**Lab 1: Introduction to Git**

In this tutorial you will learn the basic workflow for using git with Rosetta. In Rosetta we use a branching model for development. All work should be done in a branch, and merged into the “master” branch when it is ready to be tested.

**Important: In this tutorial, replace every instance of <username> with your git username (no angle brackets)**

### Visualizing changes to git repositories

1. There are tons of gui clients for all major OSes <http://git-scm.com/downloads/guis>
2. gitk is included with git. It is equally ugly in every operating system, but it's truly cross platform and installed by default, so we'll use it here.
3. Change to the directory containing your main repository and start gitk
   1. gitk &
4. Make a new view which will show every reference, including remote and deleted references
   1. in the menu View → New View...
   2. Set View name to "Everything"
   3. select "Remember this View"
   4. select "All refs" and "All (local) branches"
   5. click Apply
   6. click OK
5. To keep track of what you're doing, you can now press F5 in the gitk window after you enter each of the git commands below to keep visual track of your progress
6. It is probably worth keeping gitk or some other gui open while you perform git operations until you are comfortable using git.

**Creating a new branch**

1. Before doing any work, you need to make a branch. We’ll be making a branch off of the most recent revision in “master”
   1. First, make sure you are in the master branch

#### git branch

#### the output of this command will be a list of all local branches, with an \* before your current branch: > git branch \* master

* 1. If you are not in the master branch, switch to it

#### git checkout master

* + 1. Now rerun the git branch command -- you should see an asterix next to master.
  1. Make a new branch based on master, and switch to it
     1. When working with rosetta, all your branches should be prefixed with your username, to allow for proper “namespacing”

#### e.g. <username>/git\_activity\_1

#### If you used the get\_rosetta.sh script to download Rosetta, there’s a Rosetta-specific alias to help you with branching (Note that you should not give your user name.):

#### git personal-tracked-branch git\_activity\_1

* + - 1. Do not try to run git personal-tracked-branch <username>/git\_activity\_1
    1. (Internally, this does four things, in case you were curious -- you don’t need to run any of these commands yourself:
       1. git branch <username>/git\_activity\_1 - Create a branch with your Github username prefixed; your alias already knows your username -- you had to tell the get\_rosetta.sh script your username when you ran it.
       2. git checkout <username>/git\_activity\_1 - Switch to that branch
       3. git push origin <username>/git\_activity\_1 - Tell Github about the branch
       4. git branch --set-upstream <username>/git\_activity\_1 origin/<username>/git\_activity\_1 - Tell your local repository that it can pull from the copy of the branch on Github)
    2. git branch
       1. This should say that you are no longer in master -- the asterix should have moved to <username>/git\_activitiy\_1 if you’re still in master, then flag one of us down.

1. Now we're going to do a number of operations to the new branch to demonstrate how git handles them
   1. modify a file
      1. source/src/protocols/init/init.cc
         1. remove the word “Commons” from line 4
      2. git status
      3. Note that the modification is made but not staged

#### git add source/src/protocols/init/init.cc

* + 1. git status
    2. Note the modified file is now staged; the “add” command staged the file

#### git commit

#### This will open up your default editor, or whichever editor is specified in your .git/config file. This is probably “vi”, and the sad secret of the universe is that vi hates all of its users. It has made it darn near impossible to even quit it. But I know the secret: quit vi by hitting “esc esc w q ! enter”. In retrospect, doesn’t that series of key strokes seem obvious?

* + - 1. You will see that git has said that it isn’t going to commit because you didn’t provide a commit message. We’ll give it a commit message, but first, we’re going to tell git that we would like to use a civilized text editor: emacs.
      2. Run this command:
      3. git config --global core.editor emacs
      4. Now rerun the git commit command
      5. Type in a commit message “Changing the Rosetta Commons organization name; I probably have the authority to do that”
      6. Lines that begin with “#” are comments and won’t be saved. Make sure you don’t have a “#” in front of your message.
      7. Hit “ctrl-x ctrl-s” to save, and “ctrl-x ctrl-c” to exit. Ok, I guess you could say that emacs doesn’t like its users either.

#### git status

* + 1. What happened to your changes? (Refresh your gitk/source tree window.)
    2. Run git log --graph
       1. You should see your new commit and its commit message at the top
  1. Make another modification
     1. source/src/protocols/init/init.hh
        1. Line 22, change “C++ headers” to read “Java headers”
     2. Don’t stage or commit this change yet.
  2. Add a file

#### echo "this is a new file" > path/to/Rosetta/main/database/new\_file.txt

* + - 1. When I say “/path/to/” I mean “Replace these directories with ones that make sense on your computer; if you downloaded Rosetta into your home directory then you could type “~/Rosetta/main/database/new\_file.txt”
    1. git status
    2. Note that the status of a new file is slightly different than a file that was already part of the repository that you have changed

#### git add /path/to/Rosetta/main/database/new\_file.txt

#### git commit -m "adding new\_file.txt to the database"

* 1. Delete a file[[1]](#footnote-0)

#### git rm source/Doxyfile.src

* + 1. git status

#### git commit -m "removing Doxyfile.src"

* 1. Move a file[[2]](#footnote-1)

#### git mv source/BuildDocs.sh source/Old\_BuildDocs.sh

* + 1. git status

#### git commit -m "moving projects.settings"

* 1. Do we have any outstanding changes?
     1. git status
  2. What are those changes
     1. git diff
  3. Commit all the outstanding changes
     1. git commit -a -m "Forgotten changes"

1. Now, lets look at the changes we made

#### git log

* 1. Also look at the changes in the gitk window.

**Merging and Resolving Conflicts**

1. If the same line of file is modified in two branches, merging those branches will cause a *conflict* that requires manual resolution
2. Changes that modify different parts of the same file, or different files, are resolved automatically
3. Make a new branch off of master

#### git checkout master

* + 1. This will “undo” the modifications you made above; your modifications still exist in the repository, and you can get back to them, but the contents of the files will appear to have changed.

#### git checkout -b <username>/git\_activity\_2

* + 1. This makes a branch locally, but not on Github. (You can go to the Github website to check.) You can still push it to Github later if you want.
    2. Note that you should give your username here where you didn’t have to do that when you used the “personal-tracked-branch” command.

1. Introduce a change that will require manual resolution
   1. source/src/protocols/init/init.cc
      1. Add the word "something" to the end of line 4

#### git add source/src/protocols/init/init.cc

#### git commit

#### Type the message: "a change that conflicts with git\_activity\_1"

1. Introduce a change that will be resolved automatically
   1. source/src/protocols/init/init.hh
      1. remove lines 10-14 (the file/brief/author lines).

#### git add source/src/protocols/init/init.hh

#### git commit

#### Type the message: "modifying protocols/init/init.hh"

* + 1. git status and git log
       1. Make sure your commits are there in your branch. If git status shows files that you just modified and that you meant to commit, then you may have forgotten to use “git add” first.

1. Attempt to merge <username>/git\_activity\_2 into <username>/git\_activity\_1
   1. merges occur in one direction at a time. The operations below will merge git\_activity\_2 into git\_activity\_1 but will not modify git\_activity\_2.

#### git checkout <username>/git\_activity\_1

* 1. Let’s compare the two branches first:
  2. git diff <username>/git\_activity\_2
     1. This compares the current working directory (including uncommitted files, if any) with the given branch

#### git merge <username>/git\_activity\_2

* + 1. You will see output like this:

#### Auto-merging source/src/protocols/init/init.cc CONFLICT (content): Merge conflict in source/src/protocols/init/init.cc Automatic merge failed; fix conflicts and then commit the result.

1. Resolve the conflict and commit the merge
   1. git status can be used to view the current status of a merge in progress
   2. paths that have not yet been merged will be listed under the "unmerged paths" section
   3. open source/src/protocols/init/init.cc in a text editor
      1. The conflict will be indicated using brackets

#### <<<<<<< HEAD // (c) Copyright Rosetta Member Institutions. ======= // (c) Copyright Rosetta Commons Member Institutions. Something >>>>>>> <username>/git\_activity\_2

* + 1. The lines between <<<<<<< HEAD and ======= indicate the section of the file modified in the branch you are merging into (<username>/git\_activity\_1)
    2. the lines between ======= and >>>>>>> <username>/git\_activity\_2 indicate the section of the file modified in the branch you are merging from (<username>/git\_activity\_2)
    3. Change line 4 to reflect the changes made by both versions:

#### // (c) Copyright Rosetta Member Institutions. Something

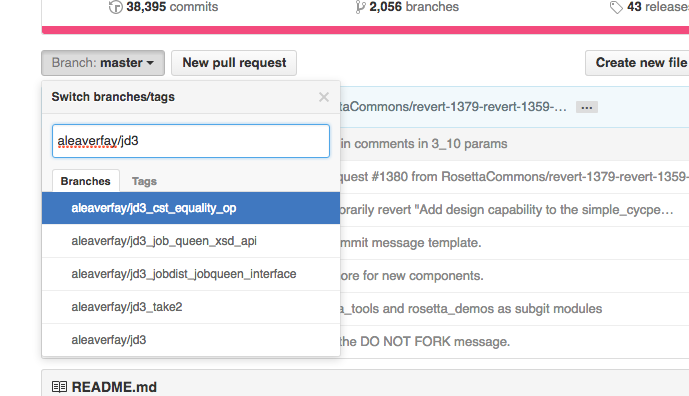
* + - 1. Make sure to remove the conflict indicators (the <<<’s, ===’s, and >>>’s)
  1. mark the file as resolved

#### git add source/src/protocols/init/init.cc

* + - 1. This is the THIRD thing you’ve seen the “git add” command do:
         1. It stages files to be commited,
         2. It tells git to start tracking new files, and
         3. It tells git that you have finished fixing conflicts during a merge.
  1. Open source/src/protocols/init/init.hh to see how the modifications to it in both branches were automatically resolved
  2. commit the finished merge

#### git commit -m "grumble grumble"

* 1. check the commit message from the terminal
     1. git log --graph
     2. OH NO! You have left a less-than detailed commit message. The horror!
  2. edit the commit message to conform to good style
     1. git commit --amend
     2. This will open up emacs; type this message: merging <username>/git\_activity\_2 into <username>/git\_activity\_1
  3. check things again, this time short style
     1. git log --graph --oneline

1. Note: In normal development instead of merging two user branches, you’d usually be merging a user branch into master. (Though merging two user branches happens frequently too.) So in steps 6&7 instead of branch “<username>/git\_activity\_1” you’d typically be merging into branch “master”. (But as we don’t want to mess up master, we’re doing it in different branches.)
2. Let’s update GitHub with the updated code. (This is a good idea when we’ve merged with master; you should also regularly push your code to githubt -- two or three times a day -- when you’re working in a branch.)
   1. Check to make sure you’re on the branch you want to push
      1. git branch
   2. Push the branch to origin (Github)
      1. git push origin <username>/git\_activity\_1
      2. If you have a recent version of git, the automatic setup with personal-tracked-branch means that a simple “git push” will automatically know that you want to push to the <username>/git\_activity\_1 branch on origin.
   3. Go to <https://github.com/RosettaCommons/main>
      1. Click on the “Branch: **master**” drop down menu to look for your branch  
         
      2. Is your branch there? There’s a long list of branches! Start typing in the name of your branch to get to your branch faster.
      3. What does the GitHub webpage say about it? What commits have been made to that branch?

**Deleting branches**

1. Branches in git can be deleted
2. If a branch is deleted and has not been merged, the changes associated in that branch will eventually be permanently deleted (usually after about 2 weeks)
3. Once a branch has been merged into another branch, it can be safely deleted without losing data
4. Delete branch <username>/git\_activity\_2 from the local repository

#### git branch -d <username>/git\_activity\_2

* 1. What does a plain “git branch” show now?
  2. -D will delete a branch that has not been merged with another branch. -d will not. -D should be used with great care.

1. Delete a branch from a remote repository (Once you’re done with a branch, it’s helpful to declutter Github.)
   1. git push origin --delete <username>/git\_activity\_1
   2. Look for the branch on <https://github.com/RosettaCommons/main>
   3. git branch
   4. Did the “push --delete” also delete the local copy of the branch?

**Undoing changes**

1. Lets try making and then undoing some mistakes. (You can do this either in the <username>/git\_activity\_1 branch, or you can make a new one.)
2. Add some files and make some modifications
   1. echo “Test contents” > test1
   2. edit README.md and change “Don’t” to “DO NOT”
3. Commit your changes
   1. git add test1 README.md
   2. git commit -m “Changes I’m not sure of.”
4. Make different changes (but don’t stage or commit).
   1. echo “More test contents” > test2
   2. edit README.md and change “DO NOT” to “Absolutely DO NOT”
5. Oops, on second thought, we didn’t want to make that prior commit just yet.
   1. git log
   2. Note the SHA1 of the commit prior to the “Changes I’m not sure of.” commit
   3. git reset --soft <SHA1 of prior commit>
   4. git status
   5. git log
6. Unstage the changes that were staged previously
   1. git reset --mixed <SHA1 of prior commit>
   2. git status
   3. What does README.md look like?
7. These changes were garbage, let’s go back to a commit we really like
   1. git reset --hard <SHA1 of prior commit>
      1. This command is the most dangerous command in git: with reset --hard, you will wipe out any uncommitted changes in your working directory. I use this command a lot, but I always run a “git status” first, and I always pause with my hand hovering over the enter key before I fire it off.
      2. If there are no uncommited changes, then you can in fact undo a git reset --hard.
   2. git status
   3. What do README.md and test1 and test2 look like?
   4. Can you see the “Changes I’m not sure of.” commit with gitk?
8. Well, maybe I do want that commit (is it too late to get it back?)
   1. git reflog
   2. Find the entry for “Changes I’m not sure of.”
   3. git reset --hard <Abbreviated SHA1 for “Changes …” commit>
   4. What do README.md, test1 and test2 look like?
   5. What does that imply about “reset --hard” and uncommitted changes?

**Exploration**

If you have time, play around with making, adding, removing and committing files and directories, as well as branching and merging. (Note, only make changes under your <username>/ namespace. Avoid modifying master or other people’s branches.)

Also take a look at the various help pages for the commands you just learned. (“git help <command>” and “git <command> -h”).

Here’s some things you can be looking out for (Google may also help).

* How do you compare the changes between two arbitrary revisions? (git diff)
  + What’s the difference between two dots and three dots in git diff?
* How do you compare a single file with an older version of that file? (git diff)
* How do you compare files while ignoring whitespace changes? (git diff)
* How do you figure out who last edited a line in a file? (git blame)
* How do you checkout just one file from a commit? (git checkout)
  + For example, if you want to revert a file to an older version.
* How do you unstage just one file?
* How can you change the amount of information/format of git log?
* If I gave you a SHA1 of a commit, how would you look at the changes for that commit? (git show; git diff; gitk)
* How do you temporarily store uncommited changes? (git stash)
* (Advanced) What is git bisect, and how would you use it?

1. If you should one day delete a file without using git rm, execute 'git rm filename' after the fact to stage the deletion [↑](#footnote-ref-0)
2. If you should one day move a file without using git mv, execute 'git rm old\_path' and 'git add new\_path' after the fact to stage the move [↑](#footnote-ref-1)