae75 Cleaning up cruft and adding helpful ...
504bf54b8745c9baae3de9718f3b20bda3ec224f Initial correct.py
```

Add changes to previous commit

```
> git commit -m 'Adding code to shake the bugs out'

> git add forgotten_file

> git commit --amend
```

Examine differences between repo and changed file(s)

```
> git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   correct.py
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
> git diff correct.py
diff --git a/correct.py b/correct.py
index 7337575..aa00e56 100755
--- a/correct.py
+++ b/correct.py
@@ -23,12 +23,17 @@ def main(argv):
-   Deleted this line
-   Deleted this line, too
+   Added this line to replace the two above
```

Examine differences between repo and changed file(s)

```
> git log --oneline
4007b532..bae75 Cleaning up cruft and adding helpful comments
504bf54b..c224f Initial correct.py
```

Misc. useful actions

```
> git mv <sourcefile> <destfile>  # like unix mv: move or rename a file
```

```
> git rm <file>  # like unix rm: remove a file from repo file tree and index
```

The commands above will automatically stage the changes and require you to commit them

```
> git grep <pattern> <file> # like unix grep; find a pattern in repo index
```

```
> git help <command> # see the man page for a Git command
```

Oops! I created a bug… how do I fix it?

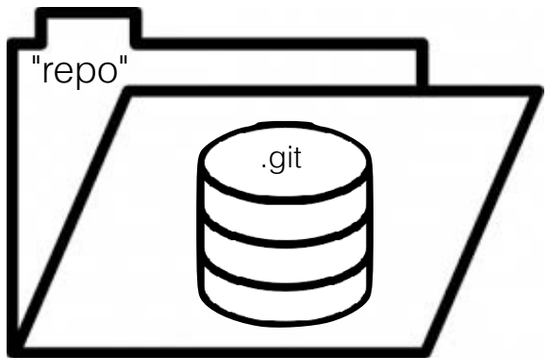
If your changes created a silent bug or were not yet committed,
roll back to the latest commit

```
> git log --oneline
4007b532bae75 Cleaning up cruft and adding helpful comments
504bf54bc224f Initial correct.py

> git checkout correct.py
```

Remote repositories

Free (do not have to pay)!



Some have:

- continuous integration
- Wikis
- JIRA

Download a remote repository for the first time:

```
> git clone url-to-remote-repo
```

Download changes and update from a remote repository:

```
> git pull origin master
```

Upload changes to a remote repository (to which you have permission):

```
> git push origin master
```

Remote repositories

Upload changes to a remote repository may give you an error:

```
> git push origin master
To https://bredeson@bitbucket.org/bredeson/scripts.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://bredeson@bitbucket.org/bredeson/
scripts.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

The remote repo is ahead of your local repo. To sync all repos, do:

```
> git pull origin master  # to update your local repo
> git push origin master  # then push your changes to remote
```

Remote repositories

If you have made changes to some files and did not commit them, you may receive an error that you are in danger of overwriting your untracked changes. you may "stash" those changes away for later:

```
> git stash
Saved working directory and index state WIP on master: 91f8020 commit message
```

You may view your stashes (and their indices) using the following command:

```
> git stash list  # view your stashes
stash{0}: WIP on master: 91f8020 commit message
```

When you have successfully pulled down your remote changes, un-stash your local untracked changes. You may refer to stashes by their 0-based index:

```
> git stash pop  # un-stashes the first stash (= stash{0}) by default

> git stash pop 2  # un-stash the third stash (with index 2)
```

Marking release versions

Release versions can be marked (tagged) in the version history

```
> git tag  # show the current tags
v1.0

> git tag -a v1.1  # add a tag called 'v1.1' to the repo

> git tag
v1.0
v1.1
```

Deleting tags

```
> git tag -d v1.1
Deleted tag 'v1.1' (was 1fd9f72)
```

Sharing tags

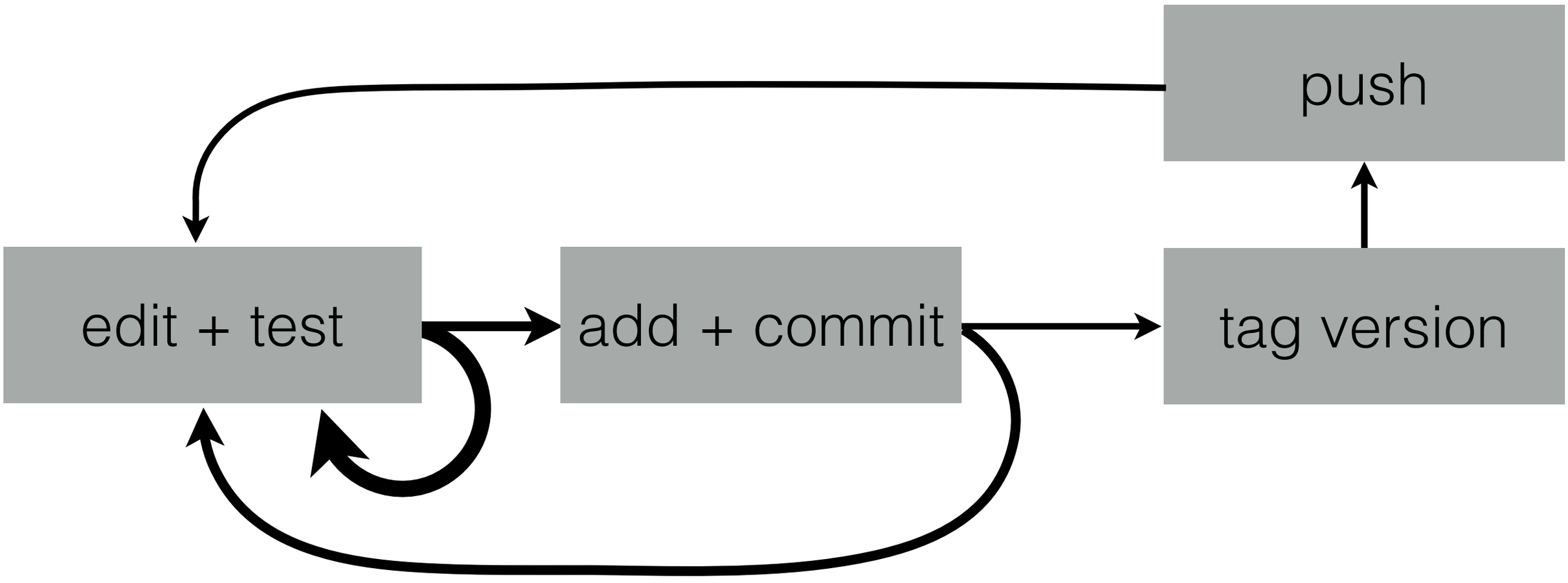
```
> git push origin v1.1  # pushes 'v1.1' tag to remote

> git push origin --tags  # pushes all tags to remote
```

Should I ever re-use tag names or delete old tags?

–NO, if you need to re-use a tag name, create a new one (e.g. `v1.1-1`)

Programming "good" practices workflow



Resources

- Official manual (HTTP and PDF): https://git-scm.com/book/en/v2
- Reproducibility/replicability in science:
  http://ivory.idyll.org/blog/replication-i.html
  http://dx.doi.org/10.1371/journal.pcbi.1003285