How To Use Git

Just the Basics!

Mary Kate Trost
July 8, 2011

Who Can Benefit From git?

- EVERYONE!!!
  - Students, Staff, Faculty, Researchers
- Track and backup personal/class projects
- Coordinating changes with others
- Providing/Maintaining files/scripts/programs for common use

What is Git?

- Version Control
  - Stores entire history
    - Diff against old versions
    - Revert to old versions
    - See log message for each version
  - Collaboration (optional)
    - Coordinate changes with others or across machines
  - Backup (optional)
    - An extra copy of your files
  - Share work
    - Separately work on updates

Use Already Existing Repository

- Alread existing Repository is called a remote/source/origin
  - Can be considered the ”master” copy
- Need your own local copy
  - Contains entire history
  - Is a backup copy of the original
    - any copy can can be made into a new remote
- Clone (Copy)the source repository (remote):
  - git clone *source* ← Puts in directory with same name as source's lowest level directory
  - git clone *source myName* ← Puts in directory named myName

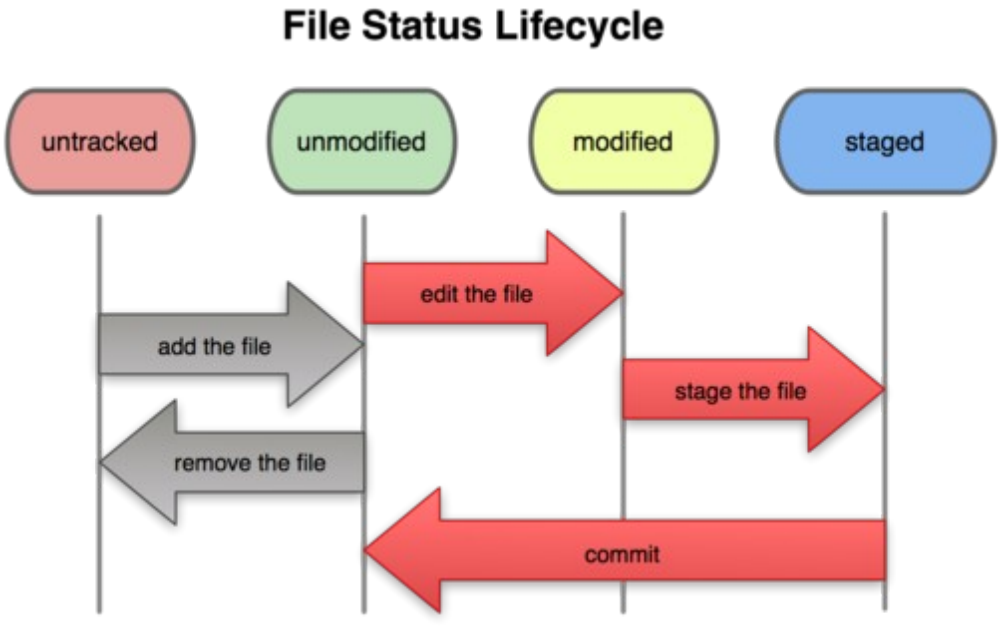
What is in a Local Git Repository?

- The Working Directory
  - Created by clone
  - Copy (checkout) of one version of the repository
    - Default is the current ”master” version
  - This is what you use to
    - Build
    - Make your changes
- .git sub-directory
  - Created by clone
  - Contains entire history
    - Access even if disconnected from the original (remote)
  - Stores changes in database
  - Do not delete it!
  - Backup of original
    - Can clone from this one

From http://progit.org/book/ch1-3.html

Working Directory File States

- Untracked
  - files not in database
- Modified
  - changed, but not stored in database
- Staged
  - prepped to be stored
- Committed/Unmodified
  - changes stored in the local database



From http://progit.org/book/ch2-2.html

Checking What Has Changed

- git status

No uncommitted changes

```
# On branch master
nothing to commit (working directory clean)
```

```
# On branch pileup
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   PileupHelper.h
#       modified:   test/Makefile
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   PileupWithGenomeReference.h
#       modified:   test/test.sh
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       PileupHelper.cpp
```

Staged Changes

Modified Files

Untracked Files

How to undo staging

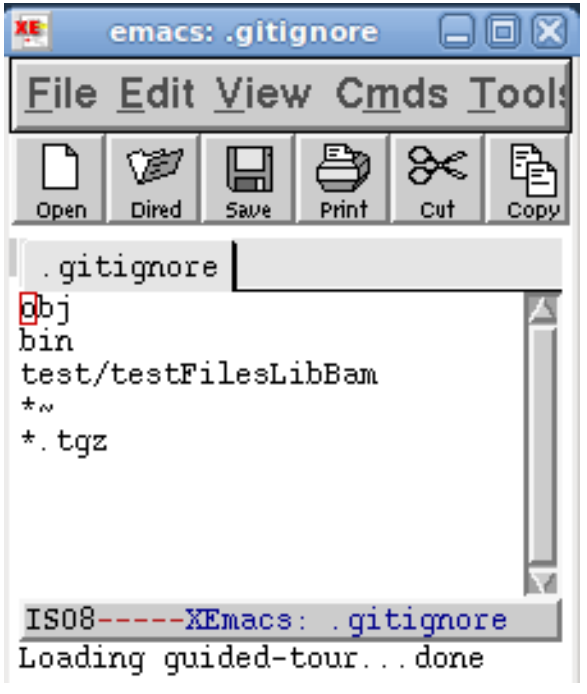
How to stage changes

How to undo changes

How to track new files

Ignoring Files

- Sometimes files you don't ever want to add
  - Files that end in '~', object files (.o), etc
- .gitignore – text file that looks similar to:
- Entire repo:
  - .git/info/exclude
- Your own set of ignored files
  - Just for you for all repos
  - git config excludesfile *path/file*



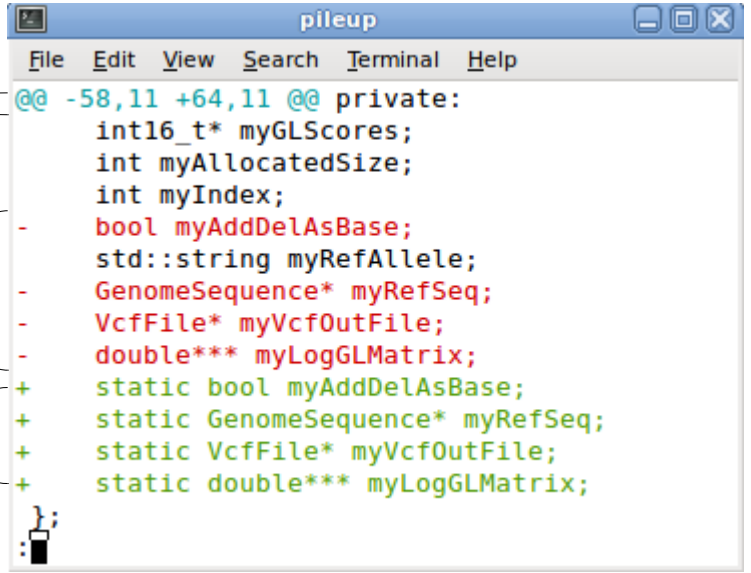
Looking at Changes

- git diff *filename*
  - Use arrows to scroll
  - Type 'q' to exit

Affected line #s

Removed lines indicated by -

New lines indicated by +



Staging your Changes

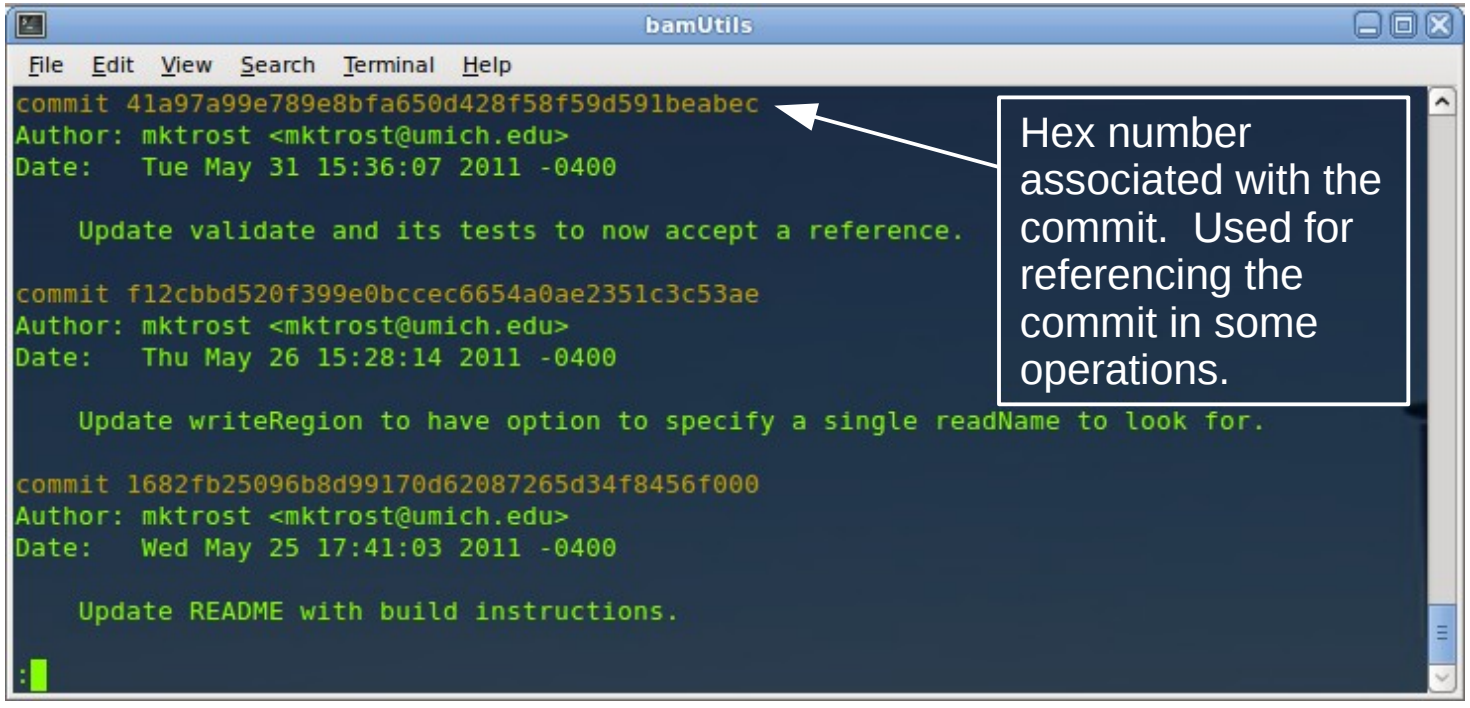
- Staging your changes/adding/removing files
  - Add new files
    - git add *filename1 filename2*
  - Stage changes to a file
    - git add *filename1 filename2*
  - Remove files (even if already removed from the directory)
    - git rm *filename1 filename2*
- Stage all changes:
  - git add .

Committing your Changes Locally

- BEFORE COMMITTING:
  - Use *git status* to check what you are adding, removing, and modifying
- Commit: Stores staged changes in the database
  - git commit -m ”*commit message*”
    - The more descriptive the commit message, the easier it is to track history
  - The set of changes is called a commit and is uniquely identified as a hex number
    - Used later to identify the commit for diffs, history, etc.

History

- git log
- git log *filename*
- -p option adds diff output to the log



Getting the Latest Updates

- git pull
  - Updates unchanged files to the latest version on *remote*
  - Merges any files that changed in *remote* and *local*
    - Unresolved if same lines edited, must be updated by hand
    - Reported on the pull command

Getting the Latest Updates

```
~/code/learnGit/learningGit$ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /home/mktrost/code/learnGit/bareRepo/learningGit
   c9bac70..c8ed4e1  master     -> origin/master
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
Automatic merge failed; fix conflicts and then commit the result.
~/code/learnGit/learningGit$ git status
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 2 and 1 different commit(s) each, respectively.
#
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      README.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
~/code/learnGit/learningGit$ cat README.txt
This repo is for Learning Git.
<<<<<<< HEAD
Update1
Update2
=======
Update from learningGit2
>>>>>>> c8ed4e15c9249a0449471195c6c46e4fba497a42
```

Merge conflict – lets you resolve

Git status indicates unmerged

Conflicts are marked in the file

<<<<<<< - start current branch's version

======= - separate's the 2 versions

>>>>>>> - end other branch's version

Resolving Merge Conflict

- Make appropriate changes
- Delete the <<<<<<<, =======, and >>>>>>>
- Add to the staged files
  - git add *nowMergedFile*
- Commit the merge
  - git commit
    - Update the default merge message with a description of how you resolved the merge
- You can also use a mergetool: git mergetool

Sharing Your Changes

- git push
  - Push your changes to the *remote*
  - Others can then *pull* them
  - If the remote was empty you need to specify where to push (origin) and what branch (master)
    - Only needs to be done once:
    - git push origin master
  - Need to ”pull” the latest changes prior to pushing changes

Creating a New ”Remote” Repo

- git init --bare --shared *myName*
- Only has the .git (database) directory
  - No source files & Can't modify/see your stored files
- Clone it in another directory as your working copy
  - Recommended to be on a different filesystem/host in case one dies, there is a backup repository stored separately
- From working copy:
  - When ready to push first set of files back to this:
    - git push origin master
  - Same as working with an already created ”remote”

Recommendations

- Commit Often
  - Especially if it compiles and you want to try something new
  - Remember, changes are only saved if committed.
- Be descriptive in your commit logs – you may be browsing them later
- May want to commit & push each day as a backup
  - May require branching to not affect anyone else
  - Put special keywords in logs to identify which ones compile and work versus ones that don't compile or are incomplete
    - Maybe start each log message with a short key identifing the fix so all commits associated with it are linked together

Gui

- There are a few options when it comes to using a gui
  - gitg, gitk, giggle, qgit, smartgit
  - I use smartgit – free for non-commercial use

Resources

- https://statgen.sph.umich.edu/wiki/How_To_Use_Git
- http://progit.org/book/
- http://www.kernel.org/pub/software/scm/git/docs/git.html