Using "git" and github
Wyatt Newman
January, 2015

In this course, we will be using a hosting service, "github" ("bitbucket" is a popular alternative) and "git" for version control (see http://git-scm.com/book/en/Getting-Started). You will need an account on github, which you can do on-line at github.com (alternatively, bitbucket.org for bitbucket).   You should register for your account using your student e-mail, as terms are friendlier for academic institutions.

1) Forking the class repository:
You will want to make a copy of the class repository (a "fork") that resides on github, and then a copy of this fork (a "clone") that resides on your local disk drive. You should assume that your "fork" on github is the data to be protected. It will get updated with changes from the shared class repository, as well as updated from changes from your workstation. You should assume that the "clone" on your workstation is *not* safe, and that *only* code that has been "pushed" to your "fork" will be protected.

The following describes how to create a "fork." You will only need to do this *once*. Thereafter, you will regularly access this copy ("fork") of the repository.

In a browser, at at address: https://github.com/cwru-robotics/cwru-ros-pkg-hydro
you should see a view like the following:
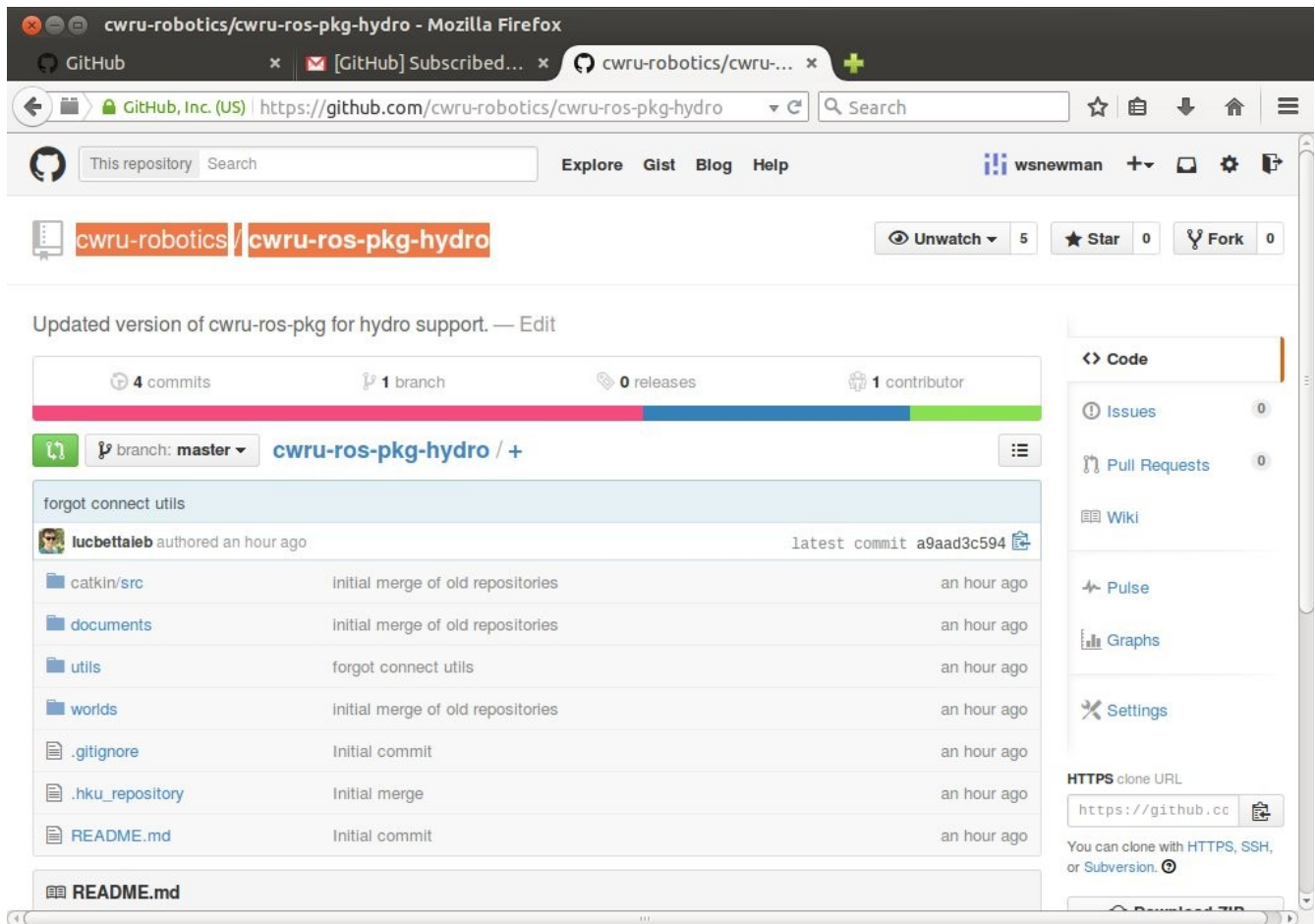
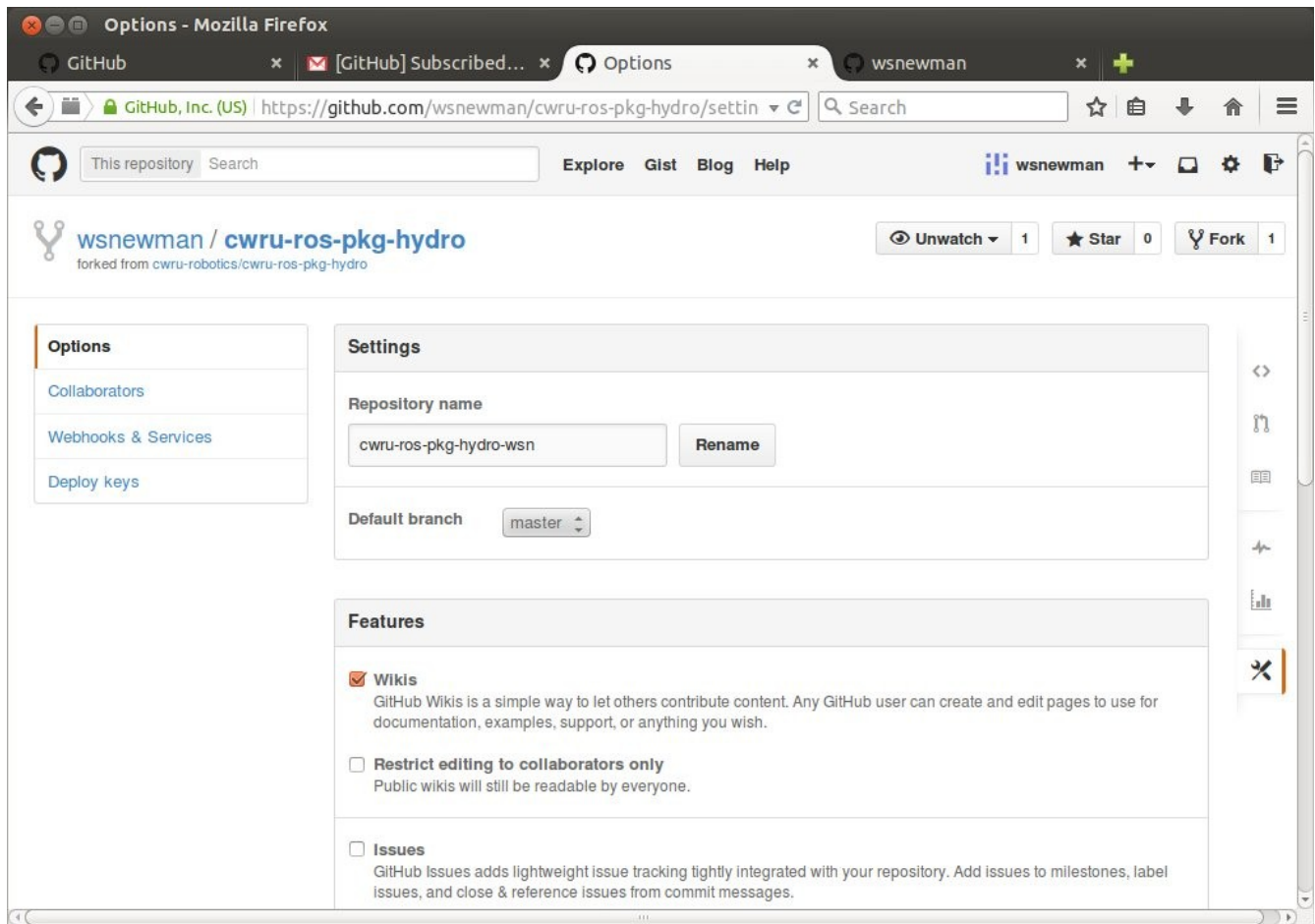*Illustration 1: master repository view*

Towards the upper right corner, there is a button labelled "fork." Push this button and you will get a complete copy of the class repository under your ownership. This is where your future code will live.

The result of my "fork" of the class repository looks like the following:
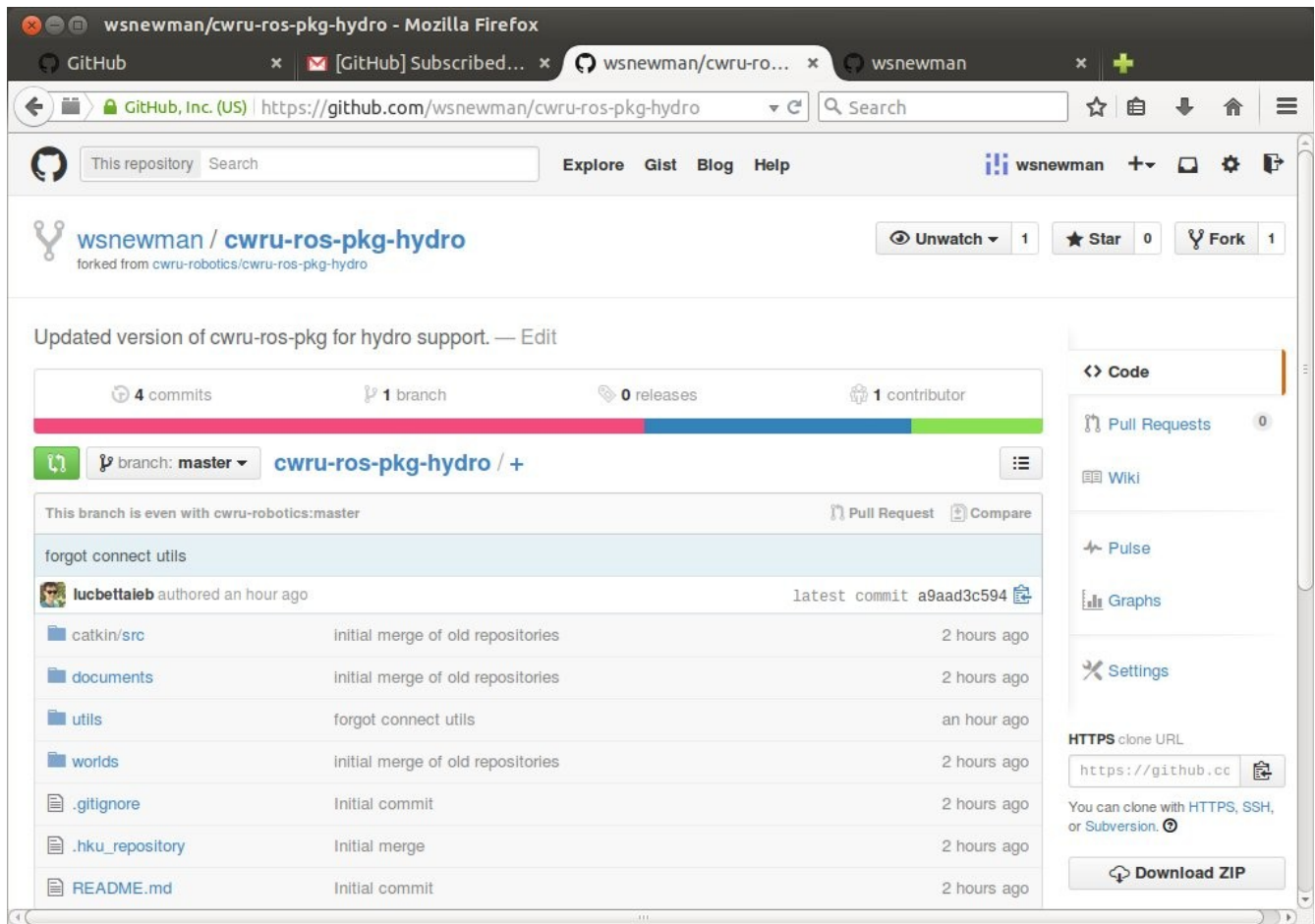
Illustration 2: View of fork of master repository

On the right-hand column of the above screenshot, there is a link called "settings." Click this, and you will be able to edit the repository name. Modify the repository name to include your own name (as I have done below, renaming this repository as cwru-ros-pkg-hydro-wsn).

Illustration 3: changing the forked repository's name

After renaming, the browser view looks like the following (note new repo name).
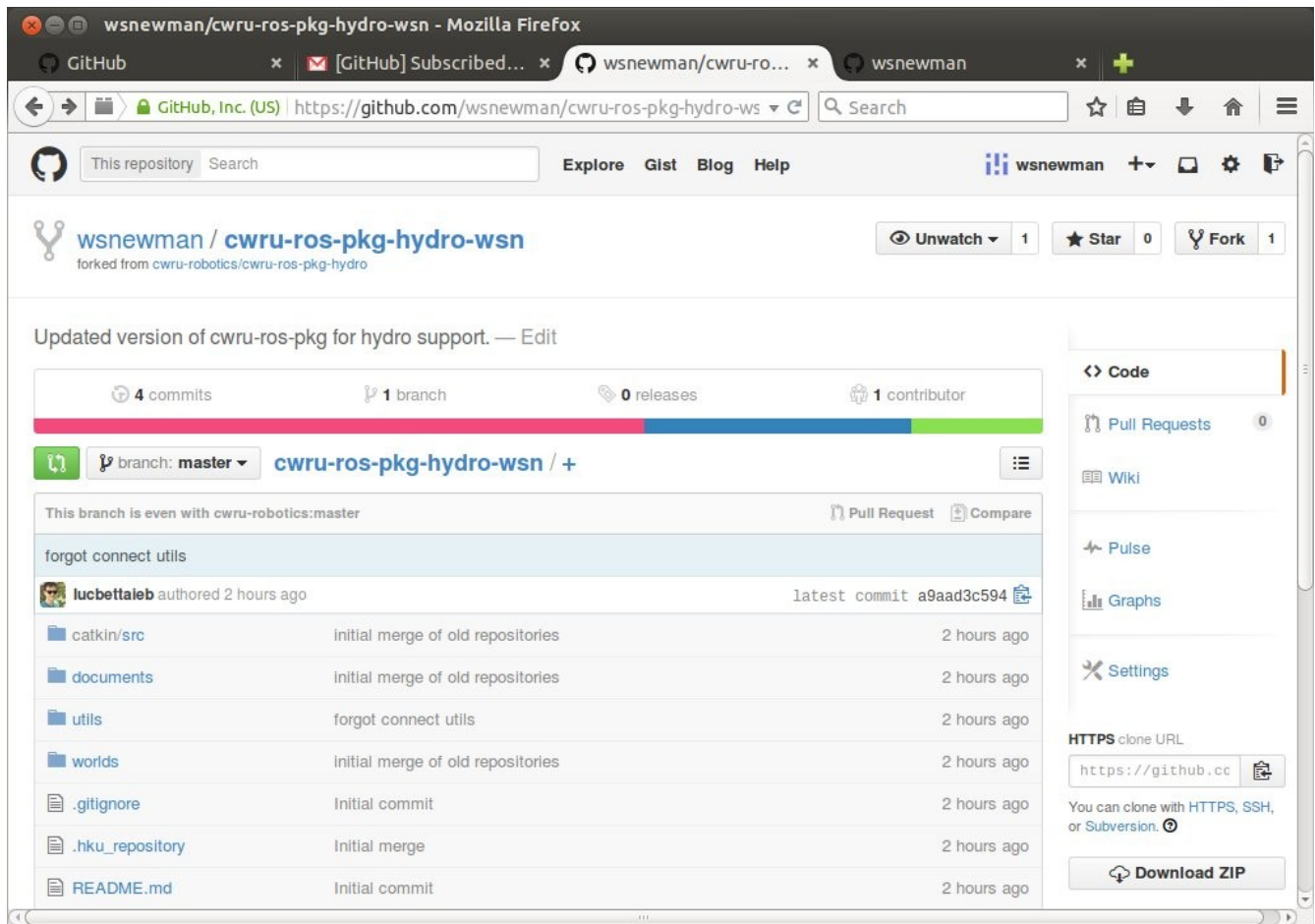
Illustration 4: view of new fork after change of name

2) Cloning your repository:
Although you will "fork" the class repository only once, you will ordinarily "clone" your repository every time to begin new development work. Particularly if you are moving around between computers, you cannot assume that you have a valid repository clone present. To make sure your repository is up to date, "clone" it from your fork.
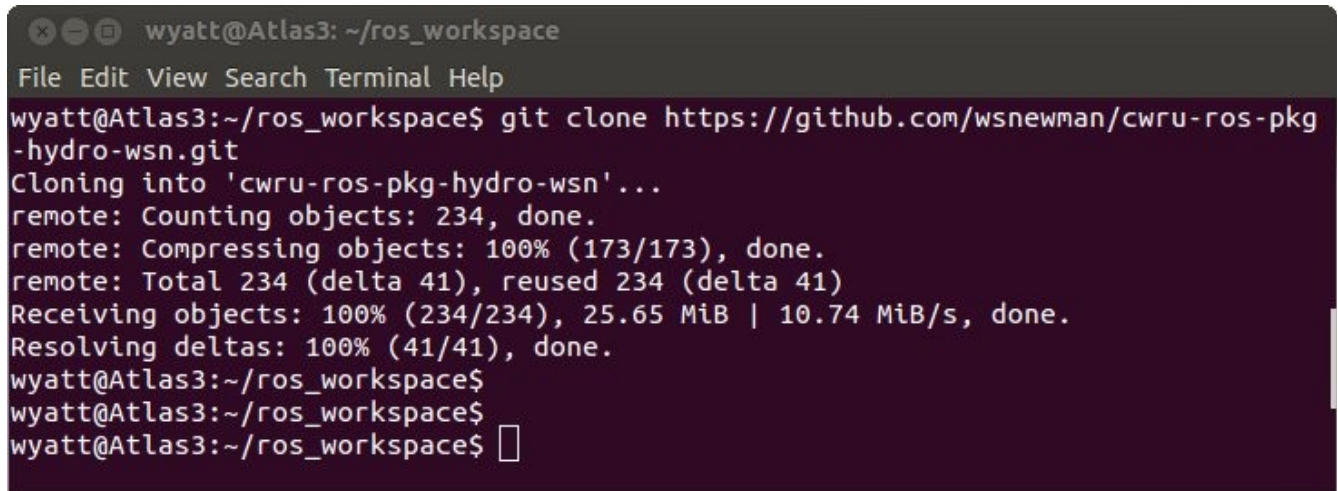
To do so, on your local Linux computer, bring up a terminal, cd to the appropriate directory (~/ros_workspace) and enter the clone command. For this command, you will need an argument that is the network path to your directory. To get this path, go back to your browser page for your new forked repository and on the right-side column (e.g. of the above figure), see the box labeled "HTTPS clone URL." Copy the contents of this box, which you will paste in as part of the clone command.

Back in your Unix terminal, enter:
      git clone https://github.com/wsnewman/cwru-ros-pkg-hydro-wsn.git

In the above, the text in orange is an example https address, obtained from the copy/paste of the github page (above figure). You should copy/paste your own fork's address here.

The result of this command should look something like the following in your terminal:

*Illustration 5: terminal view of cloning your forked repository*

At this point, you have a complete copy of *your fork* of the class repository both on github and on your local disk drive. You can be sure, at this point, that your clone is identical to your fork. You will want to keep your fork up to date both with new code that you add, as well as new code from the shared class repository.

3) Updating your fork from the class repository:
Note that in Fig 4, the github view of the forked repository contains the line "this branch is even with cwru-robotics:master." This tells you that there are no new changes in the class repository that you need.

For illustration, I made a small edit to the "README" file in the shared repository. Going back to the fork at: https://github.com/wsnewman/cwru-ros-pkg-hydro-wsn, the view now looks like this:
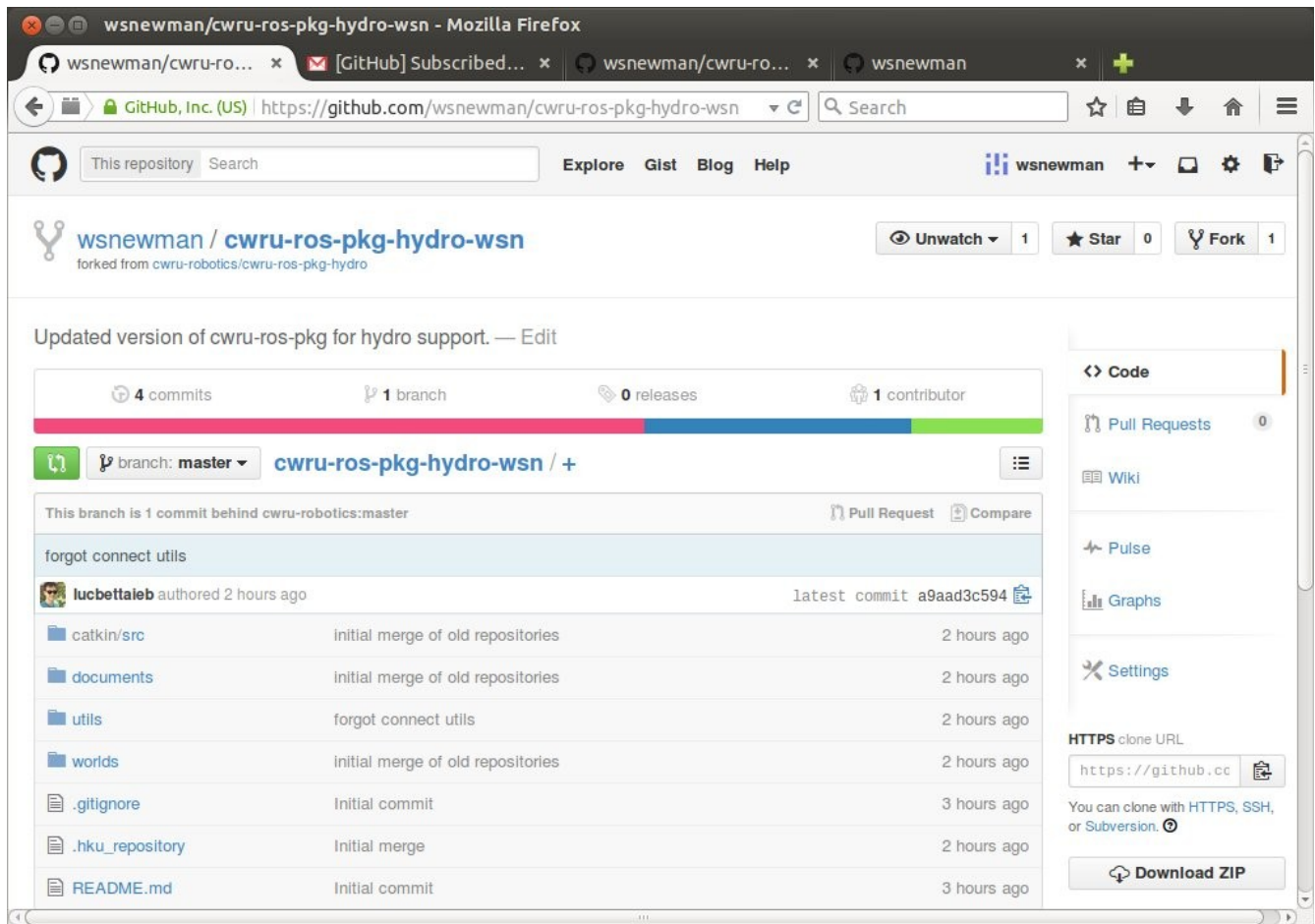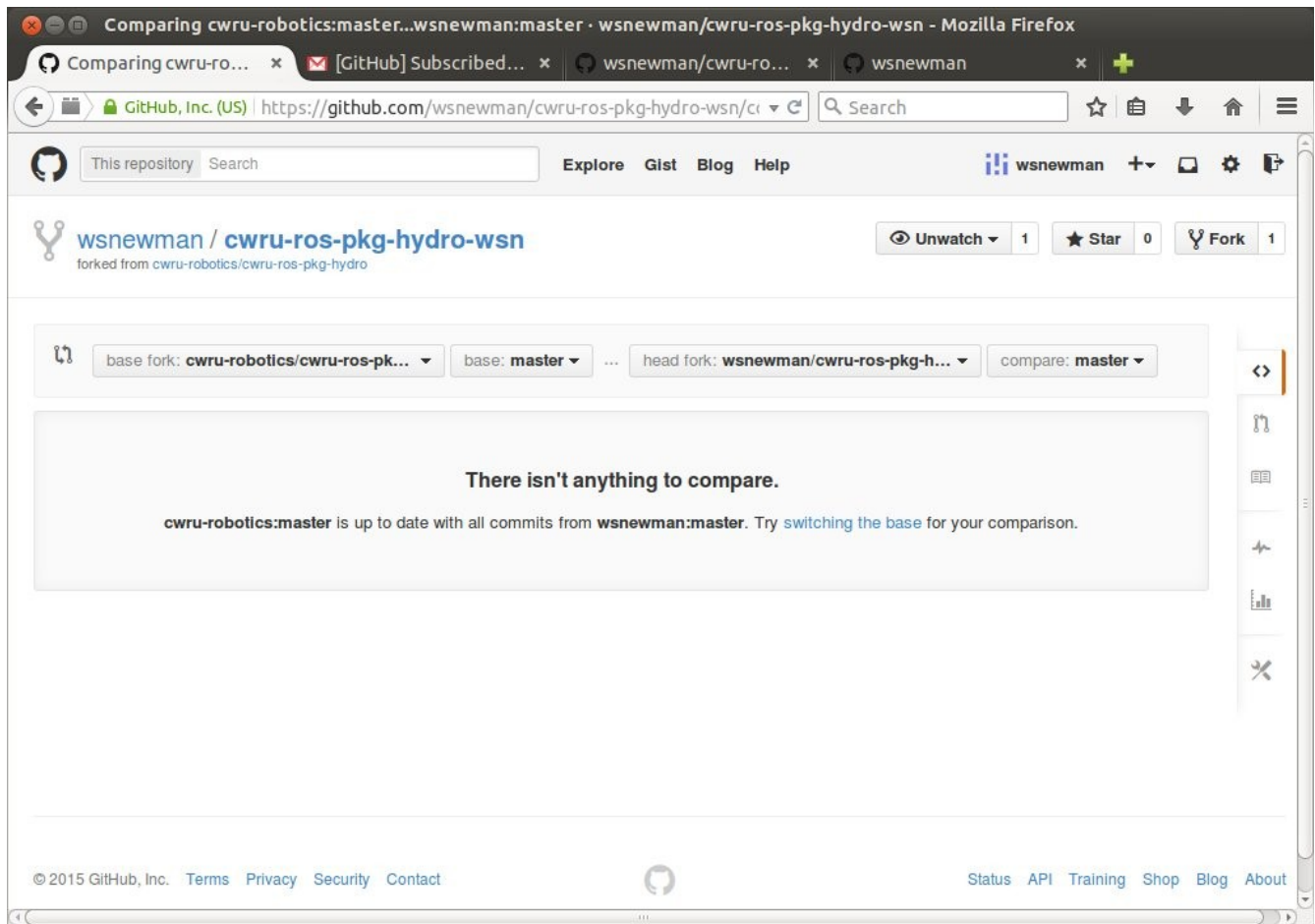
Illustration 6: fork view after class repo change

Note that this view now says "This branch is 1 commit behind cwru-robotics:master." This tells us that there have been changes to the shared repository. We will want to incorporate these changes in our fork.

Click the button in this line labeled "Compare" (to the right of the message "... 1 commit behind..."). The view looks like this:

Illustration 7: initial result of clicking "compare"

The claim "isn't anything to compare" is because we have nothing to contribute to the class repo. This will normally be the case—we typically will not fold in your work into the shared repo. However, you DO want to bring in changes from the class repo into your fork (e.g., new code I have posted for you). To do this, click the link "switching the base." The web view comes up as follows:
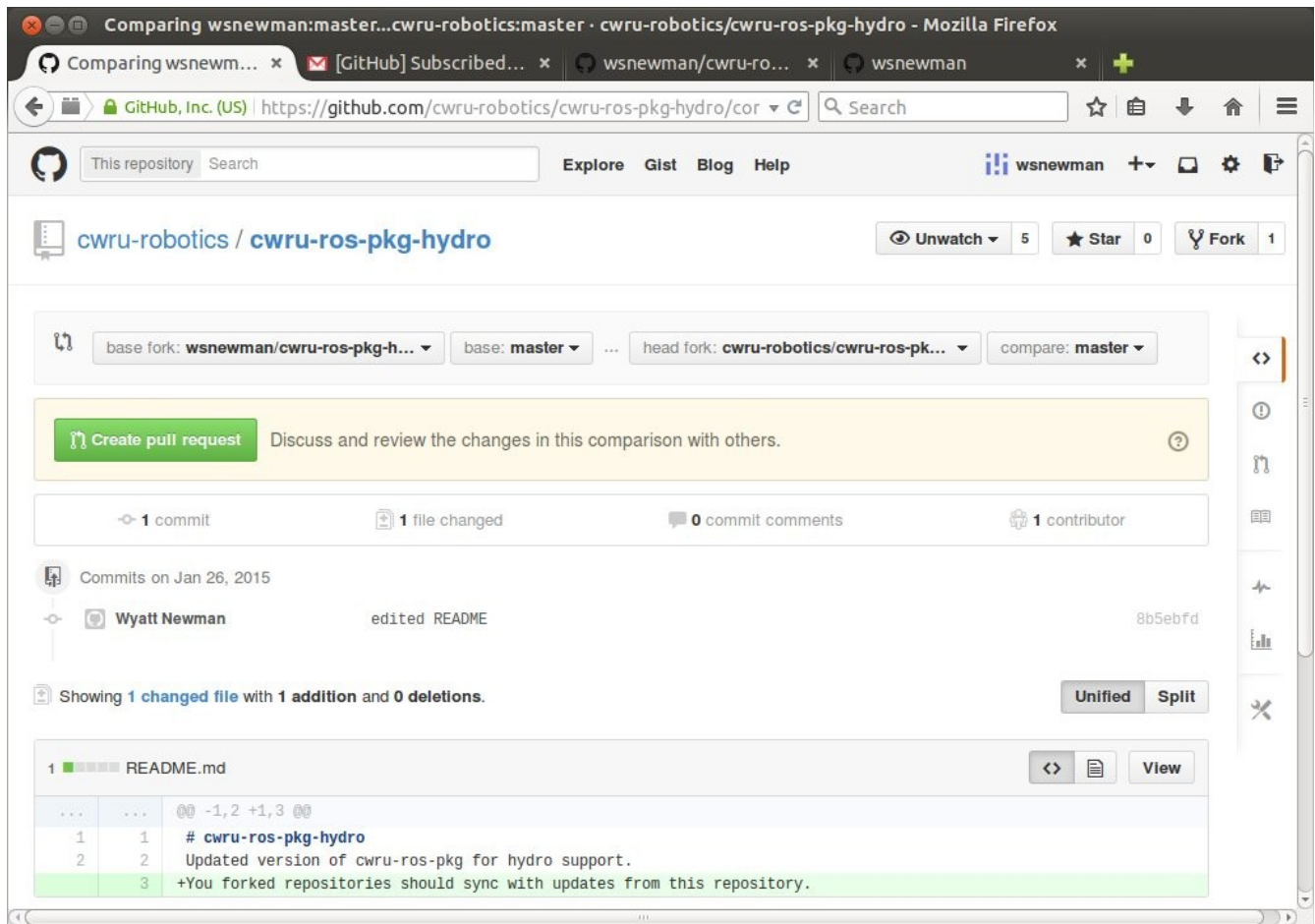
*Illustration 8: github view after switching base of comparison*

Note that the fields "base fork" and "head fork" have swapped places. You can also see the edits that were performed on "README". We want to bring our fork up to date with the changes in the class repository. To do so, click on the large, green button labelled "Create pull request". This is the mechanism to suggest folding in changes from repo1 to repo2. In this case, repo1 is the class repo, and repo2 is your personal forked copy.
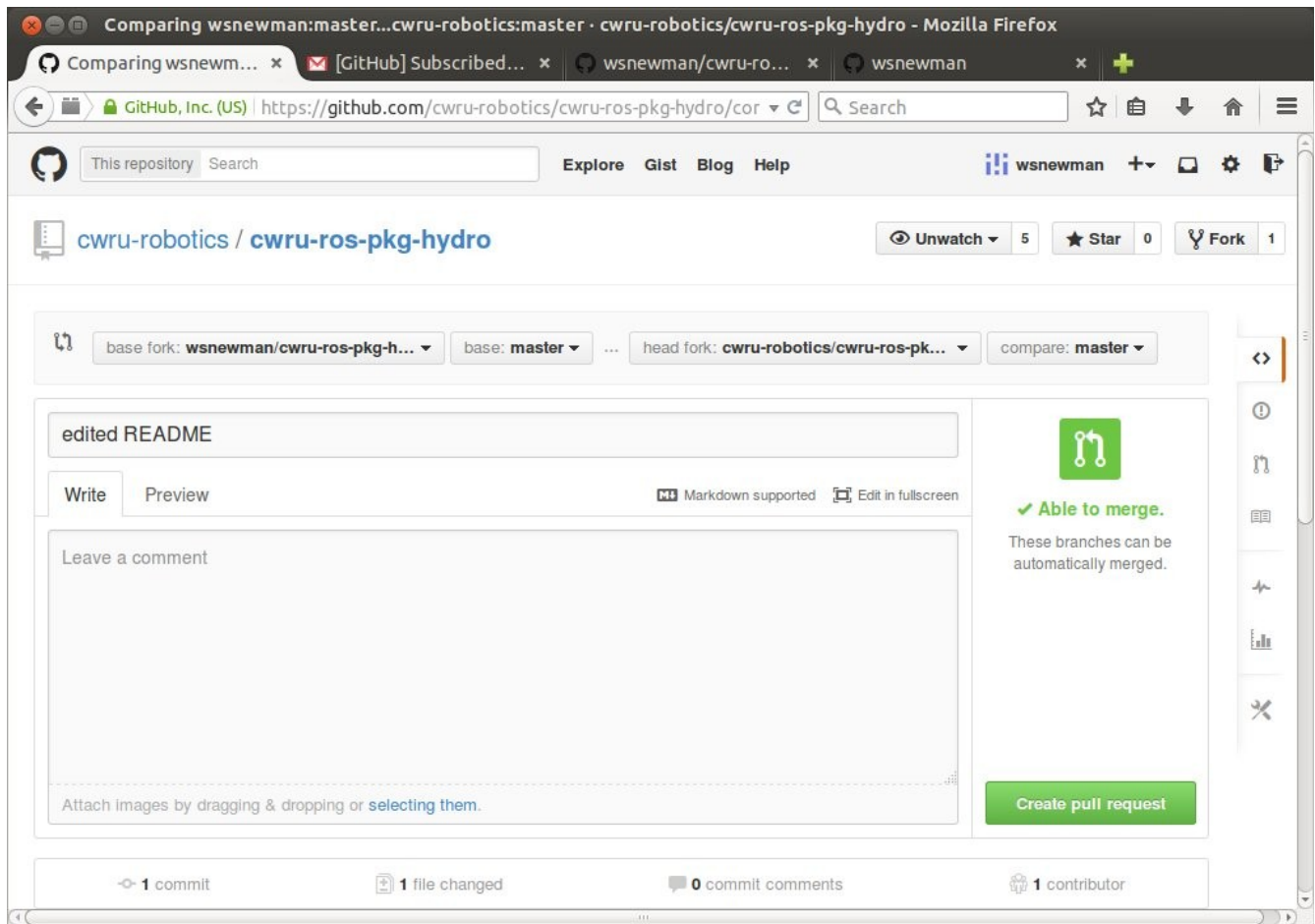
Illustration 9: creating pull request to update fork from shared repo

The "edited README" comment is copied from the update that was performed on the shared repo. You could add a comment here as well, if desired.

Next, press the button "Create pull request". This is the mechanism by which new content is incorporated in a repository (whether this request came via github or from some terminal of some computer). As you work in teams, such requests may come from team members. In the present case, we issued the pull request ourselves (and we will go ahead and approve our own request).

After issuing the pull request, you will get a screen like the following:
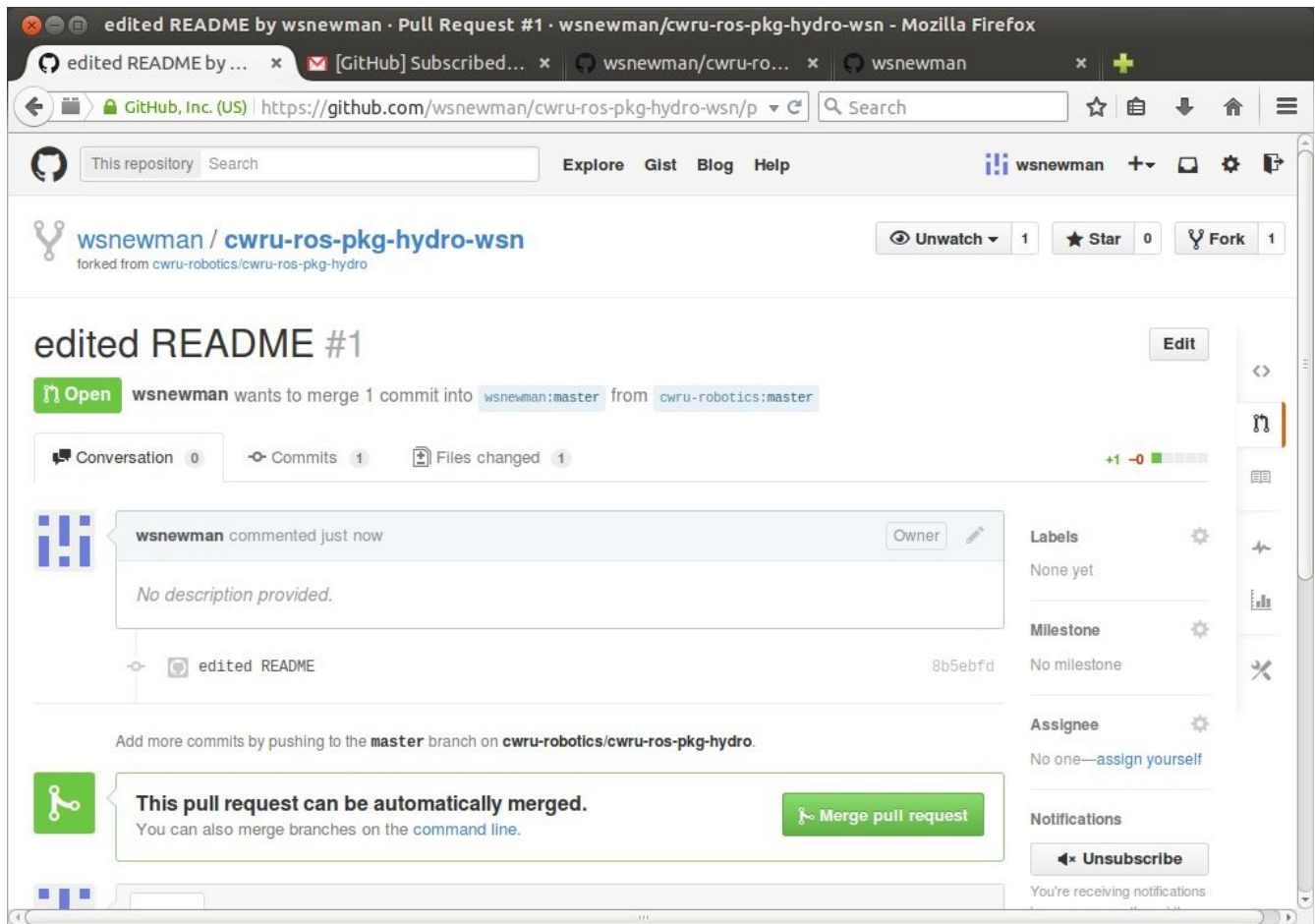
Illustration 10: 2nd step of pull request

The view above tells us that our forked repo has an "open pull request." That is, someone has requested merging new content into our repository. We could drill down into the details of the request, but in this case we have confirmation that this is an attempt to bring changes from cwru-robotics into our forked repository. We therefore go about acting on this request by performing a "merge." To do so, click the green "Merge pull request" button; you will be presented with another button "confirm merge." Clicking it yields the following:
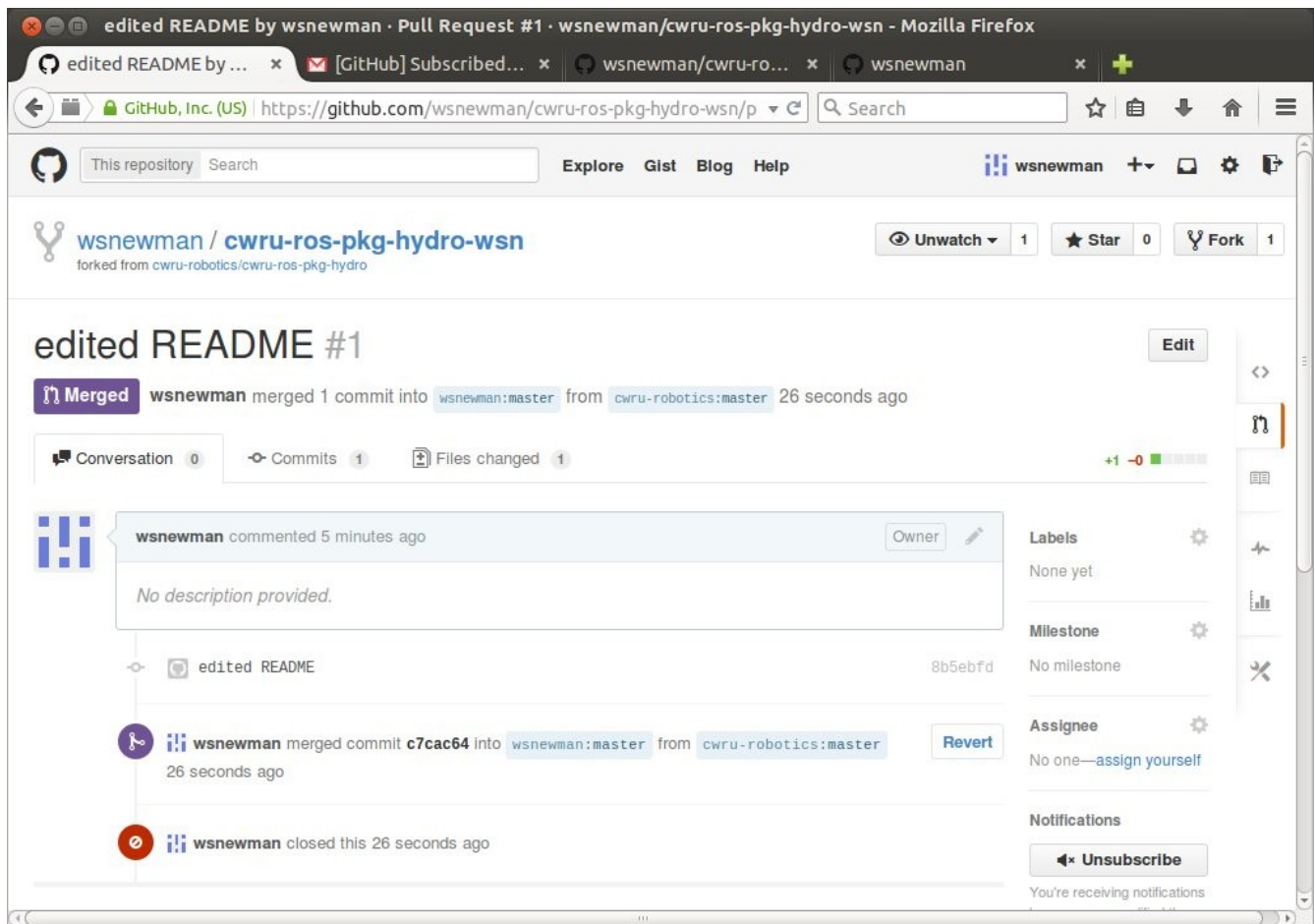
We see details about our merge, and our fork now contains all of the latest updates within the shared repository.

4) Updating your fork from your local clone:

As part of your work flow, you would ordinarily clone your fork to your local disk (see section 2), then make edits to this local copy. For all work that you want to save, you must "commit" and "push" the desired changes to your fork on github. Keep in mind, if you do not do this, you are making a conscious decision to discard all work that you have done that is not pushed to the repository. (This can be a good thing, if the day went poorly). For work that you push to your fork, you will want to make sure that, at least, you leave your repository in a state that can be compiled. You might save code that has bugs in it—but at least disable the CmakeLists entry requesting to compile that code. Then, at least, your team partners can expect that the repository is always able to be compiled, so they can make progress while you are still debugging some of your own contributions.

Assume I am on a computer that does not already have a clone of my personal repository. In a command terminal, I create a clone of my repo as in section 2. First, roscd (to navigate to the ros_workspace directory). Then clone a copy of your fork with:
    git clone https://github.com/wsnewman/cwru-ros-pkg-hydro-wsn.git
(where you will copy/paste your own fork's github address in place of the above orange text).

Next, "cd" to the resulting directory, in this case:
    cd cwru-ros-pkg-hydro-wsn.

The reason for requiring navigating into the repo directory is that "git" understands how to work with a repository by virtue of storing information about the repo in a hidden subdirectory of the repo called ".git". (You can browse this and explore the files "git" uses, if desired, but normally you would not interact with the info here). Note that you could have multiple repositories on your system. Git only understands how to perform updates by referring to the unique .git folder associated with the repository of interest. Thus, you *must* navigate to somewhere inside your repository before issuing new git commands.

From within cwru-ros-pkg-hydro-wsn (in a Linux terminal), issue this command to invoke a graphical tool for "git":
    git gui
You will see a new window like the following:
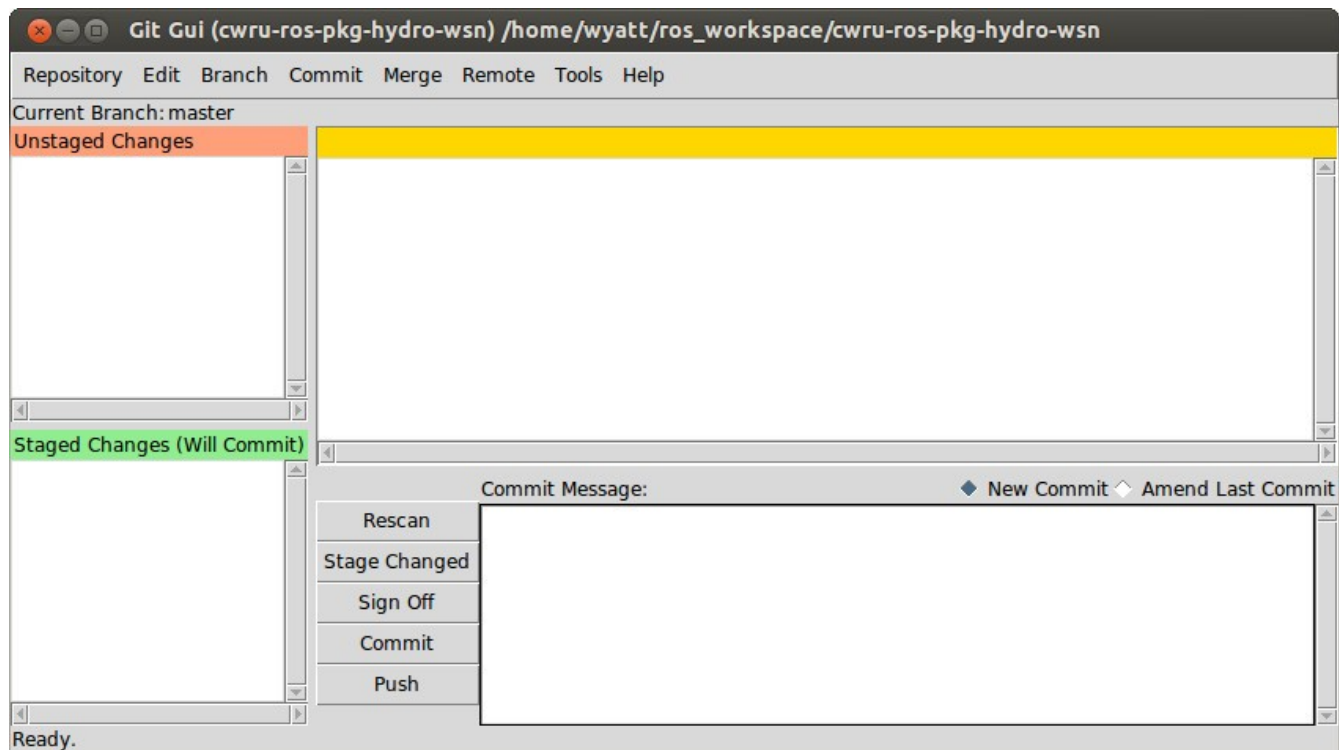
Illustration 11: GUI for git, showing no unsaved changes to this repo

The above GUI lists no unstaged changes. That is, the repository has not changed since the last commit.

Next, I make a change to the local copy of the repository. I issued the command:
        cwru_create_pkg test_pkg roscpp std_msgs geometry_msgs
which creates a new package, ready for me to add CPP code for new nodes.

Please note that I am *not* editing any of the packages that were provided for you as examples. I might make changes/additions to these packages in the future. If we both edit the same packages, you will get

"merge conflicts." As a matter of policy, only edit your own code in your own packages that are not part of the shared repository. It is fine to copy/paste example code into your new package; just do not edit within any of the packages you did not create.

Having created our new package, we can ask git gui to "rescan" to check for any changes. After clicking the "rescan" button, we see the following:
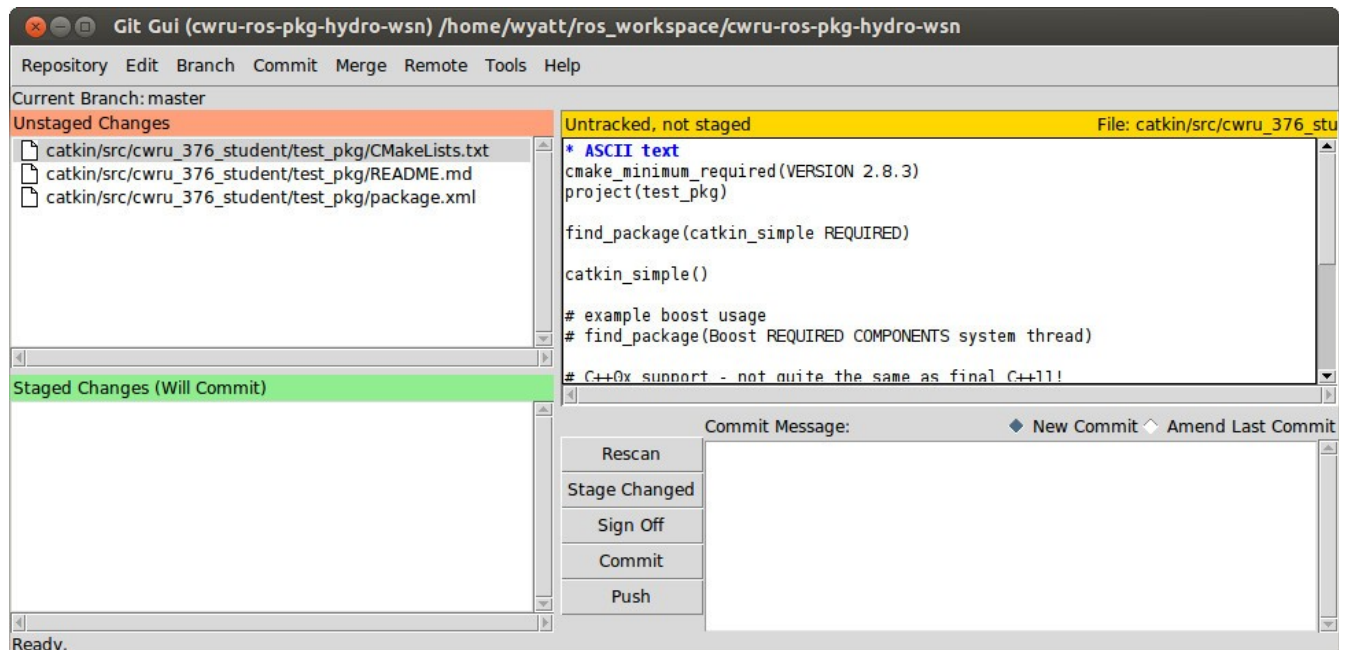
Illustration 12: Git GUI display after new package is added, and after clicking "rescan"

We see from the above display that there are three new files (or three files that have been altered). Having clicked on the name of the first file (in the upper left box), changes to that file are displayed in the upper-right box. We decide that all three of these files should be saved to our fork. By clicking on the folder icons of the upper-left box for all three files, git gui then looks like the following (including having entered a comment in the "commit message" box):
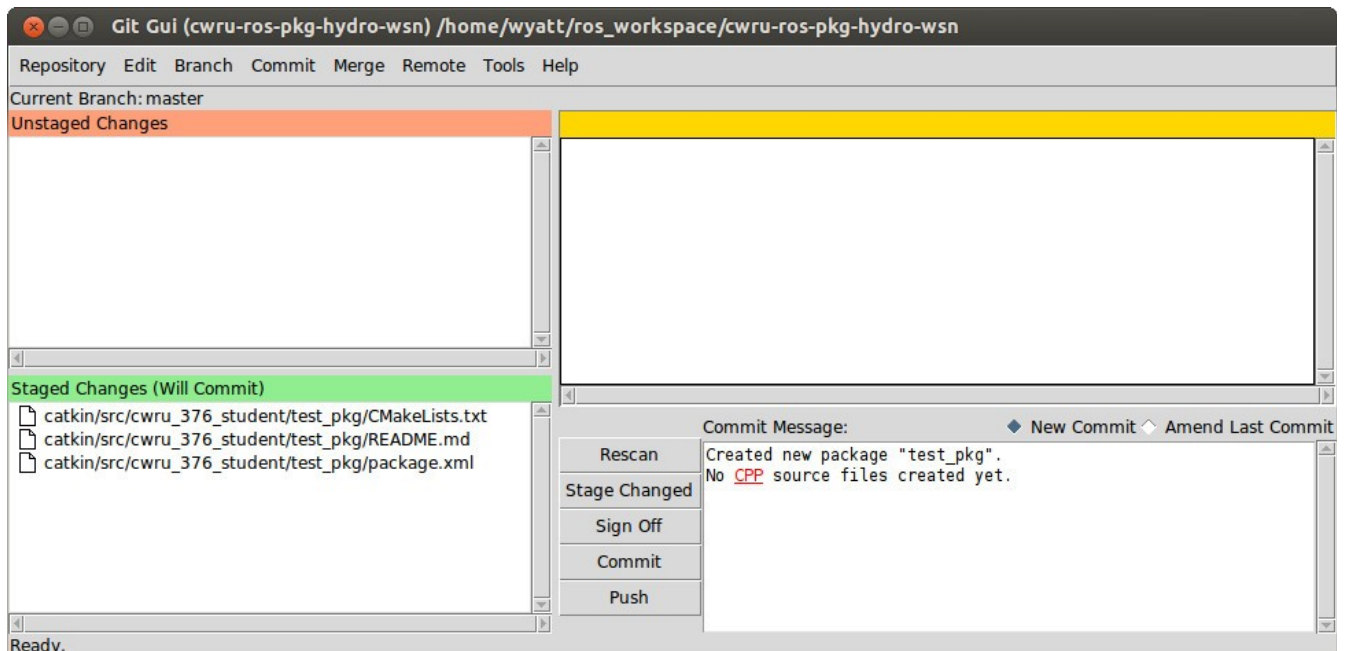
Illustration 13: git gui display prepared to commit 3 files

The 3 selected files are "staged" to be committed. (They are not saved yet). We enter a "commit message" that explains or reminds us the purpose of this commit. This is not just a good idea (as you may need to "roll back" to a known good prior state, if things get out of hand)--it is a requirement. You must enter some message to commit.

We next push the "commit" button, and the git gui display changes to this:
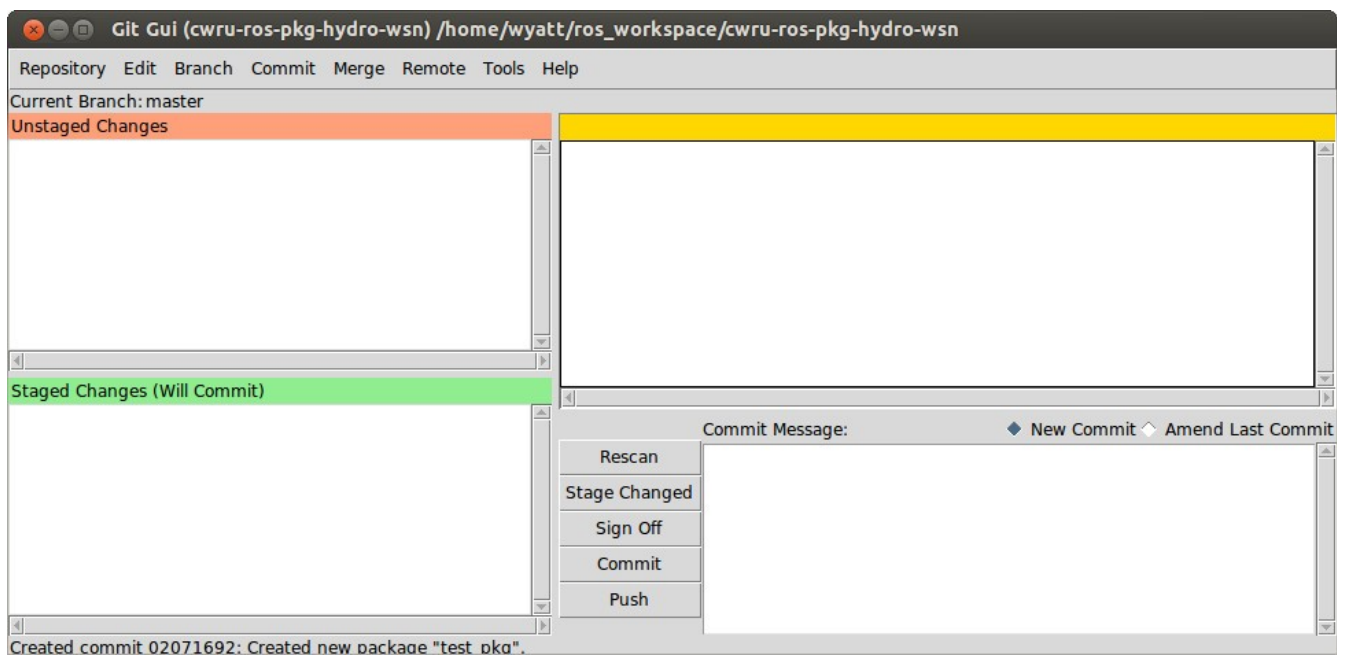

Illustration 14: git GUI after clicking "commit" but before "push"

This view is almost indistinguishable from Fig 11, except for the message at the bottom noting that we

have a "commit."

You can continue to edit and make more commits. In fact, you likely will have commits frequently (e.g. a dozen commits per hour). At some point (and certainly before you logoff), you want to "push" your commits up to github. You do this by clicking the "push" button. You will see a pop-up window like the following:
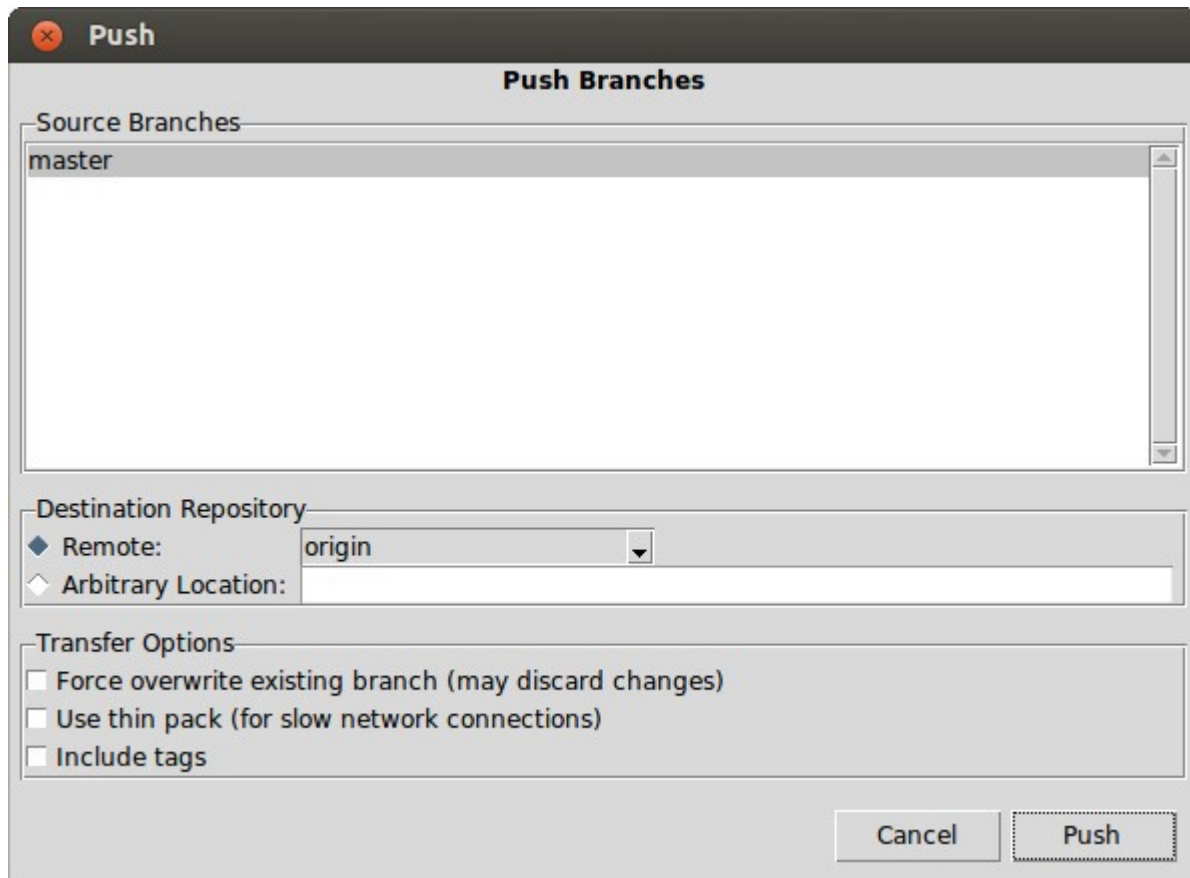

Illustration 15: pop-up window appearing after clicking "push." Click "push" again.

Various options are offered, but you should not need these. Merely click "push" again. You will be prompted for your github login name and password, appearing like that image below:
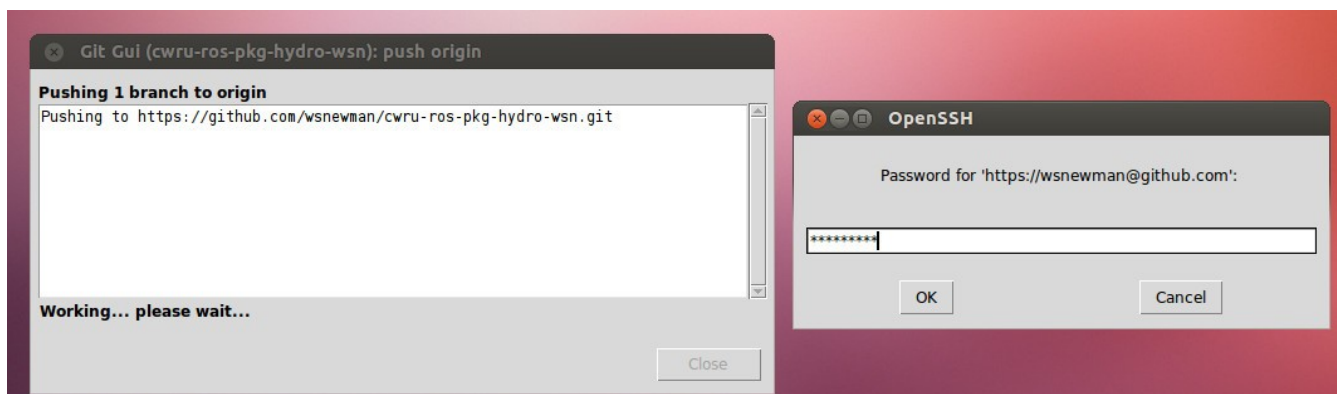
Illustration 16: Pushing changes, with prompt for github password

If you do not see the pop-up prompting for login name (and then password), look for it. It may be hiding behind another window. If you cannot find it, click "cancel" and click "push" again. After entering the password and clicking "OK", you should get a response like this:
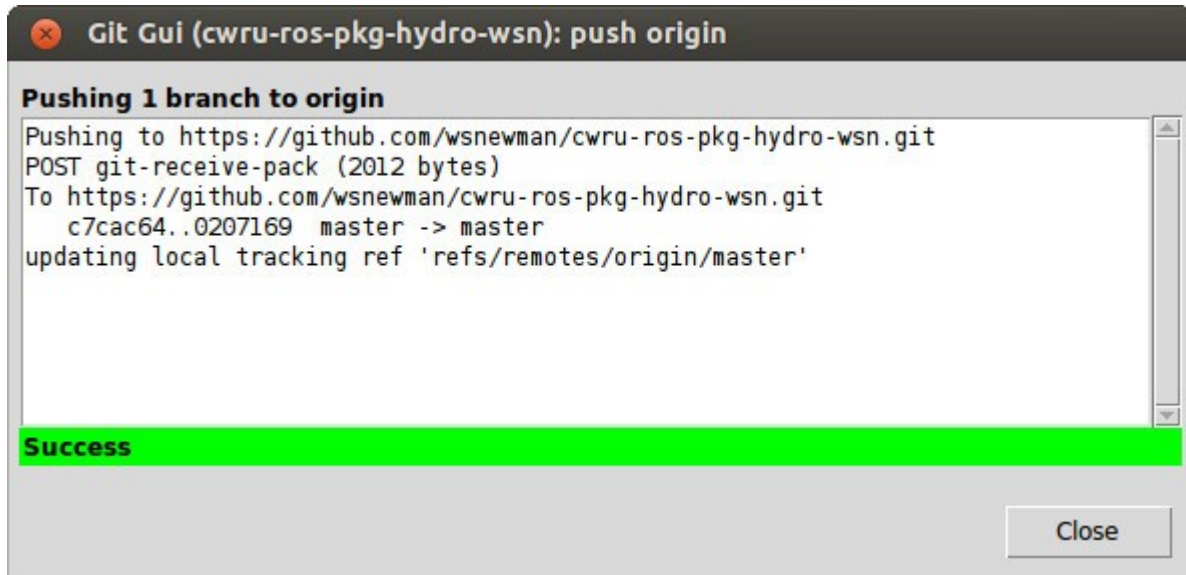

Illustration 17: Response after a successful "push"

You can close this window. Your local repository is now in sync with your forked repository on github.

By the way, you can close git gui and restart it at any time—whenever convenient. You do not have to keep it "alive" while you are working, if you find it distracting.

Returning to the browser view of the fork, the display has changed to the following:
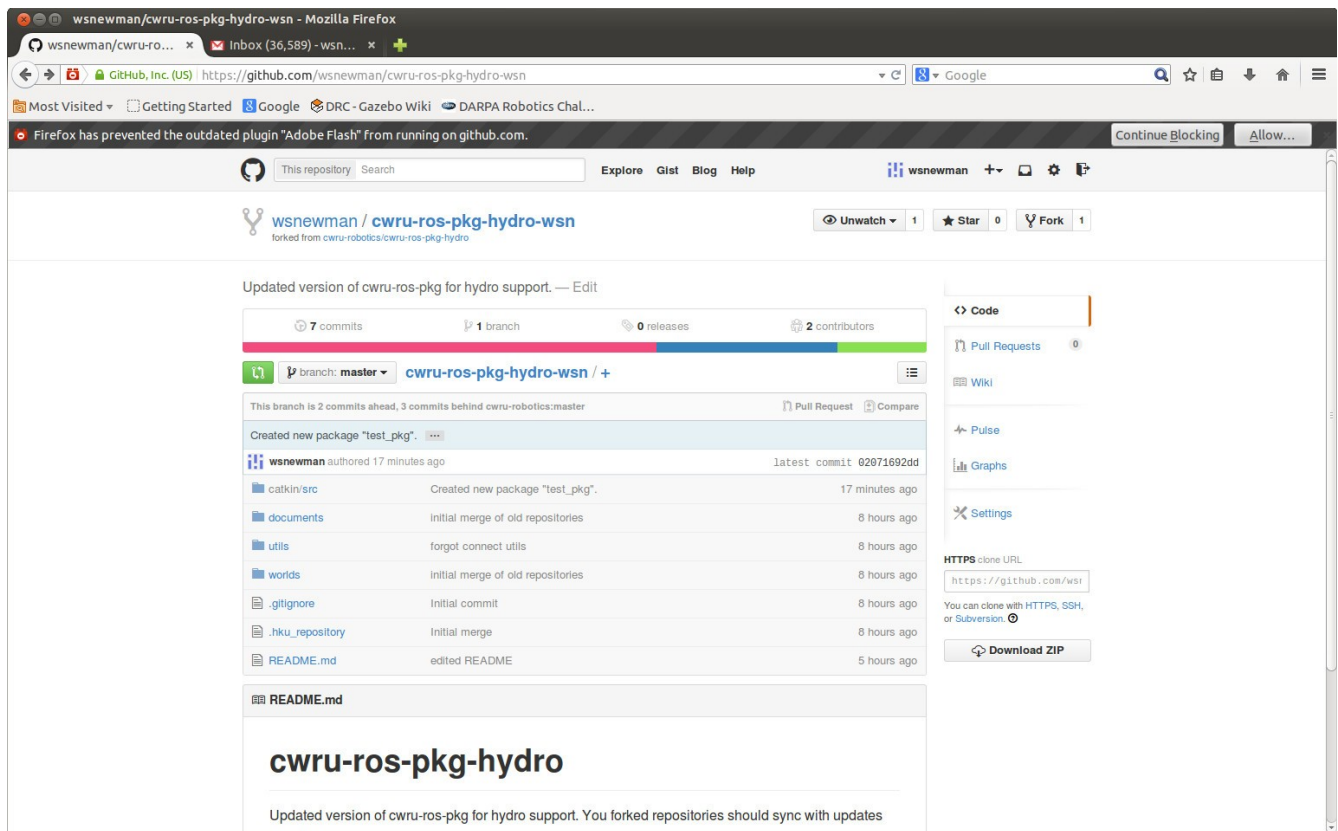
Illustration 18: fork after push from clone. Fork is now 2 commits ahead and 3 commits behind cwru-robotics

Note that the view claims that the for, is both 2 commits ahead of and 3 commits behind cwru-robotics. Clicking on the "compare" button yields the following:
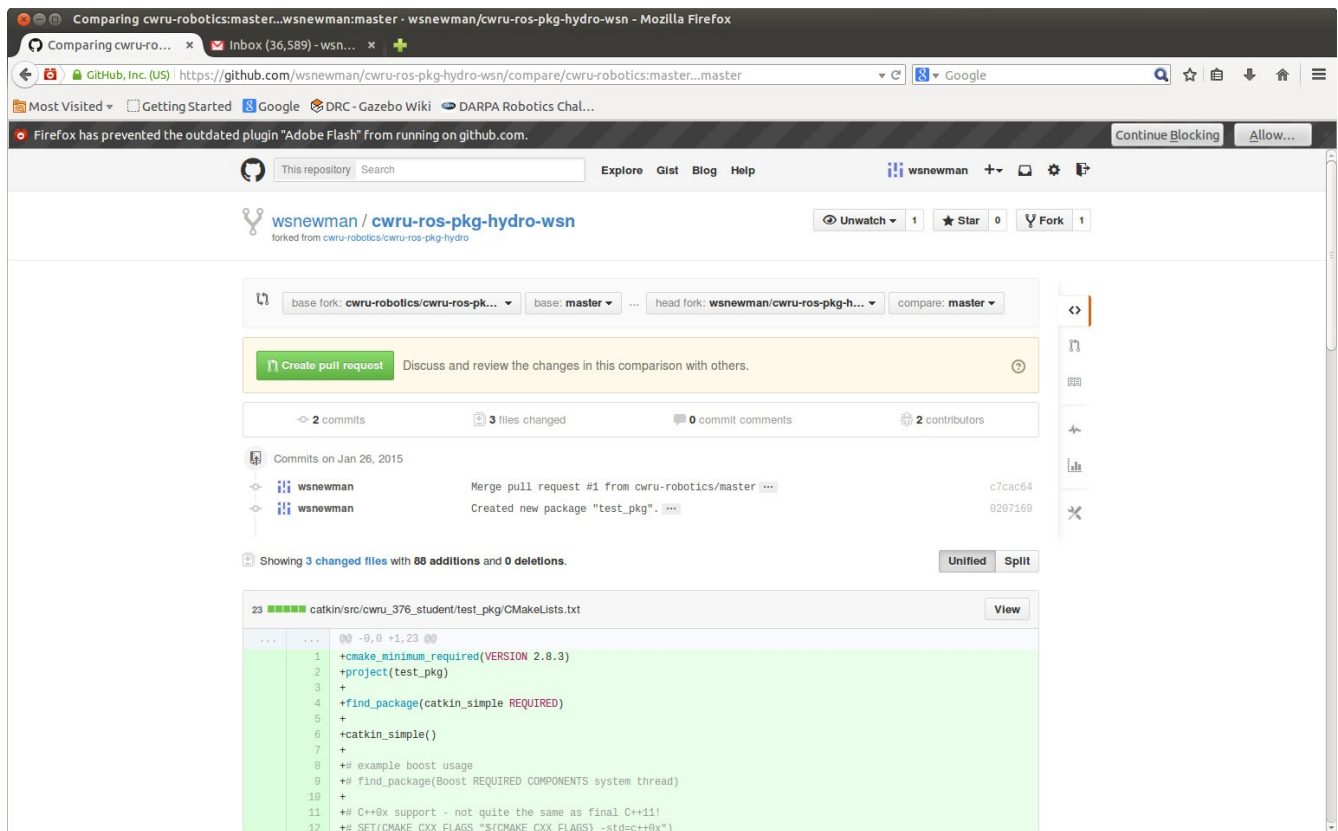
Illustration 19: github view of fork after push from clone, and after clicking "compare"

Much of the view is truncated, but scrolling down would reveal every line of code in the fork that has been changed with our recent push, and which is not part of the shared repository in cwru-robotics. This will be normal. You should have code that is not part of the shared repository.

However, we may be concerned about changes to the shared repository that are not yet part of our fork.

We want to reverse the order for the comparison. Near the top of the page, there is a drop-down menu labeled "base fork:", which is currently set to the shared repo. Use the drop-down menu to change this to your personal repo. The screen then looks like this:
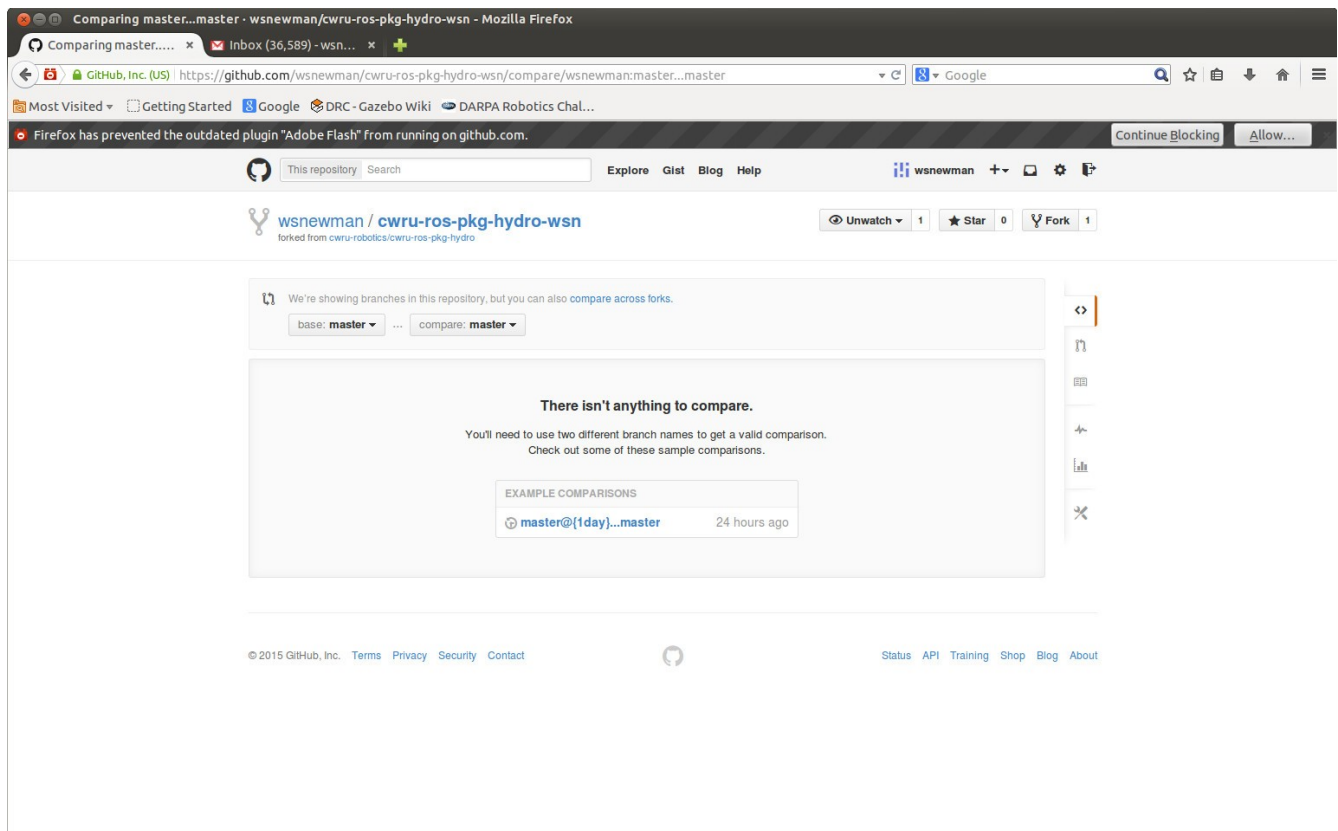
Illustration 20: reverse compare: display after changing the base repo

Next, click on the link to "compare across forks." In the subsequent display, you will be able to choose the "head fork:" via a drop-down menu. Choose the shared (cwru-robotics) repository. The view then looks like this:
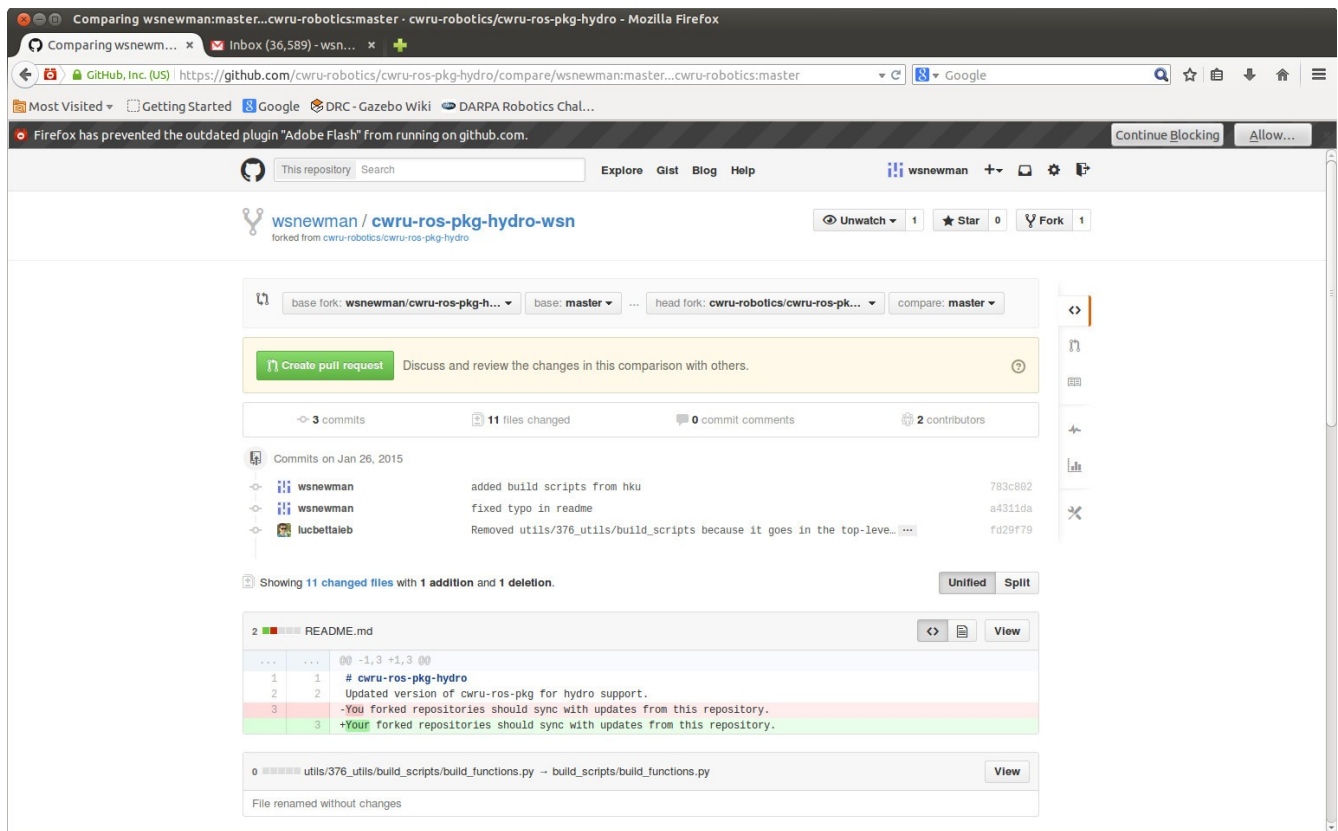
Illustration 21: reverse compare after swapping from/to repos

We see that there are 3 commits and 11 changed files in the shared repository. (Coincidentally, Luc was also committing changes while I was documenting this process). There was a small change (a typo correction) to the README file, which is highlighted by the line removed (in pink) and the line inserted (in green). Additional committed changes are listed, and these are visible if one scrolls down the display.

Having reviewed these changes, we decide we want to merge them into our personal repo. This is done as described above in section 3, figures 3 through 10: click: create pull request, create pull request, merge pull request, confirm merge. The fork now includes the latest changes from the shared repo—as well as changes made from the local cloned repo.

5) Resolving merge conflicts:
You should avoid editing any of the files from the shared repo. Further, you should avoid editing any files or packages created by your collaborators. If you follow this policy, you will avoid merge conflicts.

However, as you work in teams, you may find that you and a partner have both edited the same file. Here is an example.

User 1 clones a repo, then edits one line of CmakeLists.txt file in demo_pkg, adding the line:
cs_add_executable(example src/example1.cpp)

User 2 clones the same repo, and user 2 also starts editing the same CmakeLists.txt file, instead adding the line:
cs_add_executable(example src/example2.cpp)

User 1 commits and pushes her code first. User 2 is now out of sync.

User 2 then attempts to do the same, but the "push" command issues this error:
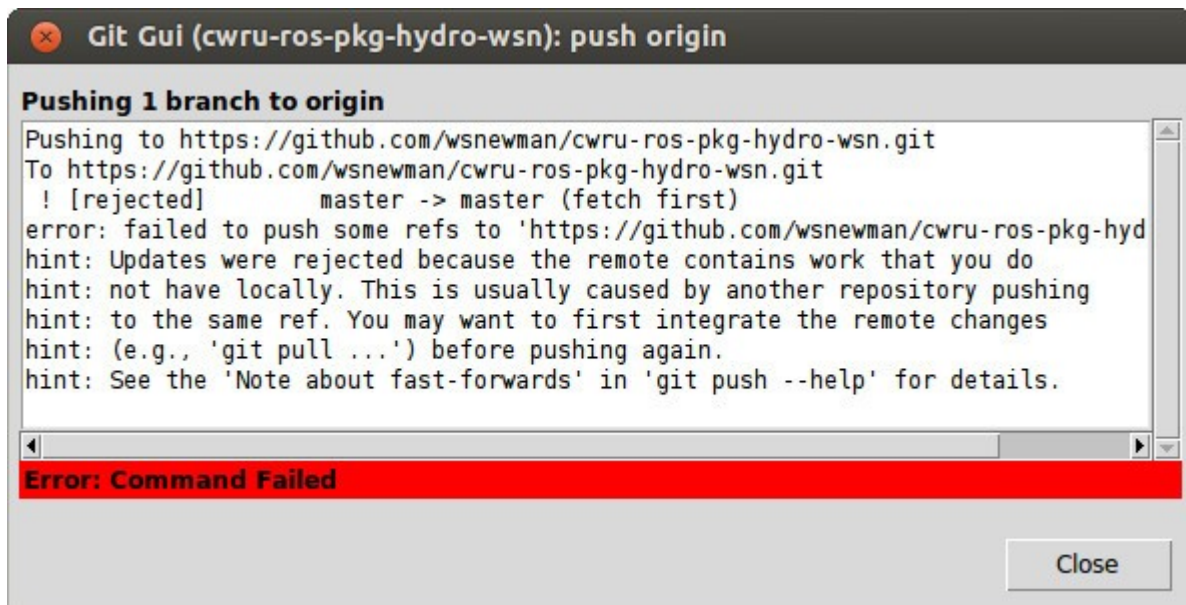
Illustration 22: push error: local repo does not contain latest changes in fork

We do not want to re-clone the fork, as this would destroy our latest work. Instead, we can attempt to update our local repository from the fork. This can be done via git gui, but it is easier via command-line arguments for "fetch" and "merge".   Here is the "git fetch" command:
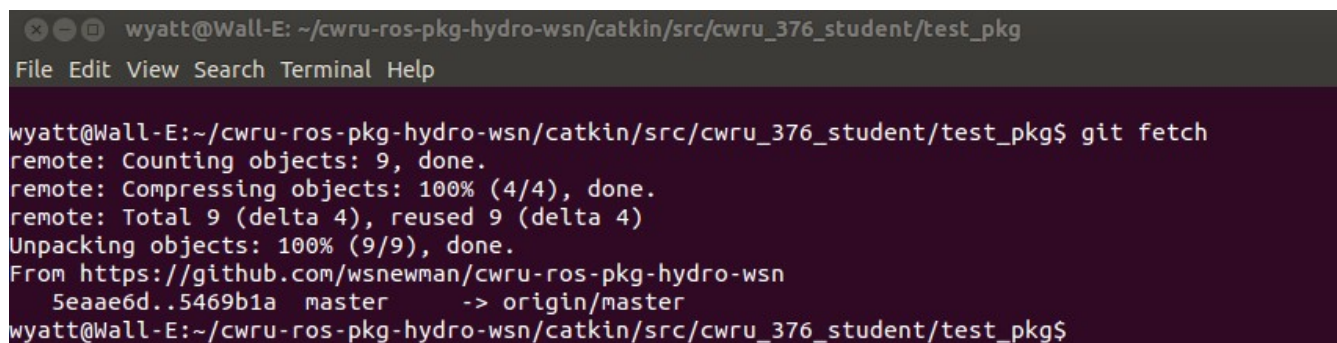
Illustration 23: issuing "fetch" command from terminal, navigated to within repo. Fetch is successful.

Followed by a "merge" command, which produces the following error response:

Illustration 24: Attempt to merge in the "fetched" repo--merge conflict error reported

We need to resolve the conflict before we can proceed. A tool to do this is:
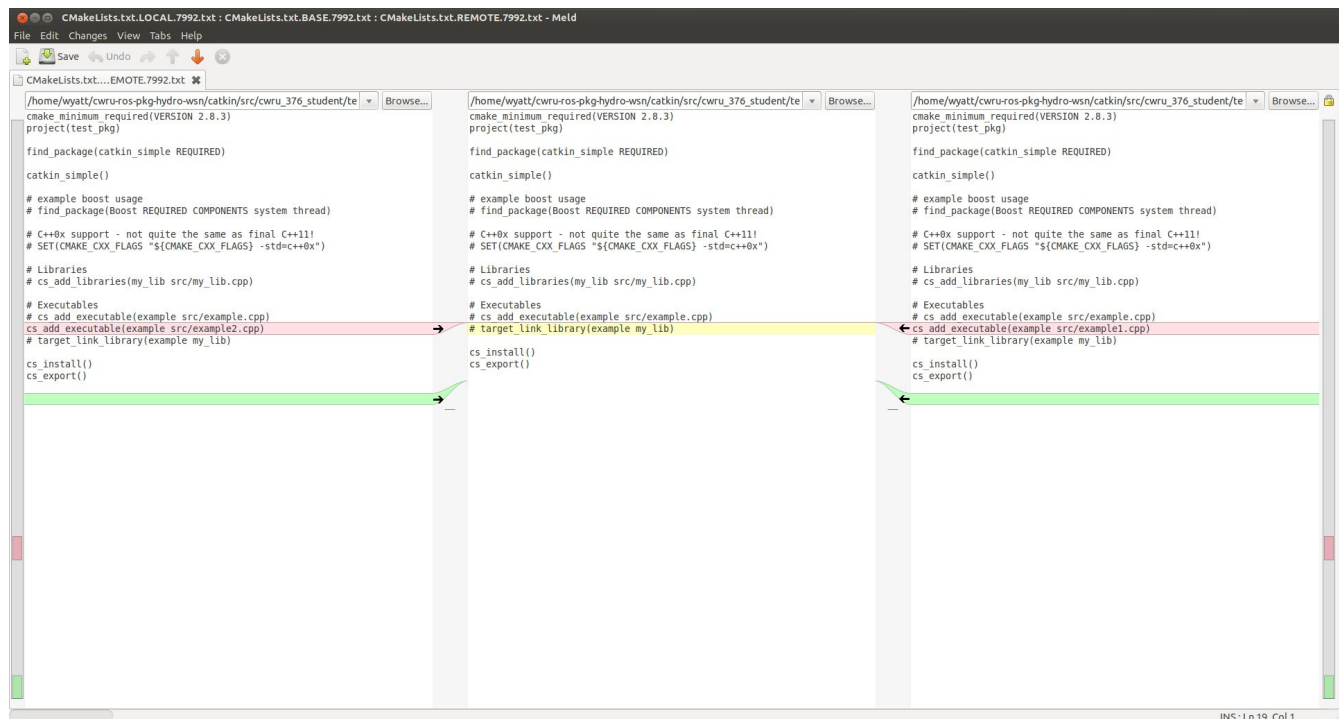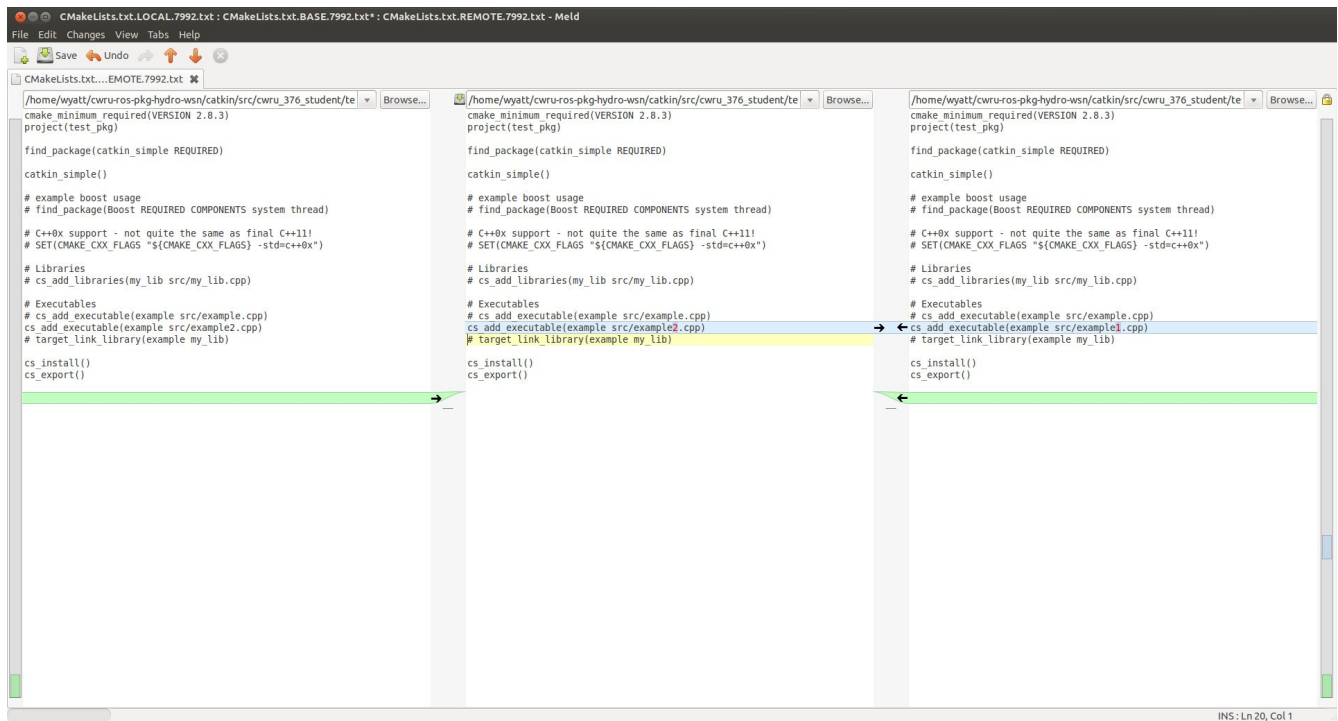   git mergetool
Which brings up a GUI, in this case

Illustration 25: view of mergetool: left is user 2's changes, right is user 1's changes. middle will be the resolution

We see in this view the file before either user1 or user 2 edited it. We see on the left the edit user 2 attempted, and on the right is the edit user 1 performed (highlighted).

I choose to click on the arrow next to user 2's edit, producing the following result:

Illustration 26: mergetool result after choosing option from the left

Note that this will wipe out user 1's edit. You can, alternatively, make changes that accommodate both edits (if that is compatible). Else, you will need to work out with your partner which change to accept.

We choose: file->save from the mergetool menu, then quit the mergetool program.

User 2, returning to git gui, sees the modified file ready to be saved.
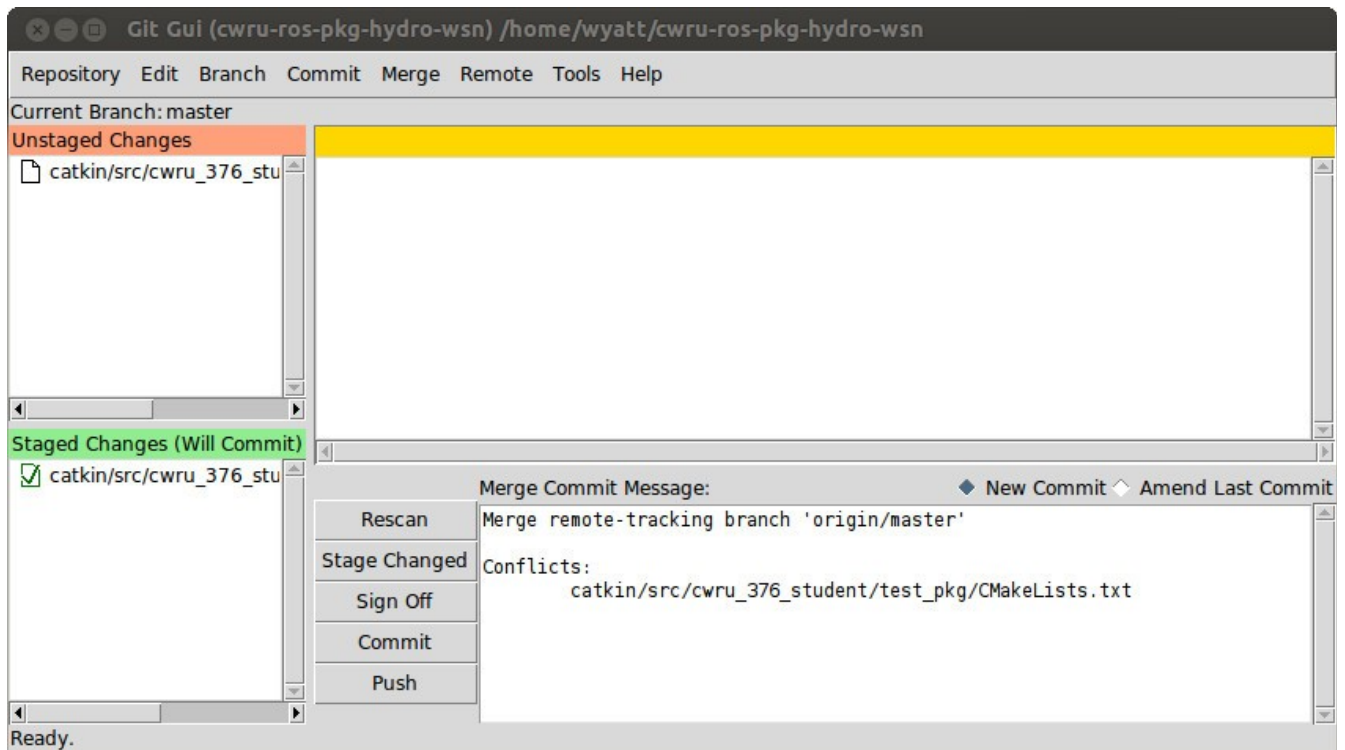
Illustration 27: git gui after mergetool

Note that a message has been inserted recognizing a merge conflict was addressed. After committing and pushing, we get a success message from github.

When user 1 next performs either a new clone, or performs a fetch and merge, user 1 will find her original edits from this document are gone and replaced with the merge conflict resolution performed by user 2. Both users are now back in sync with the repository.