CS207: Systems Development for Computational Science

Version Control: Big ideas and `Git`

Instructor: David Sondak
TFs: Charles Liu, Eric Wu and Kevin Wu

Harvard University
Institute for Applied Computational Science

September 6, 2017

Today's Plan

A large portion of today's lecture is based upon a version control lecture given by Dr. Chris Simmons at UT-Austin in his course *Tools and Techniques of Computational Science* co-developed by Dr. Karl Schulz.

The topics for today are:

- Lecture workflow
- Version Control — Big picture
- Types of version control
- `Git`

Lecture Workflow

At the beginning of each lecture, do the following:

1. Open a terminal
2. `cd cs207_firstname_lastname`
3. `git pull upstream master`
4. `git add .`
5. `git commit -m "Updated cs207 notes for Lecture n"`
6. `git push`

We need to talk about all of this ($\Uparrow$): why we do it and what it means.

The Multiple Roles of a Computational Scientist

- Computational scientists are expected to be:
  - Physicist — addressed in curriculum
  - Numerical analyst — addressed in curriculum
  - Software developer — not usually addressed
- Software engineering is vital
  - Understandable code

Software engineering gap

Software engineering is not part of the formal curriculum and therefore most scientific software developed in academia is unverified. Furthermore, the software developed is unable to ever be verified without significant time investment.

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
- How can you get back to an old version?
  - Keep a copy of every version of every file
    - Disk is cheap, but this could get out of hand quickly
    - **Huge** pain to maintain
  - Use a tool

Some Example Tools

- Free
  - RCS — Revision Control System
  - CVS — Concurrent Versions System
  - SVN — Subversion

  } Centralized

  - Git — https://git-scm.com/
  - Mercurial — https://www.mercurial-scm.org/

  } Distributed

- Commercial
  - MS Visual Studio Team System
  - IBM Rational Software: Clearcase
  - AccuRev
  - MKS Integrity

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 Subverions 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

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

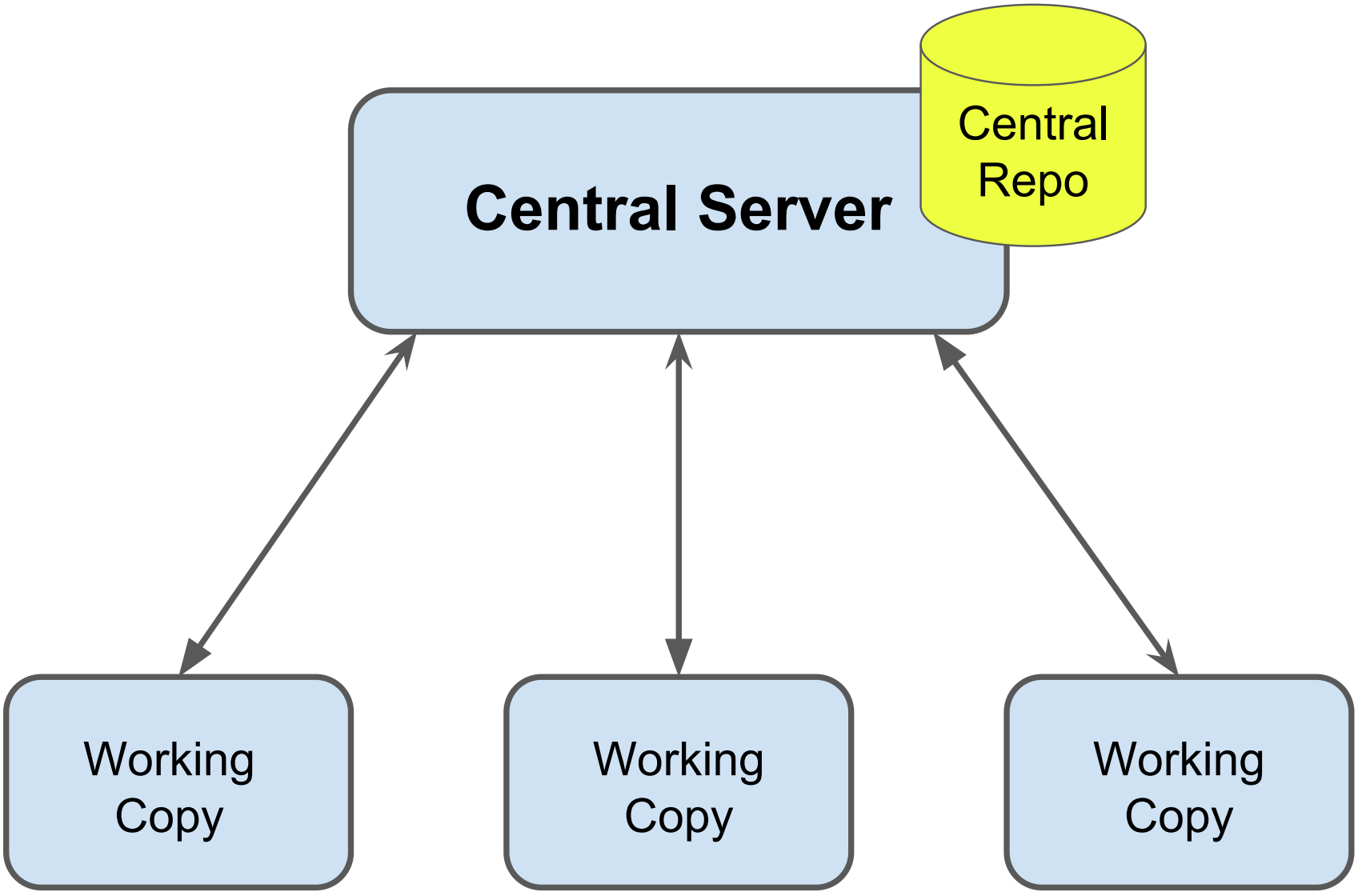
Git is ...

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

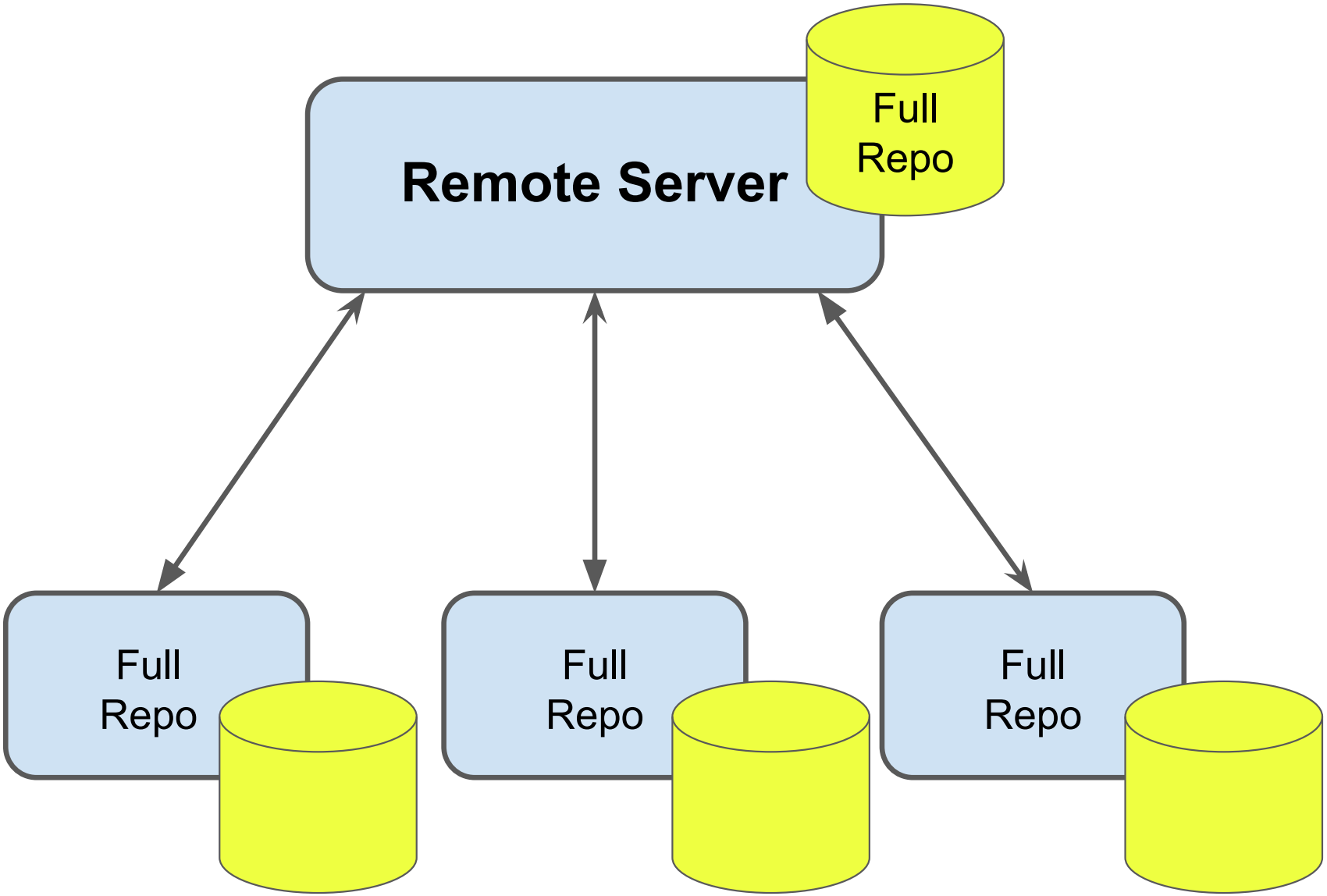
Distributed

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

Centralized VC vs. Distributed VC



Centralized VC vs. Distributed VC



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