

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

- Need your own local copy
 - Contains entire history
 - Is a backup copy of the original
 - any copy 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` ←

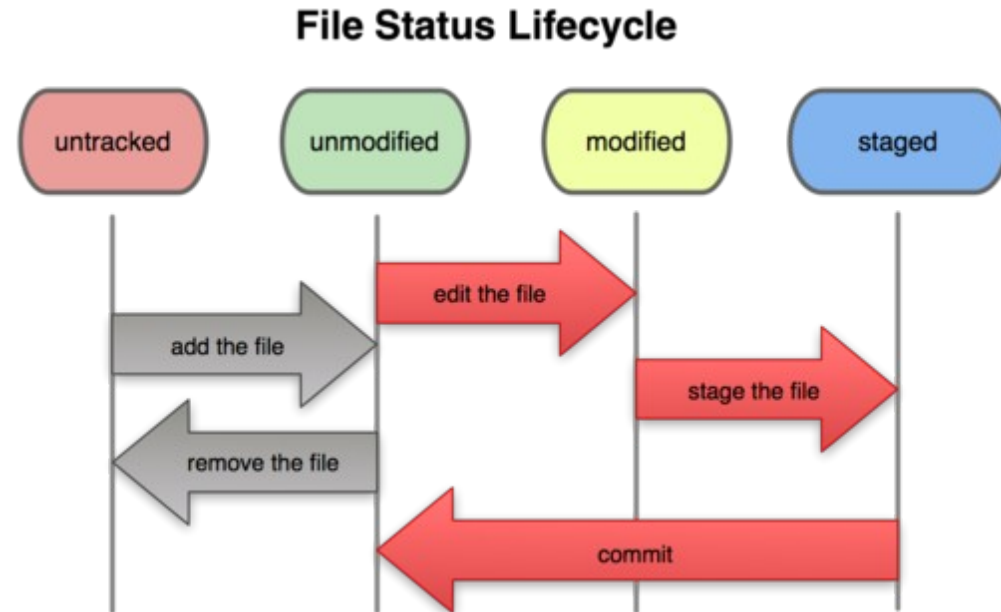
Puts in directory named myName

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

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)
```

Staged Changes

```
# On branch pileup
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#    new file:   PileupHelper.h
#    modified:  test/Makefile
```

How to undo staging

Modified Files

```
# 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
```

How to stage changes

Untracked Files

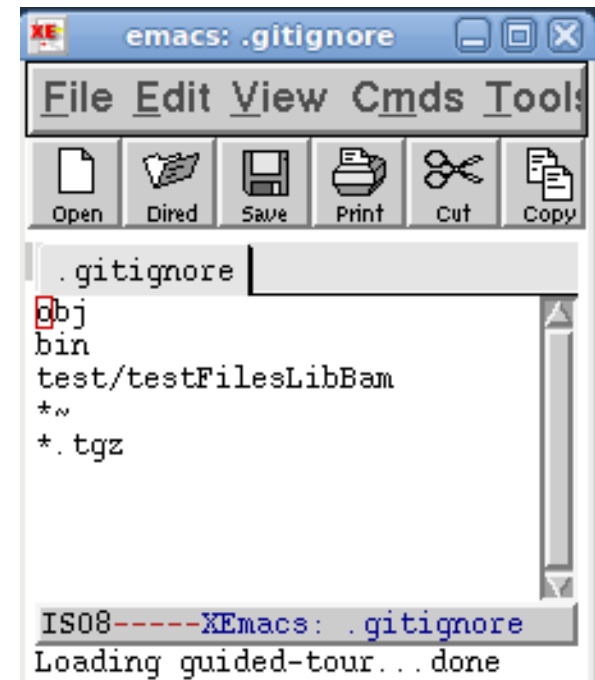
```
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#
#    PileupHelper.cpp
```

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`



The screenshot shows an Emacs editor window titled "emacs: .gitignore". The window has a menu bar with "File", "Edit", "View", "Cmds", and "Tools". Below the menu bar is a toolbar with icons for "Open", "Direc", "Save", "Print", "Cut", and "Copy". The main text area contains the following content:

```
.gitignore  
obj  
bin  
test/testFilesLibBam  
*~  
*.tgz
```

At the bottom of the window, there is a status bar that reads "ISO8-----XEmacs: .gitignore" and "Loading guided-tour... done".

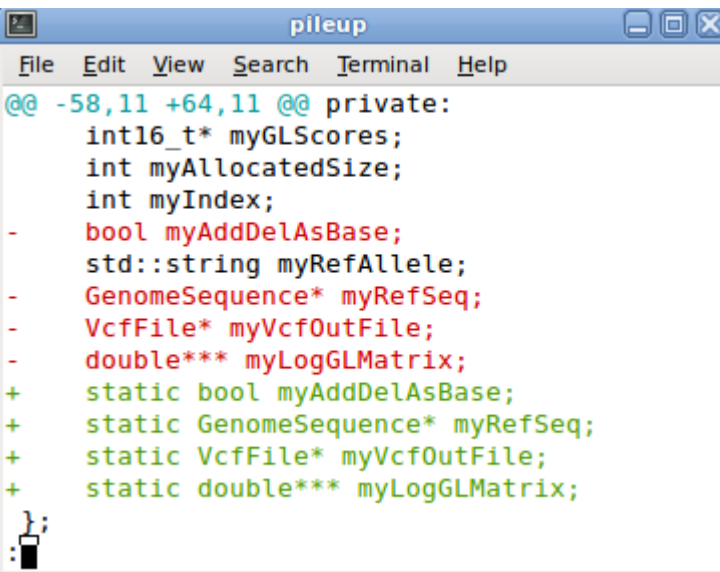
Looking at Changes

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

Affected line #s

Removed lines indicated by -

New lines indicated by +



```
@@ -58,11 +64,11 @@ private:
    int16_t* myGLScores;
    int myAllocatedSize;
    int myIndex;
-   bool myAddDelAsBase;
-   std::string myRefAllele;
-   GenomeSequence* myRefSeq;
-   VcfFile* myVcfOutFile;
-   double*** myLogGLMatrix;
+   static bool myAddDelAsBase;
+   static GenomeSequence* myRefSeq;
+   static VcfFile* myVcfOutFile;
+   static double*** myLogGLMatrix;
```

Staging your Changes

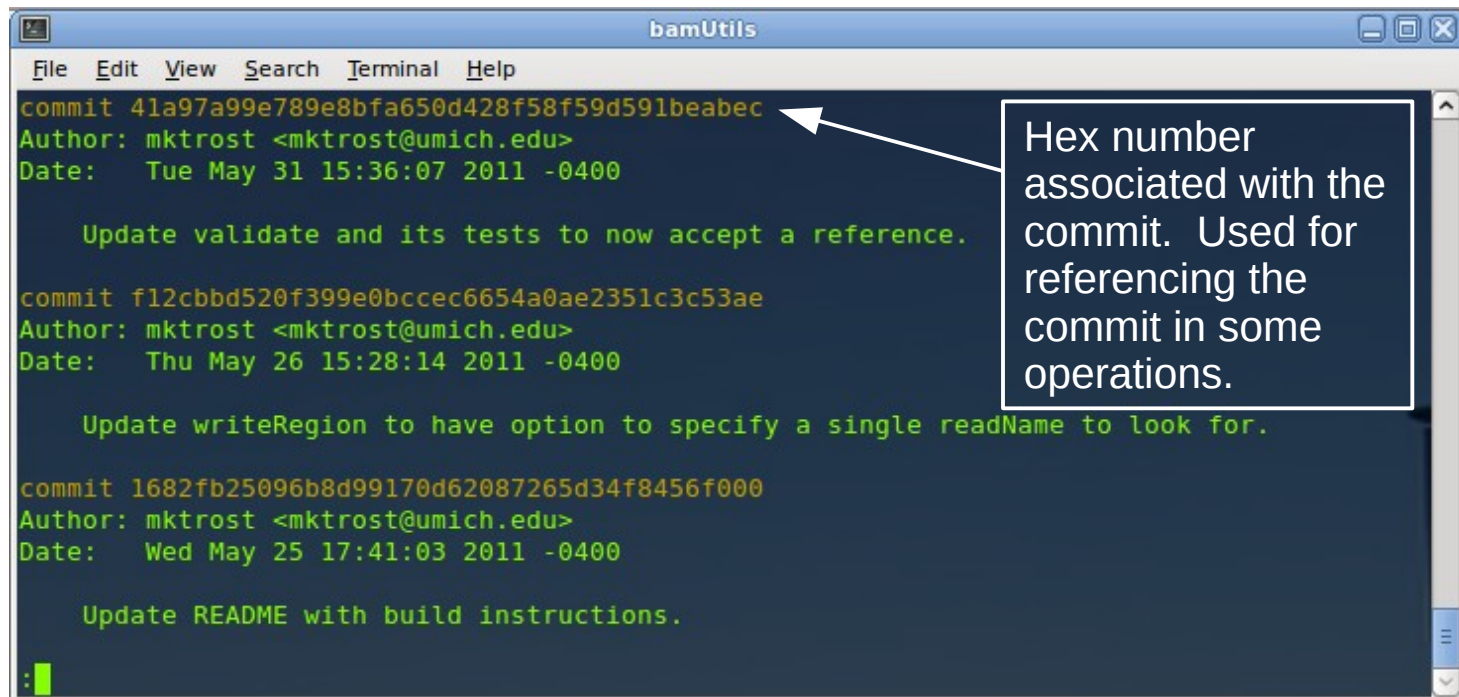
- Staging your changes/adding/removing files
 - Add new files
 - `git add filename1 filename2`
 - Stage changes to a file
 - `git add filename1 filename2`
 - Remove files (even if already removed from the directory)
 - `git rm filename1 filename2`
- Stage all changes:
 - `git add .`

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



```
File Edit View Search Terminal Help
commit 41a97a99e789e8bfa650d428f58f59d591beabec
Author: mktrost <mktrost@umich.edu>
Date: Tue May 31 15:36:07 2011 -0400

    Update validate and its tests to now accept a reference.

commit f12cbbd520f399e0bccec6654a0ae2351c3c53ae
Author: mktrost <mktrost@umich.edu>
Date: Thu May 26 15:28:14 2011 -0400

    Update writeRegion to have option to specify a single readName to look for.

commit 1682fb25096b8d99170d62087265d34f8456f000
Author: mktrost <mktrost@umich.edu>
Date: Wed May 25 17:41:03 2011 -0400

    Update README with build instructions.

:
```

Hex number associated with the commit. Used for referencing the commit in some operations.

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

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 identifying 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>