A lesson on Git for the OpenSim project

Agenda

1. Quick tour of GitHub.
2. Why are we switching from SVN/SimTK to Git/GitHub?
3. How is Git different from SVN?
4. How does Git store your history?
5. Install git on your computer.
6. Tutorial A: **using** the Simbody Git repository.
    1. TortoiseGit
    2. command-line git
7. Tutorial B: **contributing** to a repository.
    1. TortoiseGit
    2. command-line git
8. Managing your own project.
9. Review of key concepts.

NOT covering: how to be the maintainer of a git project.

Why are we switching from SVN/SimTK to Git/GitHub?

1. **SVN** and **SimTK**:

   a. simple (central repository, fewer commands, simpler concepts).

   b. discourages code review.

   c. branching has high inertia, discourages experimenting.

   d. difficult for outsiders to contribute to the project (open source?).

2. **Git**:

   a. easy to branch code (encourages parallel development, experimenting).

   b. allows use of GitHub.

   c. complicated.

   d. same number of letters as 'SVN'.

3. **GitHub**:

   a. easy to take someone's code and modify it for your own purposes (fork).

   b. friction-less way to tell developers about bugs (issues).

   c. easy to contribute to projects of which you're not a member (pull request).

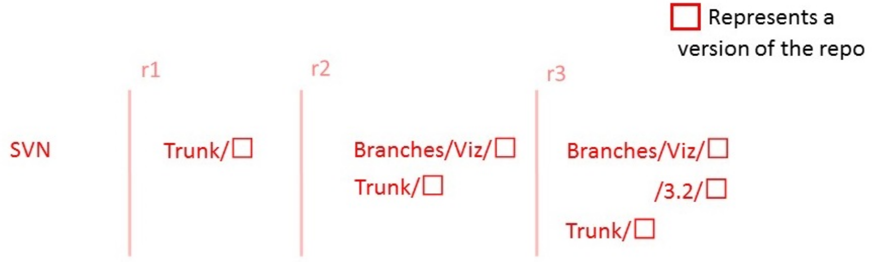
   d. job opportunitites (employers ask for your GitHub username).

SVN branches vs git branches

SVN

- Branches are copies of the code in a different part of the repository.
- Branches are made infrequently, often linger and are not deleted, even after they are no longer used.

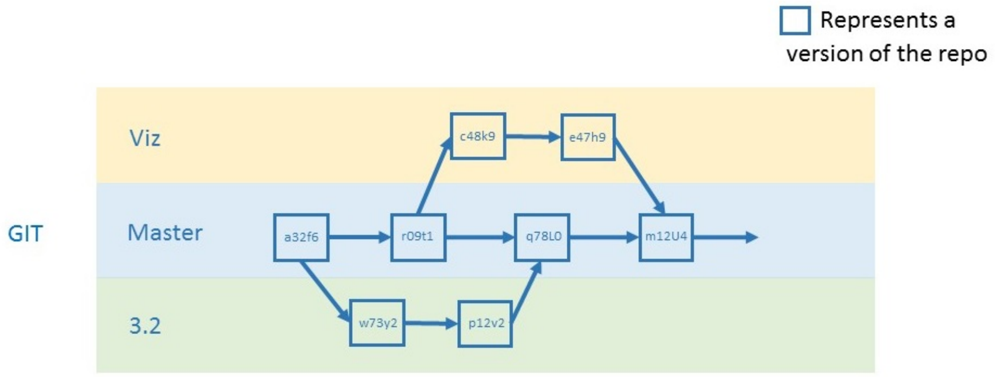
1. Create release branch, develop in parallel with Trunk (unstable)
2. When release is done, merge part of release branch back into Trunk.



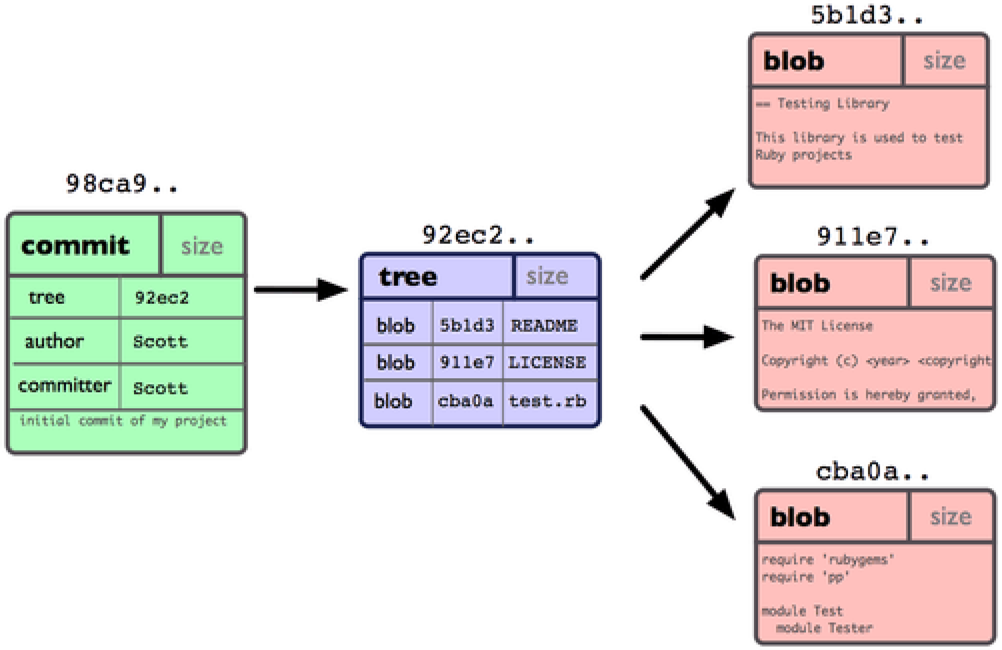
GIT

- Branches are just labels to commits (git's revision).
- Branches are made frequently, and are meant to be deleted. All work is done in branches other than master.

1. To implement a feature, make feature branch ('new-component-interface').
2. When done with feature, merge back into master ("Trunk").
3. master is always stable; unstable code is in feature branches.

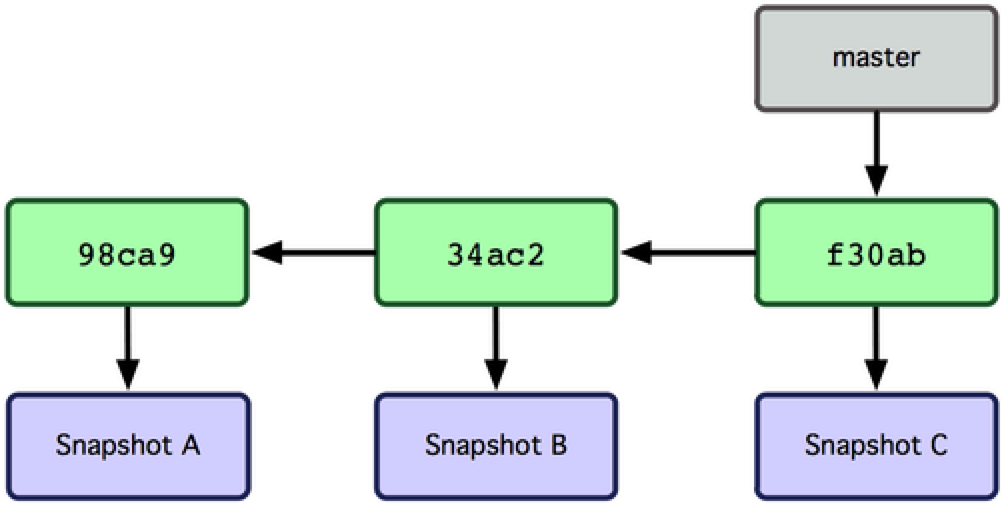


A commit contains metadata and the tree of the files at that point in history.



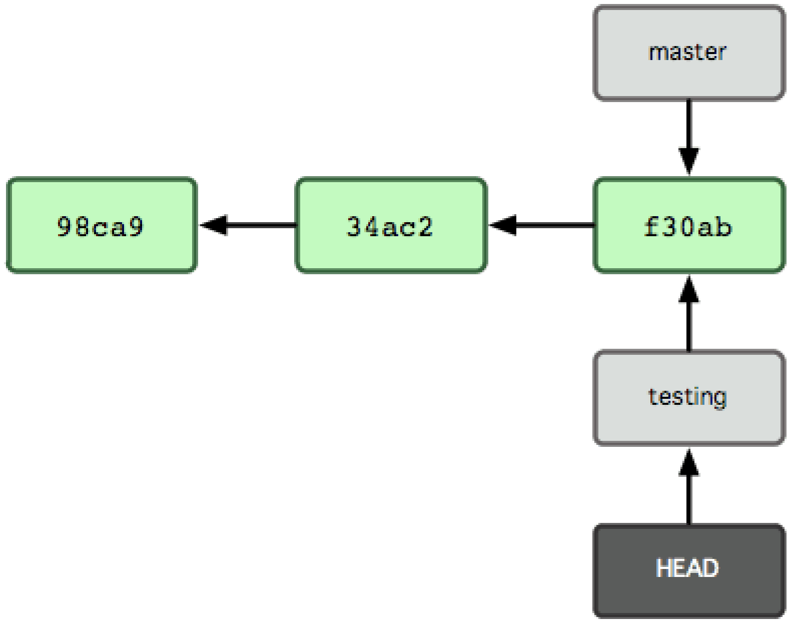
The history is a graph of commits.

Branches, such as `master`, are just pointers to a particular commit.

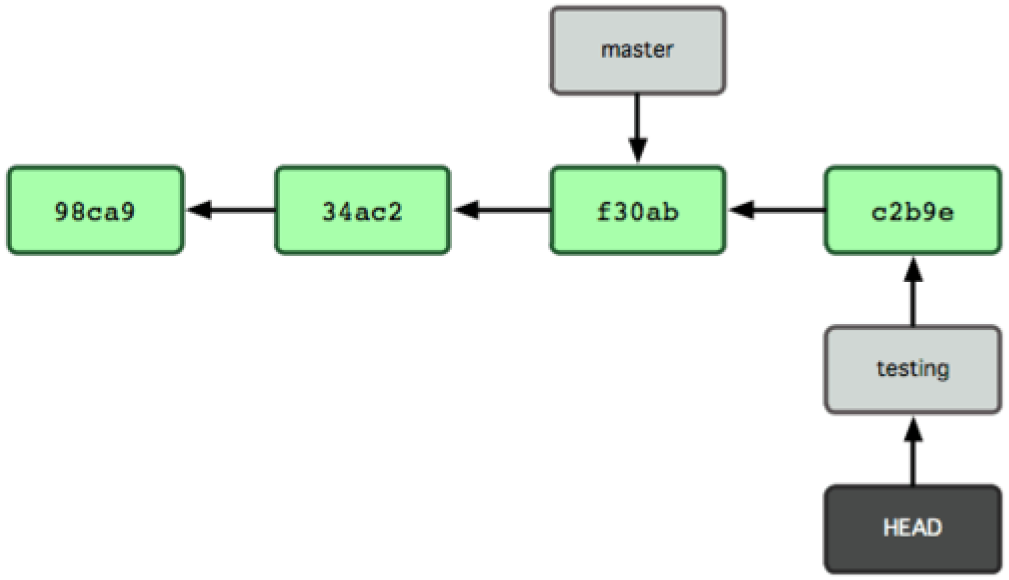


HEAD refers to the branch/commit currently checked out.

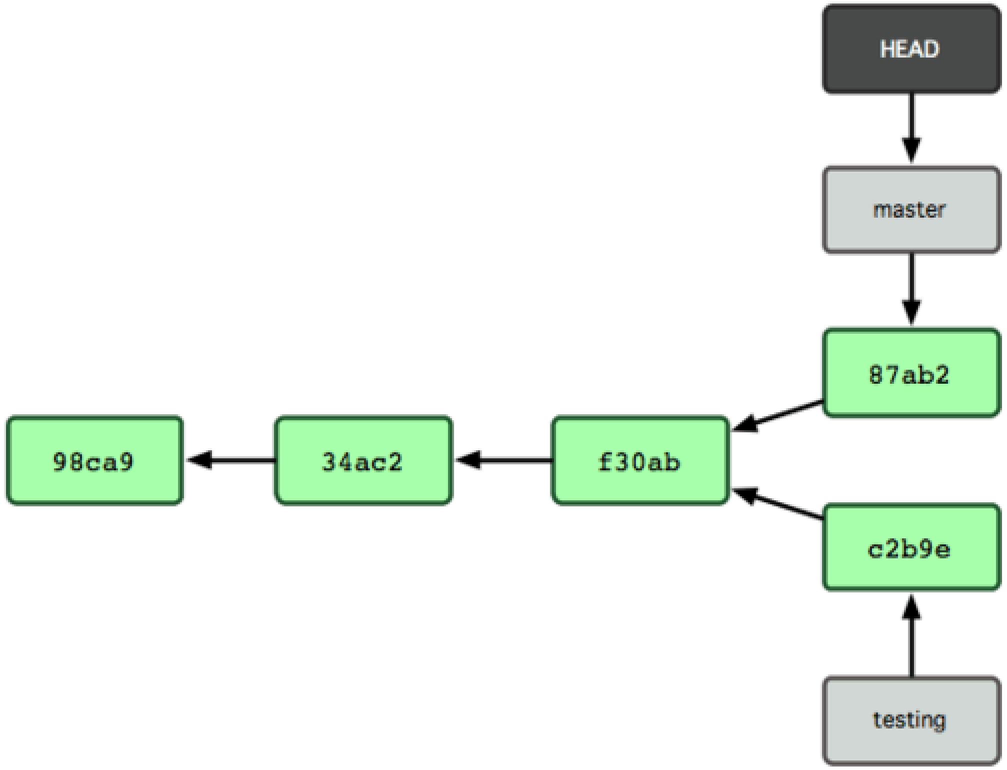
Here, we've created a new branch `testing`, that points to the same commit as does `master`.



This is what the history looks like when we commit on the testing branch.



This is what the history looks like if the branches have diverged.



Install git

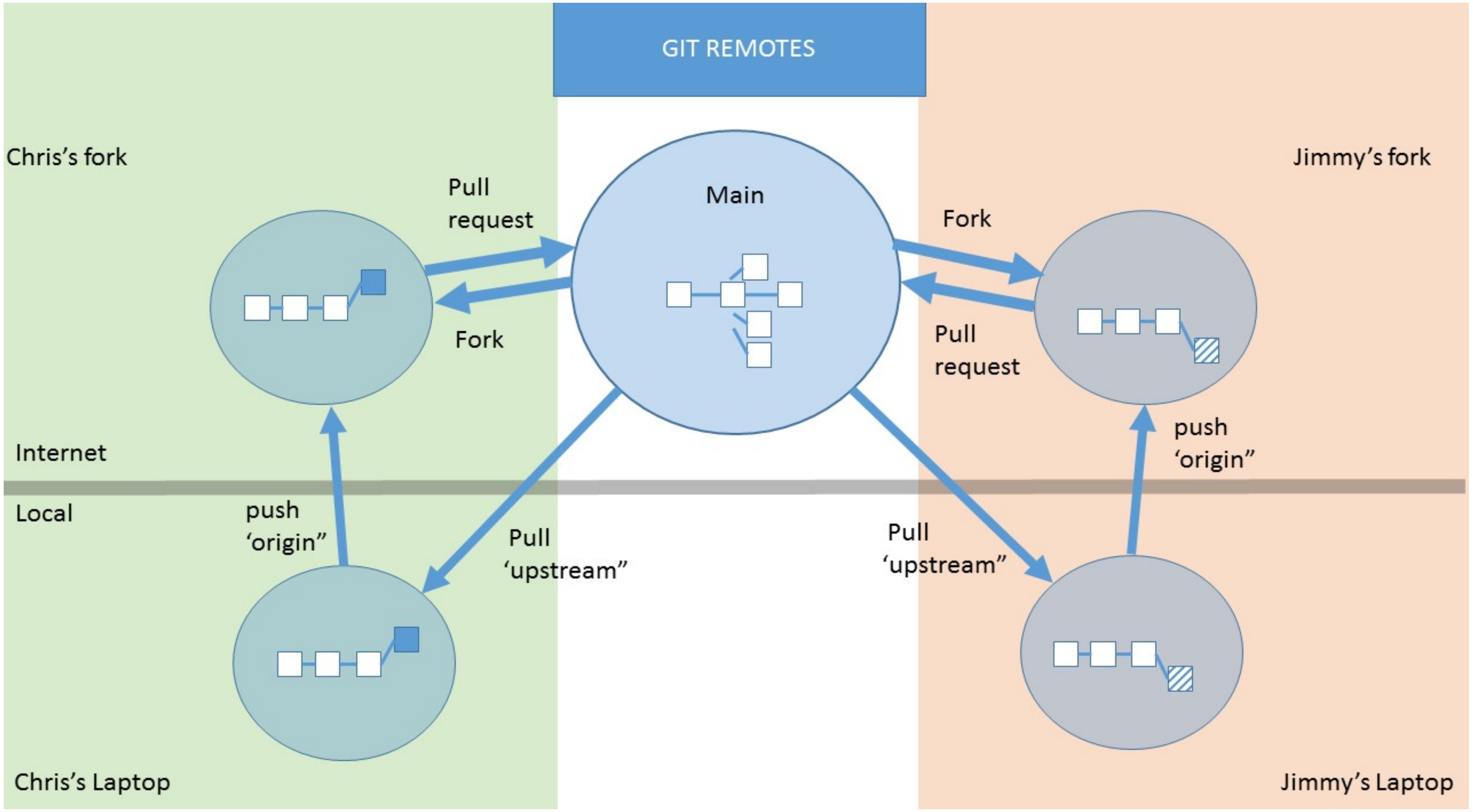
1. Install msysGit (git for windows)
    1. Use Git Bash only (cannot access git from regular Command Window).
    2. Checkout Windows-style, commit Unix-style line endings.
2. Install TortoiseGit.
3. Configure GIT on your computer.
    1. `git config --global user.name "Christopher Dembia"`
    2. `git config --global user.email "cld72@cornell.edu"`
    3. SSH...not now.
4. Create an account on GitHub.
5. Get added to opensim-org.
6. Explore the opensim-org page.

Consuming code on GitHub: Simbody

1. Clone the simbody/simbody repository to your computer.
2. Explore with TortoiseGit: clone, log, remotes, branches, tags, checkout, revision graph.
3. Explore with command-line git:

```
git clone https://github.com/simbody/simbody.git
git remote -v # -v for verbose
git branch
git tag
git status
git log
git checkout Simbody-3.4.1 # this is a tag.
```

Git is a distributed VCS: remotes



Contributing: dummy-opensim-core (slide 1)

1. GitHub fork (copy from one place on GitHub to another place on GitHub).
2. Clone your fork (copy from GitHub to your machine).

   ```
   git clone https://github.com/myname/dummy-opensim-core.git
   ```

3. Add the upstream repository as a remote on your computer.

   ```
   git remote add upstream https://github.com/opensim-org/dummy-opensim-core.git
   ```

4. Create a new branch.

   ```
   git checkout -b myfeature
   ```

5. Make some changes.

   1. Create a new file.
   2. Add the new file.

      ```
      git add mynewfile.txt
      ```

   3. Mess around!
   4. What's the state of the working copy?

      ```
      git status
      ```

6. Commit changes.

   ```
   git commit -am"My message." # -a to commit all tracked changes.
   git commit -m"my message."
   git commit
   ```

Contributing: dummy-opensim-core (slide 2)

1. Push your changes to your fork.

   ```
   git push origin myfeature
   ```

2. Create a pull request (PR) on GitHub.
3. Wait for your PR to be accepted.
4. Update your local repository and delete your feature branch.

   ```
   git checkout master
   git pull upstream master
   git branch -d myfeature # delete LOCAL feature branch
   ```

Contributing: dummy-opensim-core (slide 3)

1. Deal with upstream changes (that occured while you worked on feature).

   a. Repeat previous two slides.

   ```
   git checkout -b feature2
   (mess around; edit the top few lines of CMakeLists.txt)
   git commit -am"Destroyed OpenSim."
   git push origin feature2
   (submit pull request)
   ```

   b. Rebase.

   ```
   git checkout master
   git pull upstream master
   git checkout myfeature
   git rebase master
   git push origin :feature2 (deletes branch on GitHub)
   git push origin feature2
   (pull request can now be accepted)
   ```

Developing on a project for which you're a member (also: personal projects).

- Don't use a fork.
- For SMALL changes, commmit directly to master.
- Submit pull requests WITHIN the same user/organization (on GitHub).

Review of git/GitHub terms and commands

Terms

- HEAD
- master
- origin
- upstream

Commands (what's the equivalent in SVN?)

- git clone
- git status
- git commit
- git add
- git checkout
- git merge
- git rebase
- git reset
- git branch
- git tag
- git remote [add]
- git cherry-pick
- git rm [--cached]
- git init
- git fork
- git pull-request

Culture

1. Don't work directly on the master branch.
2. Commit often, and atomically (code should compile at each commit).

3. Commit messages

   b. First line should be brief; 50 characters.

   c. Skip a line before your detailed description.

Online resources to learn about git

- The git book. Really well-written. If you want to understand what's going on, look here. http://git-scm.com/book
- Interactive way to learn the basics of command-line git. https://try.github.io/levels/1/challenges/1
- Pretty thorough description of different git commands. http://gitref.org/

Sherm's links:

- Nice overview of git internals: http://nfarina.com/post/9868516270/git-is-simpler
- Git for computer scientists: http://eagain.net/articles/git-for-computer-scientists/
- Git crash course: http://git-scm.com/book/en/Getting-Started-Git-Basics
- What is a branch: http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is
- Branching workflow discussions:
    - Basics: http://git-scm.com/book/en/Git-Branching-Branching-Workflows
    - Chris D recommends this: http://nvie.com/posts/a-successful-git-branching-model/
    - Different workflows: https://www.atlassian.com/git/workflows
- GitHub's teaching resources: http://teach.github.com/

List of git cheat sheets:

- https://na1.salesforce.com/help/pdfs/en/salesforce_git_developer_cheatsheet.pdf the one I'd use. describes a common workflow.
- http://ndpsoftware.com/git-cheatsheet.html very interactive. shows workspace, index, local repo, remote repo.
- https://help.github.com/articles/git-cheatsheet an annotated list of commands
- http://www.git-tower.com/blog/git-cheat-sheet/ very pretty; includes rebase
- https://www.atlassian.com/dms/wac/images/landing/git/atlassian_git_cheatsheet.pdf organized by commands
- http://web.archive.org/web/20090419122050/swxruby.org/git-cheat-sheet.pdf tries to describe sequences of commands, and describes some more advanced commands. not very pretty