CS207: Systems Development for Computational Science

https://harvard-iacs.github.io/2019-CS207/lectures/lecture3/

David Sondak

Harvard University
Institute for Applied Computational Science

9/12/2019

Version Control

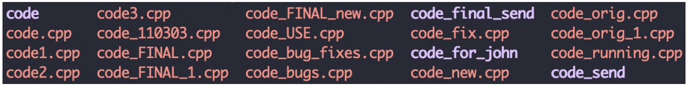
- Minimum guidlines — Actually using version control is the first step
- Ideal usage:
  - Put **everything** under version control
  - Consider putting parts of your home directory under version control
  - Use a consistent project structure and naming convention
  - Commit often and in logical chunks
  - Write meaningful commit messages
  - Do all file operations in the version control system
  - Set up change notifications if working with multiple people

Source Control and Versioning

- Why bother?
- Codes evolve over time
    - Sometimes bugs creep in (by you or others)
    - Sometimes the old way was right
    - Sometimes it's nice to look back at the evolution

Version control is a non-negotiable component of any project.

Why?



```
code        code3.cpp        code_FINAL_new.cpp  code_final_send  code_orig.cpp
code.cpp    code_110303.cpp  code_USE.cpp        code_fix.cpp     code_orig_1.cpp
code1.cpp   code_FINAL.cpp   code_bug_fixes.cpp  code_for_john    code_running.cpp
code2.cpp   code_FINAL_1.cpp code_bugs.cpp       code_new.cpp     code_send
```

Reproducibility   Maintainability   **Project longevity**

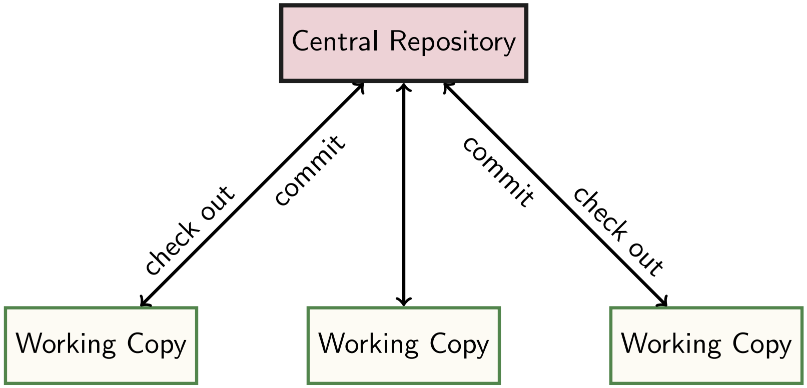
Examples of Version Control

$$\left.\begin{array}{l}\bullet\ \text{Mercurial}\\[2em] \bullet\ \text{Git}\end{array}\right\}\ \text{Distributed Version Control}$$

$$\left.\begin{array}{l}\bullet\ \text{Concurrent Versions System (CVS)}\\[2em] \bullet\ \text{Apache Subversion (SVN)}\end{array}\right\}\ \text{Centralized Version Control}$$

$$\left.\begin{array}{l}\bullet\ \text{GoogleDrive}\\[2em] \bullet\ \text{Dropbox}\end{array}\right\}\ \text{Don't use these for software}$$

Centralized Version Control



Comments on Centralized Source Control

- A central repository holds the files in both of the following models
  - This means a specific computer is required with some disk space
  - It should be backed up!

1. Read-only Local Workspaces and Locks
   - Every developer has a read-only local copy of the source files
   - Individual files are checked-out as needed and locked in the repo in order to gain write access
   - Unlocking the file commits the changes to the repo and makes the file read-only again

2. Read / Write Local Workspaces and Merging
   - Every developer has a local copy of the source files
   - Everybody can read and write files in their local copy
   - Conflicts between simultaneous edits handled with merging algorithms or manually when files are synced against the repo or committed to it
   - CVS and Subversion behave this way

CVS — Concurrent Versions System

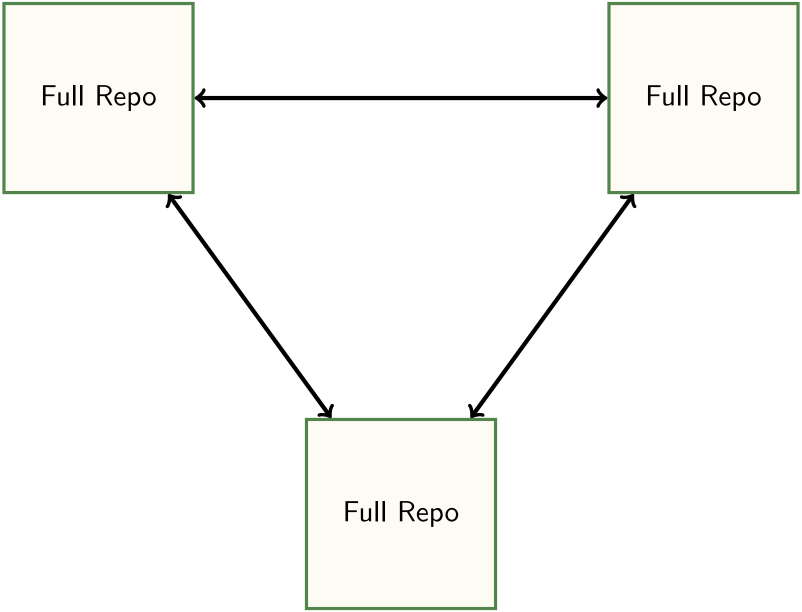
- Started with some shell scripts in 1986
- Recoded in 1989
- Evolving ever since (mostly unchanging now)
- Uses read / write local workspaces and merging
- Only stores differences between versions
    - Saves space
    - Basically uses `diff(1)` and `diff3(1)`
- Works with local repositories or over the network with rsh / ssh

Subversion

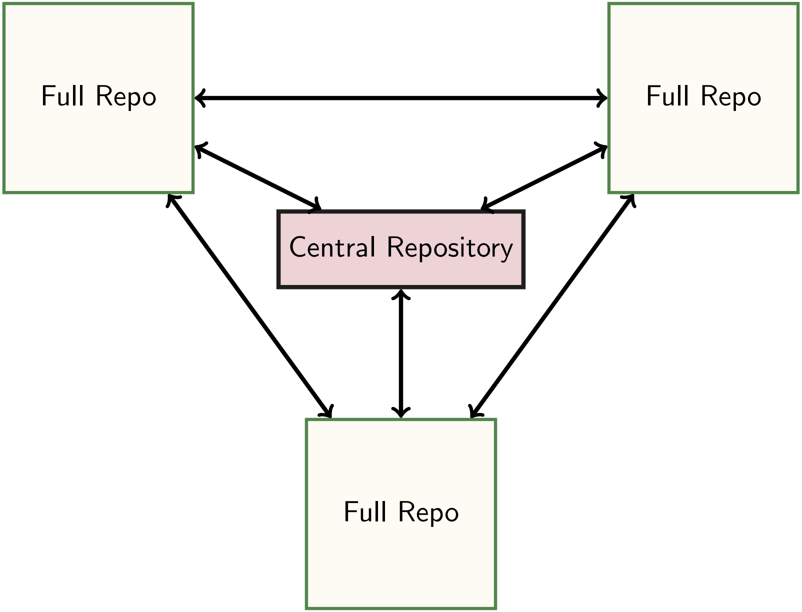
Subversion is a functional superset of CVS (if you learned CVS previously, you can also function in Subversion)

- Began initial development in 2000 as a replacement for CVS
- Also interacts with local copies
- Includes directory versioning (rename and moves)
- Truly atomic commits
  - i.e. interrupted commit operations do not cause repository inconsistency or corruption
- File meta-data
- True client-server model
- Cross-platform, open-source

Distributed Version Control



Distributed Version Control



Getting Started with Git

There are **many** Git tutorials:

- https://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide
- https://bitbucket.org/
- https://github.com/
- ⋮
- Others on the course Resources page

Git was created by Linus Torvalds for work on the Linux kernal $\sim$ 2005

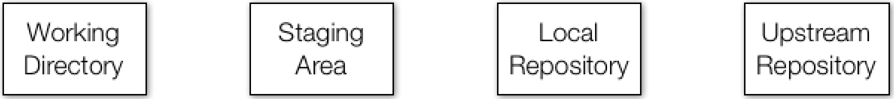
Git is ...

- A **Distributed** Version Control system or
- A **Directory** Content Management System or
- A **Tree** history storage system

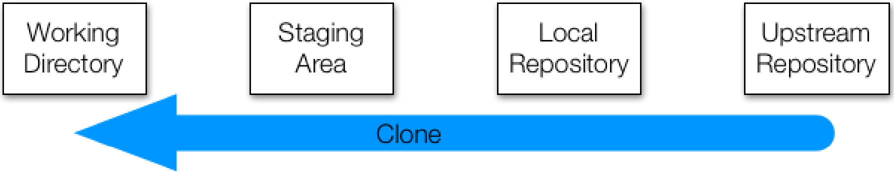
Distributed

- Everyone has the complete history
- Everything is done offline
- No central authority
- Changes can be shared without a server

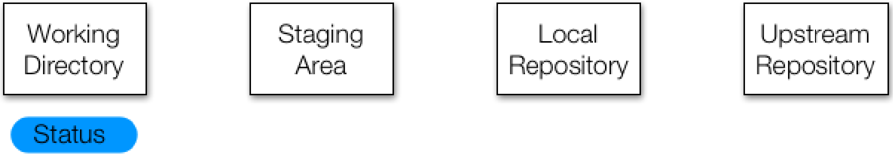
The Bare Essentials of `git`



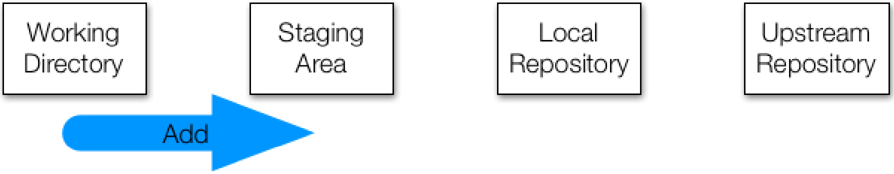
The Bare Essentials of `git`



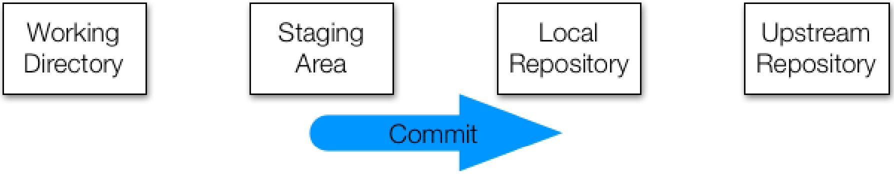
The Bare Essentials of `git`



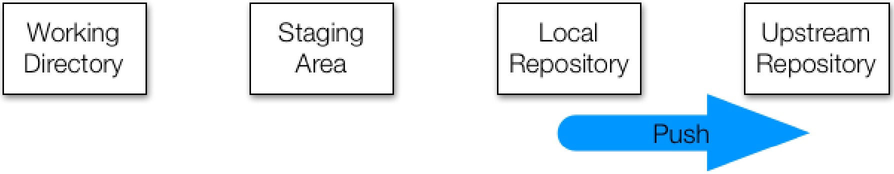
The Bare Essentials of `git`



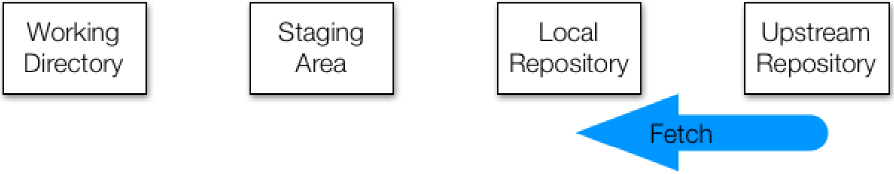
The Bare Essentials of `git`



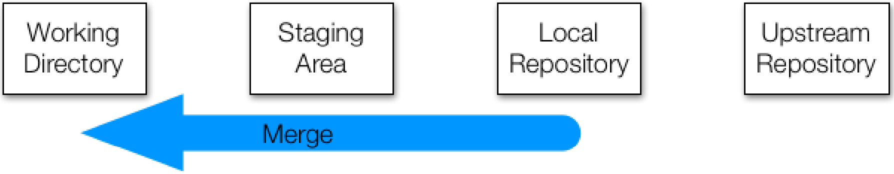
The Bare Essentials of `git`



The Bare Essentials of `git`



The Bare Essentials of `git`



When to Commit?

- Committing too often may leave the repo in a state where the current version doesn't compile.
- Committing too infrequently means that collaborators are waiting for your important changes, bug fixes, etc. to show up.
  - Makes conflicts much more likely
- Common policies:
  - Committed files must compile and link
  - Committed files must pass some minimal regression test(s)
- Come to some agreement with your collaborators about the state of the repo