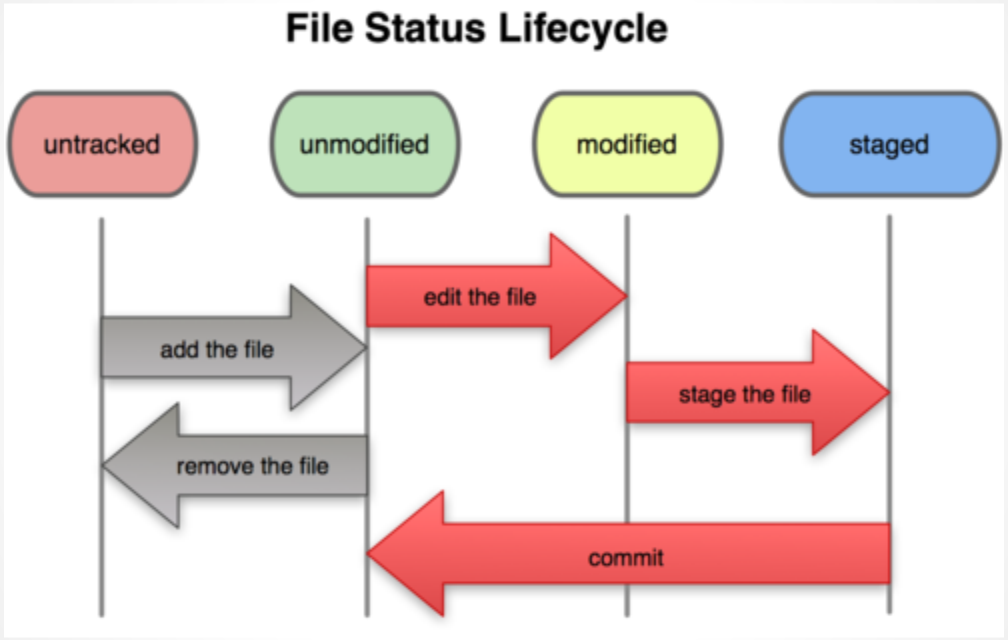
Distributed Version Control with git



gti-scm book License

Andrew Danner, SWICS
15 November 2013

5 Things you should stop doing

- Sharing code via email
- Sharing code via .tar, .zip, .tgz
- Sharing passwords
- _oldCopy22.cpp version control
- dropbox version control

4 Reasons to learn version control

- Get employed
- Share code with others
- Collaborate on cool projects
- Code audits, legal protection?

Why Version Control Systems (VCS)?

- Manage source code changes in a sane way
- Track progress
- Allow undo/revert (It worked a week ago)
- Multiple Branches
  - Release version
  - Devel version
  - Experimental features
  - Quick patches, bug fixes
- Sharing of code, Collaboration

Distributed vs Centralized

- Centralized (CVS, Subversion)
  - One central repository: the gold standard
  - All updates made against central repo
  - No access to repo? No updates
  - Must sync with central repo before adding updates
- Decentralized (git, mercurial, bazaar)
  - Multiple copies/clones/forks of repositories
  - You can always have a local repo [Part I]
  - You can optionally have a central repo [Part II]
    - push to remote, pull from remote
    - can have multiple remotes
  - More distributed sharing options

What [not] to put under version control

- DO
    - text based things made by humans
    - source code
    - scripts
- DON'T
    - large binary files that change often
        - images, audio, video
    - Things automatically built
        - executables, object files
    - Temporary files
    - Sensitive data: passwords, private ssh keys
    - Ignore these things with .gitignore file

Git: A DVCS

- Used for many projects
  - Linux kernel
  - github.com
- May seem overwhelming at first
- Can get started with a few basic commands
- Learn more incrementally
- Today: Using git locally
- Next week: Collaborative git

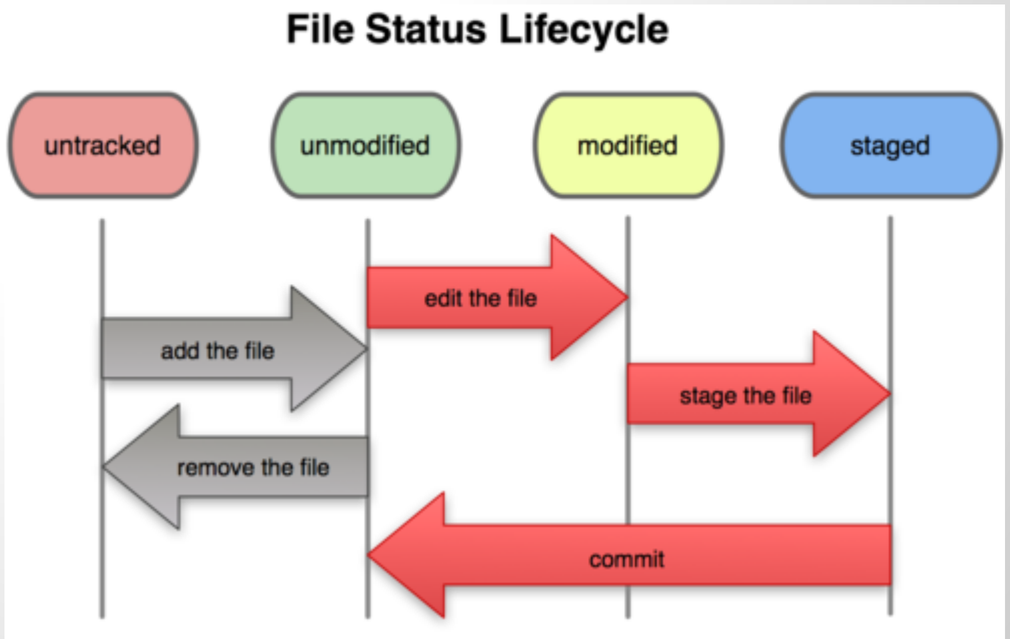
First time setup

- Do this once per network
- check git config -l
- if no user.name:
  - git config --global user.name “My Name”
  - git config --global user.email “me@place.com”
- This step will identify your code modifications as belonging to you

See also https://help.github.com/articles/set-up-git

Git: A DVCS. Initial setup

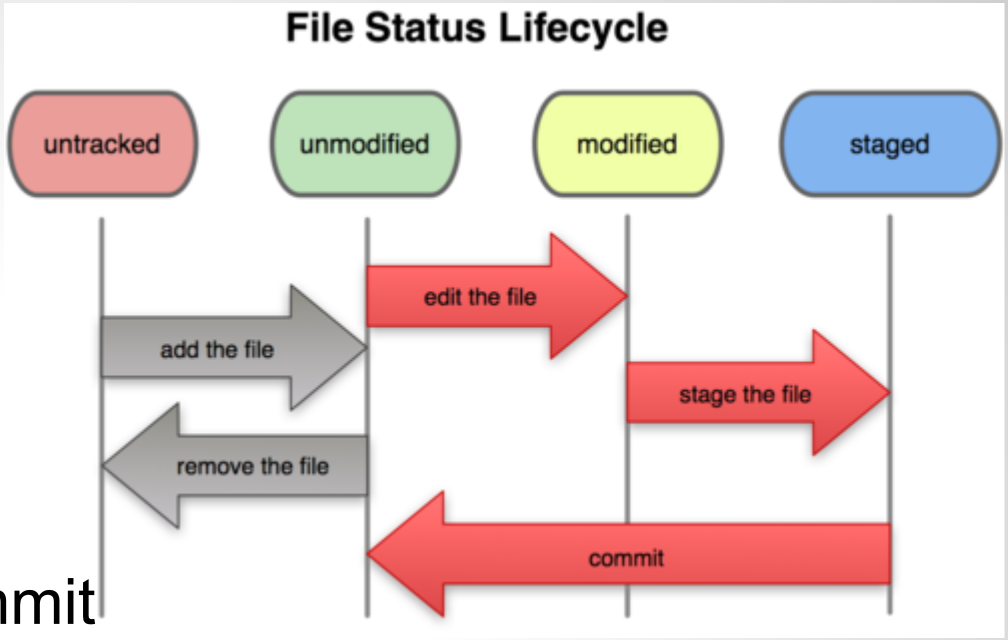
- git init woot
  - run init once per project
  - next week: git clone
- add some files
- git status (the ls of git)
- git add
- git commit
  - git commit -m
- .gitignore



Demo

```
git config -l

# if needed
git config --global user.name=”Andrew Danner”
git config --global user.email=”adanner@corgination.org”

git init woot
cd woot
vim Readme.txt
git status
git add Readme.txt
git status
git commit -m "initial checkin"

vim prog.py
vim Readme.txt
git status
git add prog.py Readme.txt
git commit -m "I'm programming"

git status

gitg
```

Daily workflow



- Edit old files
- Add new files
- git status
- git add
    - “stages” files for commit
- git commit, git commit -m
    - saves changes in history
- .gitignore
    - ignore files that you don’t want to version control
    - *.o, *.avi, *.bak, *~, .*.swp, build/*

Reviewing changes

- git log
- gitg, gitx, git
- git status
- git add
  - “stages” files for commit
- git commit, git commit -m
  - saves changes in history
- .gitignore
  - ignore files that you don’t want to version control
  - *.o, *.avi, *.bak, *~, .*.swp, build/*

Local git repo

- Working Directory
  - what your directory currently looks like
  - pretend git wasn't there
- Stage (Index)
  - Things that are added to be part of next commit
  - Not committed yet
- History (Local repository)
  - Committed from staging area
  - Part of git history
- Stash
- Upstream (Remote repository)

Git Branching and Merging

- Use branches to work on multiple features in parallel
- Test out new ideas, fix bugs
- You should do most of your development work in a branch
- There seems to be a lot of branching FUD surrounding git. These folks probably were burned by some other VCS in the past that had poor branch support
- Git has great branch support

Demo - Branching

```
git status
#create and switch to new branch
git checkout -b devel
git status
vim prog.py
python prog.py
git status
git add prog.py
git commit -m "awesome feature"

#move to existing branch
git checkout master
vim prog.py
git commit -a -m "documented code"
git status
git branch
gitg &

#fixing conflicts
git merge devel
vim prog.py
git status
git add prog.py
git commit
```

```
#fast forward merge after conflict
git checkout devel
git merge master

#not all merges result in conflict
```

Git Branching and Merging

- git branch newfeature
- git checkout newfeature
- add some changes
- git checkout master
- use gitg to view repo history
- add changes. branch divergence!!!
- git merge <frombranch>
- merge conflicts and resolutions
  - do not blindly add conflicted files back into git
  - you will most likely break your code
- git branch lists, creates, deletes branches

Undoing changes

- git mv
- git rm   removes from git and working tree
- git rm --cached only removes from git
- git checkout --
- git revert, the anti-commit
- git rebase
  - helpful when collaborating
  - only use on local repos
  - do not rebase remotes
  - not really an undo. more of a redo

Preview of next week

- Sharing with others
- cloning existing projects
- remotes, push, fetch, pull
- publishing local repos
- Swat CS git server
- github
- acls/bare repos?
- Q&A

Remote repositories

- Sharing/Collaborating is usually done with a remote repository
- git clone
- git fetch, git pull
- git push
- git remote add
- git branch -a, -av, -avv
- Local stuff still applies
- push: share from your local to remote
- pull: pull from remote to your local

Other commands

- git cherry-pick
- git stash
- git help

Other tools

- Swarthmore git server
- github for more public projects
- git svn clone

Git resources

- Pro Git book
- Git @ Swat
- Git Terminology See also *git help glossary*
- Git Ready learn git one feature at a time
- Git Immersion
- Understanding Git
- Visual Git Reference
- Git Cheatsheet