Introduction to Git

Kate Hedstrom
IMS, UAF

Linus Torvalds


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Version Control Software

- **System for managing source files**
  - **For groups of people working on the same code**
  - **When you need to get back last week's version**
- **In the past, I have used RCS, CVS, and SVN, each better than the last**
- **Git was designed for managing the Linux kernel and therefore has these goals:**
  - Fast
  - Support many, many developers
  - Distributed
  - Open Source

Distributed?

- Every checkout gives you a copy of the whole repository
- Can compare branches, history while offline
- Can check in your changes to your local repository
- Sharing updates with others is optional

Getting Started With Git

- **Set up who you are:**

```
% git config --global user.name "you"
% git config --global user.email   \
  "you@home"
```

- **Get colorful (if you want):**

```
% git config --global color.ui "auto"
```

- **Without "--global" applies to current directory only**

Start a New Repository

- **In the directory with your code:**
  - git config --list
  - git init
  - git add . # all files in current dir
  - git commit -m “initial message”
- **You now have a .git directory with a database of your files**
- **Revision numbers are SHA1 numbers, same for the same content**

From a Repository

- **From a git url:**
  - git clone <url>
- **Could be another local directory:**
  - git clone dir1 dir2
- **From a svn url:**
  - git svn clone <url>
- **Default is to suck down the entire history into the database**

Main git commands

- add – add sources to next commit
- commit – check in changes locally
- checkout – change branches
- push – send your changes to a remote site
- pull/fetch – get changes from remote site
- Status – find out which files would change on commit
- diff – find out what's different between index and current sandbox
- help

Example

- **Change/add some local files**
  - git add newfile
  - git commit
- **"git add" adds files to the commit list (index) for the next commit**
- **Can selectively add only some of your changes to make logical commits, otherwise:**
  - git commit -a #commits all changes

Git example

```
% ls /my/src/cpp
cpp.h  cpp.c   Makefile  ...
% cd /my/src/cpp
% git init
# Tell git which files to track
% git add .
% git commit
[make some changes]
% git commit -a
```

Comments on Previous

- Svn takes more fuss to get going (not shown)
- New files have to be explicitly added
- Any directory can become a git repository
- Better for text files than for binary

xkcd Cartoon

Message advice

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.
Wrap it to about 72 characters or so. The
blank line separating the summary from the
body is needed; tools like rebase can get
Confused if you run the two together.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for
the bullet, preceded by a single space, with
blank lines in between, but conventions vary.
```

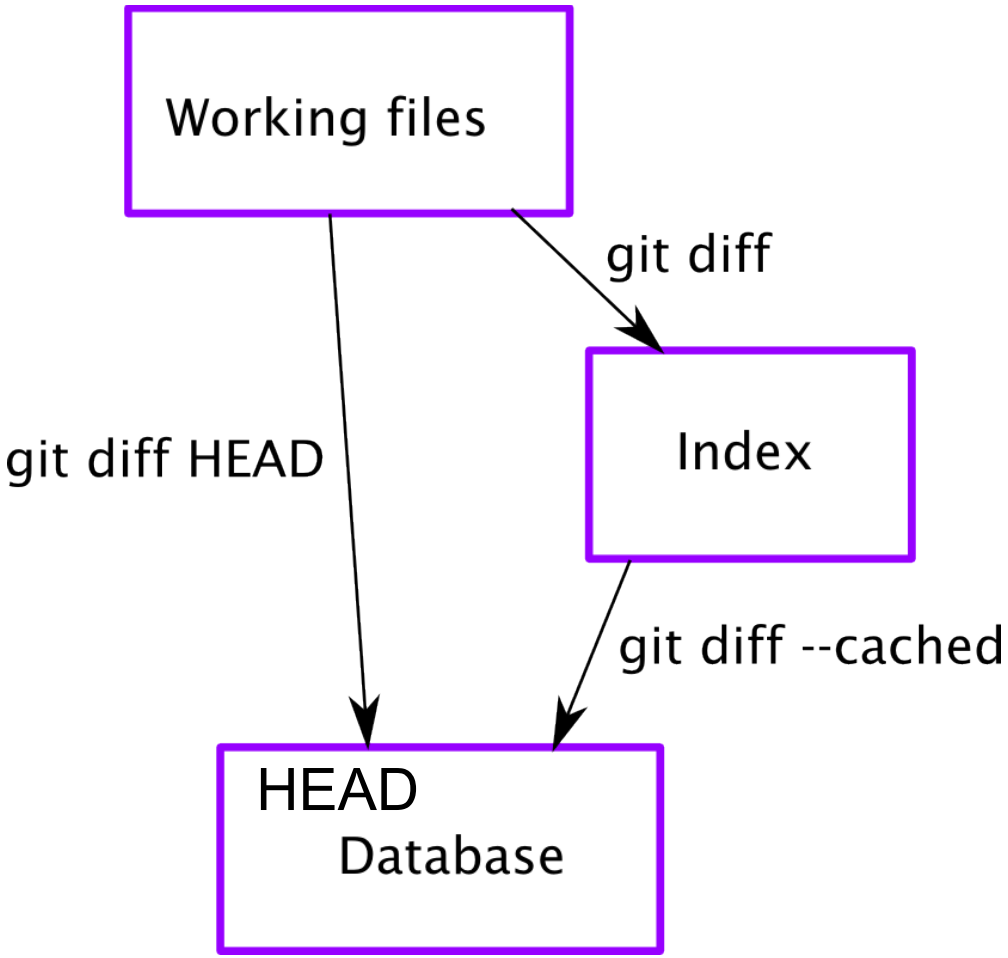
Seeing History

- git log
- gitk (gui)
- git diff HEAD^
- git log HEAD^^^ or HEAD~3
- git diff b324a87 (SHA1)
- git diff --cached (between index and HEAD)

Index?

- The index is a store of what would be checked in on "commit"
- Contains files that merged cleanly and those put in with "git add"
- "git diff" shows difference between index and current sandbox
- "git diff HEAD" shows difference between last checked in and sandbox

Index as Staging Area



Coordination

```
# on pacman
% git clone <URL> roms
% cd roms
[make some changes]
% git commit -a
% git push origin master
```

```
# on this Mac
% git clone …
% cd roms
% git pull …
% make
```

- **Coordinate code on multiple systems**
- **Coordinate between multiple programmers**
- **Can be single version or multiple branches**

Fetch

- Pull is a fetch and a merge
- I use push/pull between my own repositories
- Fetch then merge is better when working with others.

Updates

- **An update when two people have changed things can involve:**
  - No conflict because changes are in different places
  - Conflict because two different changes to the same region in the code
- **If there is a conflict this must be resolved by human intervention**
- **One option is to reset (undo the merge)**

Other git commands

- delete – no longer need a file
- move – rename a file or move to new location
- merge – merge changes from another branch
- cherry-pick – pick one update from some other branch
- remote – register a remote repo
- rebase – reorder the history in your local repo
- stash – add to a stack of rough drafts

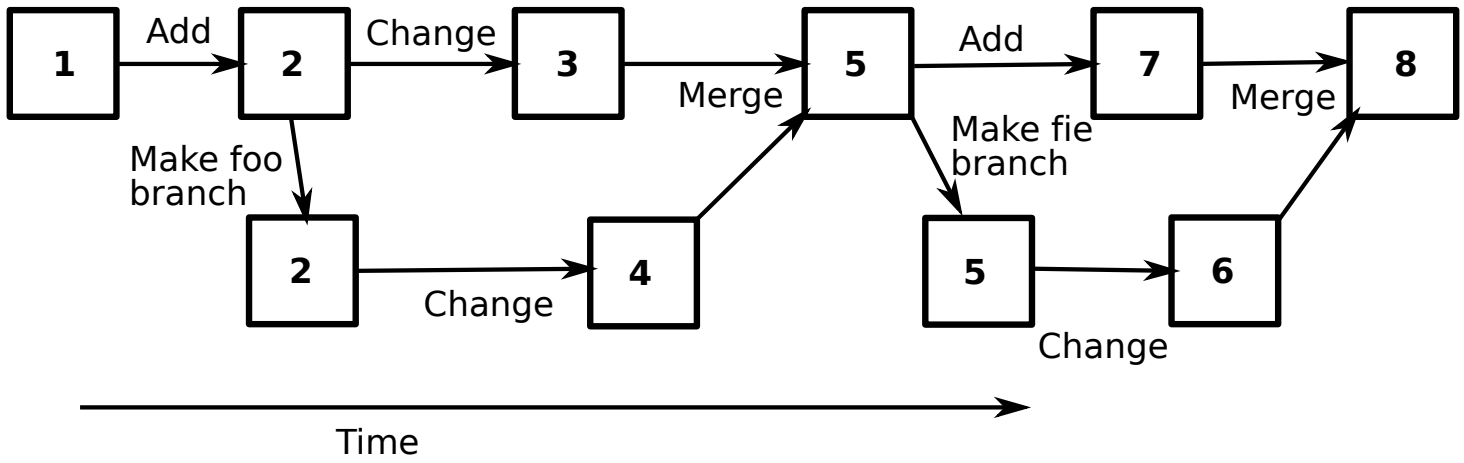
Revision Numbers

- **git uses a database to store the files**
- **Each revision has a unique number to describe that snapshot – it's a SHA1 with 40 characters**
  - SHA1 for each file
  - SHA1 for a tree of files
- **Can see the numbers with "git log"**
- **Every commit creates a new revision number**

What about Branches?

- **See the branches:**
  - git branch
- **See all the branches:**
  - git branch -a
- **Make a new branch:**
  - git branch <new> # copy of current
- **Switch to that new branch:**
  - git checkout <new>

Branches



- **A branch starts as a duplicate**
- **Delete branches after merge and testing**
- **Rebase can be used to put change 7 after 6**

Conflicts

- **If there is a conflict, git will let you know (check git status)**
- **The merge failures will look something like:**

```
Clean code before
<<<<<<< HEAD:<file>
My code
=======
New code
>>>>>>> branch:<file>
Clean code after
```

More Conflicts

- **Once you've cleaned up the mess, tell git you're ready:**

```
git add filename
```

- **Git puts file into the index**
- **You can instead toss the changes with:**

```
git checkout HEAD filename
```

- **Once all the files are clear (check with "git status") commit the index to the repo:**

```
git commit
```

Other

- **Remote repos are seen as a kind of branch (tracking branch)**

```
git fetch pete    # get pete only
git remote update # update all
```

- **Always, always check in changes before a pull/fetch/merge – different from svn.**
- **Check in changes before changing branches – unless you want the change to be on the new branch**

Git Svn Handshaking

- **Not quite as robust as git alone**
- **Based on Perl scripts in svn distribution (not always installed)**
  - git svn clone <url>
  - git svn clone -r 1043 <url>
  - git svn rebase # fetch from upstream
  - git svn dcommit # commit to upstream
  - git svn log

Bare Repositories

- Full database, but no working files
- Create with “git clone --bare url”
- For repositories that get pushed to (web servers, for example)

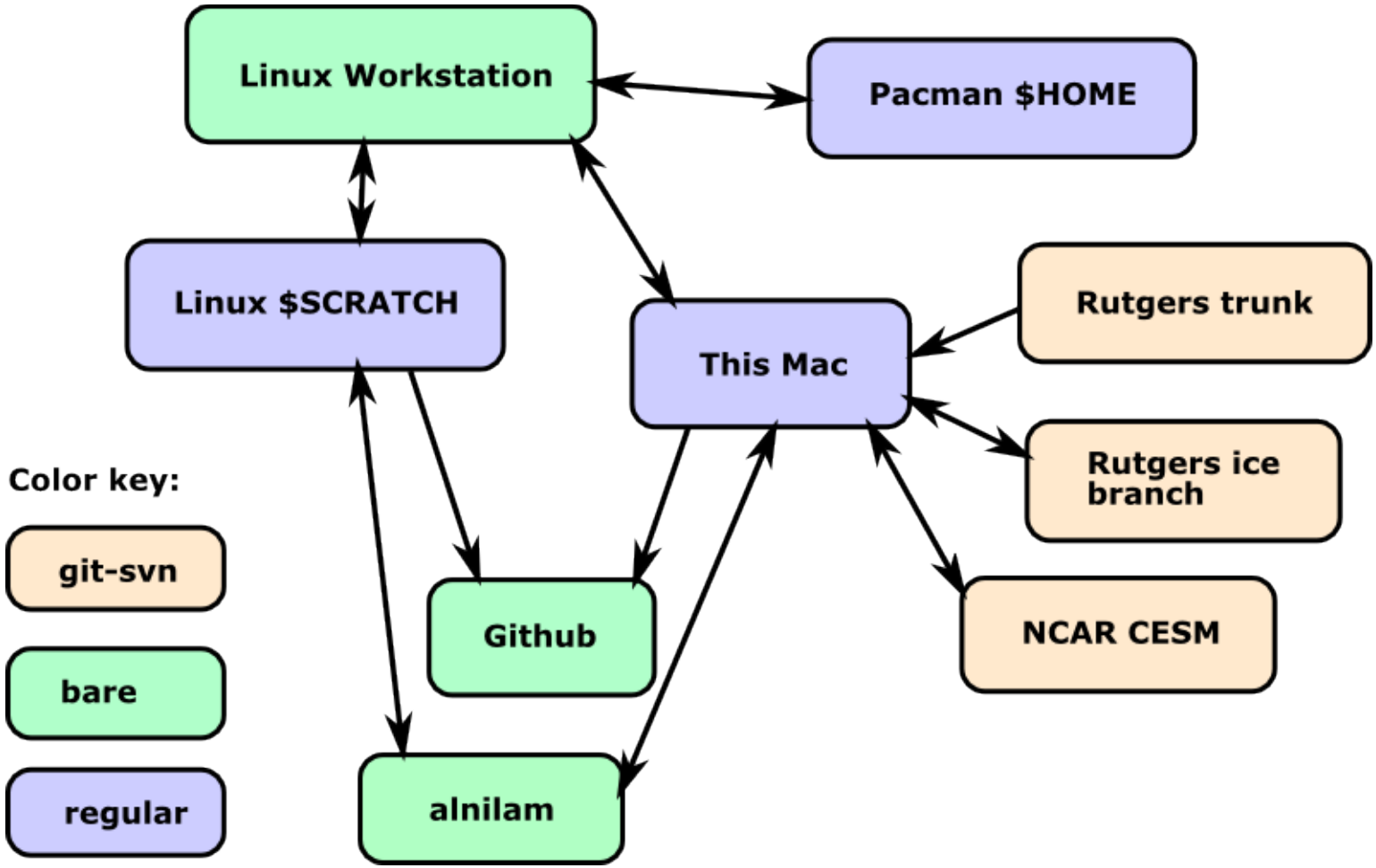
My Branches

- Copy of the svn code (public)
- Fish model branch
- Global model branch (with tripole grid)
- Any other thing I'm working on temporarily

My Insane Repo Collection

- Bare repository on Linux workstation
- Public branch on github
- Cloned to each supercomputer via ssh
- Cloned to colleague's computer via ssh
- git-svn working best on Mac
- Mac has my git-svn repo, plus clone of "origin" repo, also NCAR CESM-ROMS and Hernan's trunk via git-svn

My Insane Repo Collection



Workflows

- **Just a few people with private (bare) repo**
  - Push/fetch at will, communicate however
- **Huge groups (like Linux)**
  - Can have dictator in charge of master branch
  - Tree structure with lieutenants overseeing components
- **Keep main branch clean and test on temporary topic branches**

Random other things

- **Tell git to stop tracking file**
  - git rm --cached <file>
- **Launch gui editor**
  - git difftool
  - git mergetool
- **Set some aliases**
  - git config --global alias.co checkout
  - git config --global alias.br branch

Git Drawbacks?

- Best with one project per repository (roms, plotting, matlab tools all separate entities)
- Yet another tool to learn
- Git-svn doesn't handle svn Externals
- More rope to hang yourself…

Learn more

- **Online at http://git-scm.com/documentation – man pages, a cheat sheet, even a free book**
- **git help**
- **If you like these ideas, but prefer a Python tool, check out Mercurial at: http://mercurial.selenic.com/**

Work Along?

- **git config --global user.name "me"**
- **git config --global user.email "me@work"**
- **git config --global color.ui "auto"**
- **git config --list**
- **Find some code to track… If desperate:**
  - http://www.people.arsc.edu/~kate/simple_codes/

In Code Directory

- git init
- git add . # or git add *.f
- git commit # -m “message”
- make a change….
- git status
- git commit -a

Ignoring Files

- Edit .gitignore
- git add .gitignore
- git diff HEAD
- git commit
- gitk
- Can look at .git/config

Branches

- git branch hotter
- git checkout hotter
- Edit some file
- git commit -a
- git checkout master
- Edit same file
- git commit -a
- git merge hotter

Conflict? then Clone

- Fix conflict
- git add file
- git commit
- git branch -d hotter # delete branch

Remotes

- cd ..
- git clone dir1 dir2
- In dir2:
  - Look at .git/config
  - git branch –a
- Make some changes to dir1 and check them in
- Go to dir2:
  - git fetch
  - git merge origin/master

Remotes

- Changing in dir2 and doing ‘git push’ gives error because dir1 is not a bare repo
- Need to go to dir1 and make dir2 a remote:
  - git remote add dir2 ../dir2
  - git remote update (or git fetch)
  - git diff dir2/master
  - git merge dir2/master