Using “git” and bitbucket

Wyatt Newman

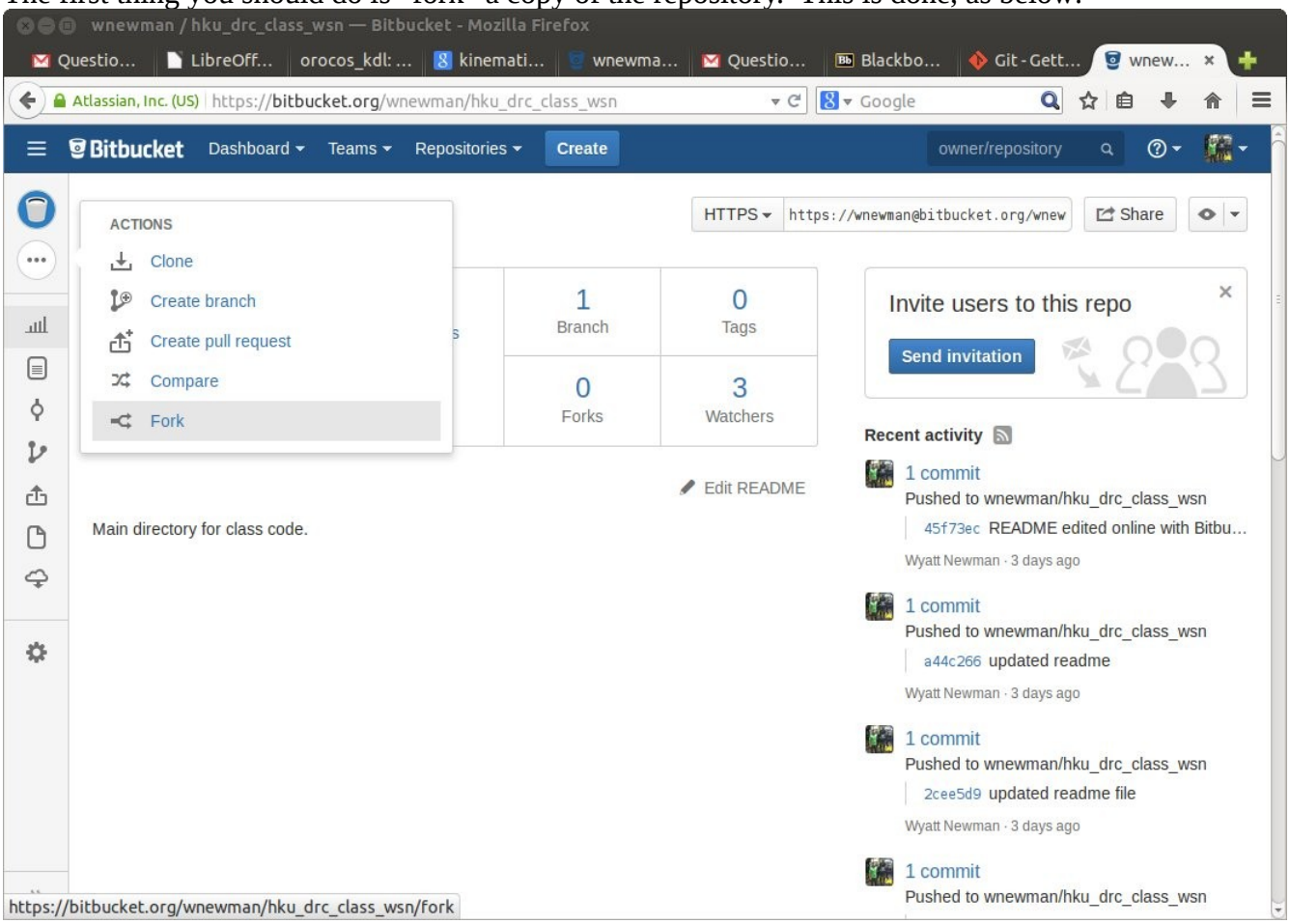
September, 2014

In this course, we will be using “bitbucket” and “git” for version control (see http://git-scm.com/book/en/Getting-Started). You will need an account on bitbucket, which you can do on-line at bitbucket.org. You should register for your account using your student e-mail, as this will allow a larger number of free additions to the “team” for sharing our repository.

Once you have provided your bitbucket user ID to me, I will enable your access to the code repository, and you will receive an invitation to access it.

Forking and Cloning a Repository:

The first thing you should do is “fork” a copy of the repository. This is done, as below:

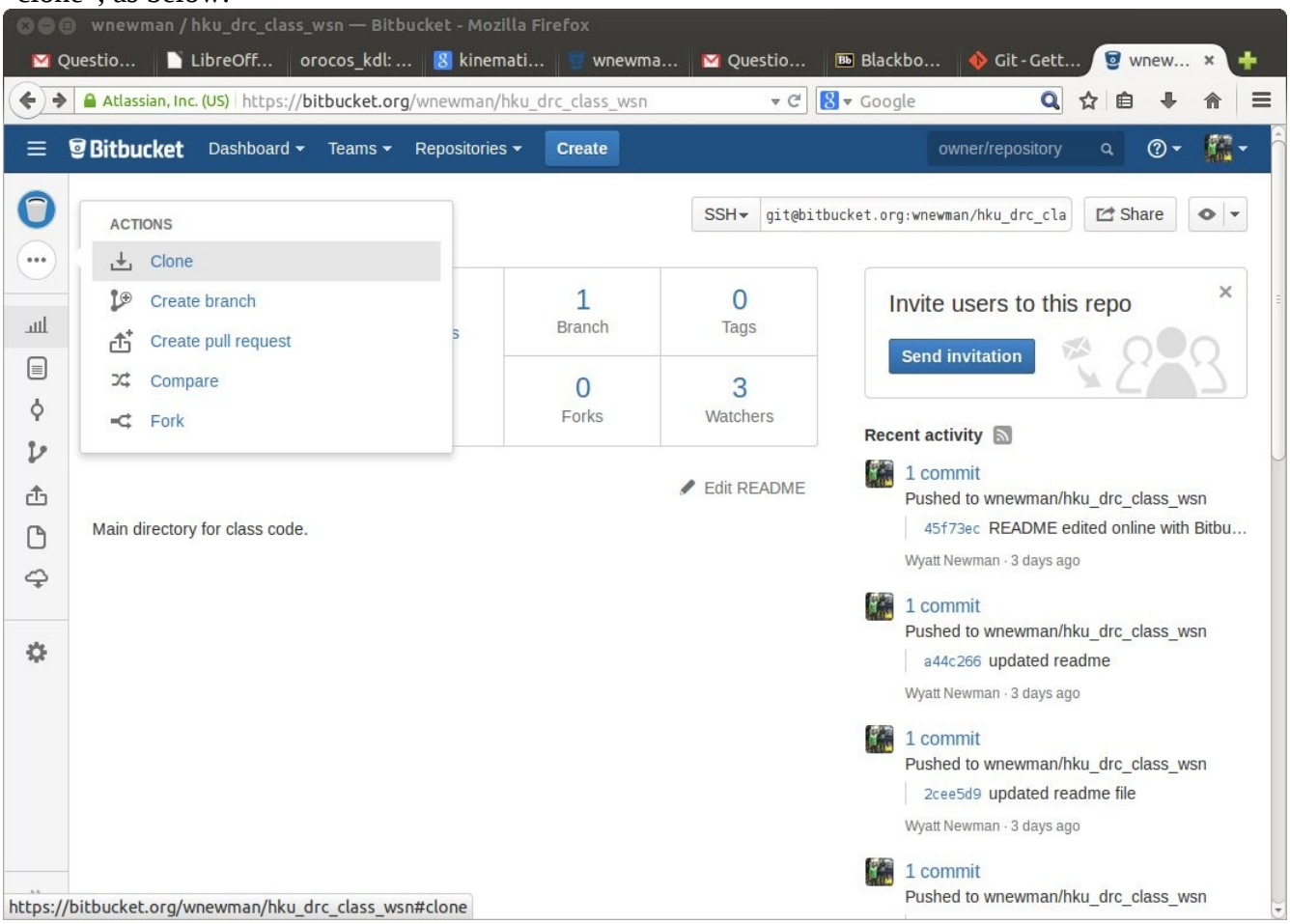


After clicking “fork”, fill out the pop-up form to give your forked copy a name. Please include some identifier in your fork's name indicating your ownership of it.

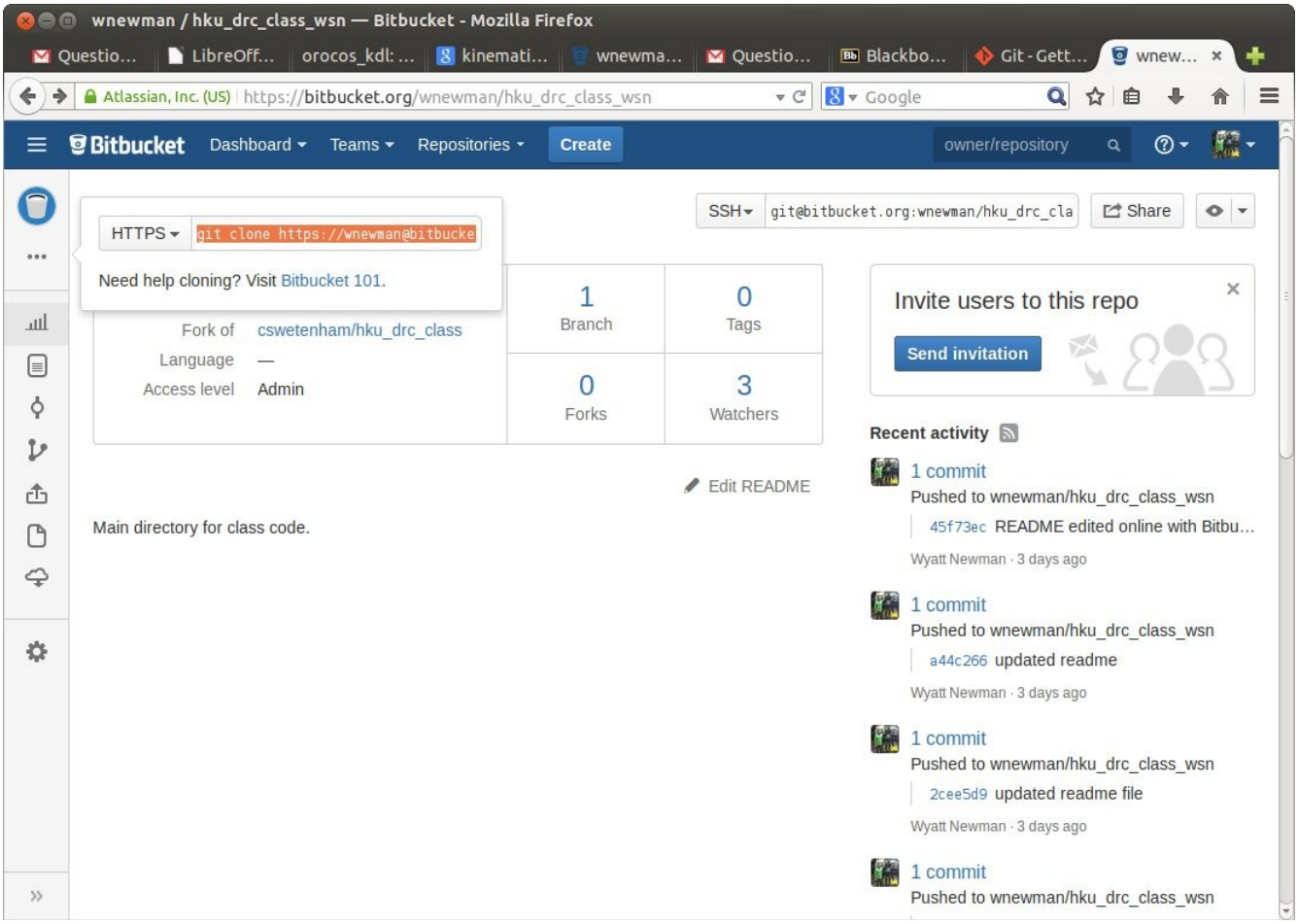
This will create a new repository (a copy of the class code) that is stored on bitbucket. You will only need to do this *once*. Thereafter, you will regularly access this copy (“fork”) of the repository.

The next step is *cloning*. Invoking this step differs, depending on whether you will be re-using the same computer file system for your development work or not. If you are running in a guest session (for

which your files will be deleted when you log off), then you should “clone” your repository every time you login. “Cloning” will make a full copy of your repository from bitbucket to your local machine. To do so, use your browser to get to bitbucket, select and click on your fork of the repository, then click “clone”, as below:



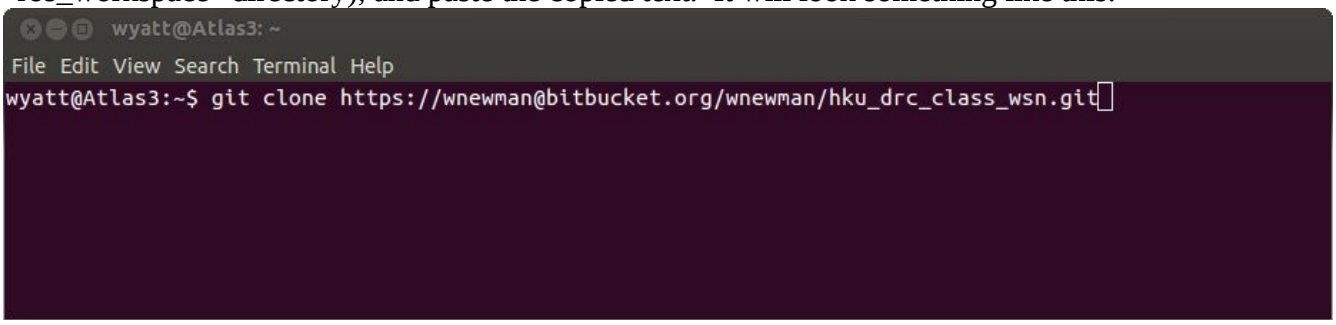
Clicking “clone” will bring up something like the following:



Note that this display includes some highlighted text. There is a menu option next to the highlighted text, offering you choices of HTTPS or SSH. You should use HTTPS (unless you have gone through the steps of setting up SSH; see the class repository's wiki for details).

Copy this text into your buffer (e.g., right-click->copy).

On your local machine, open up a terminal, navigate to a desired directory (normally, your "ros_workspace" directory), and paste the copied text. It will look something like this:



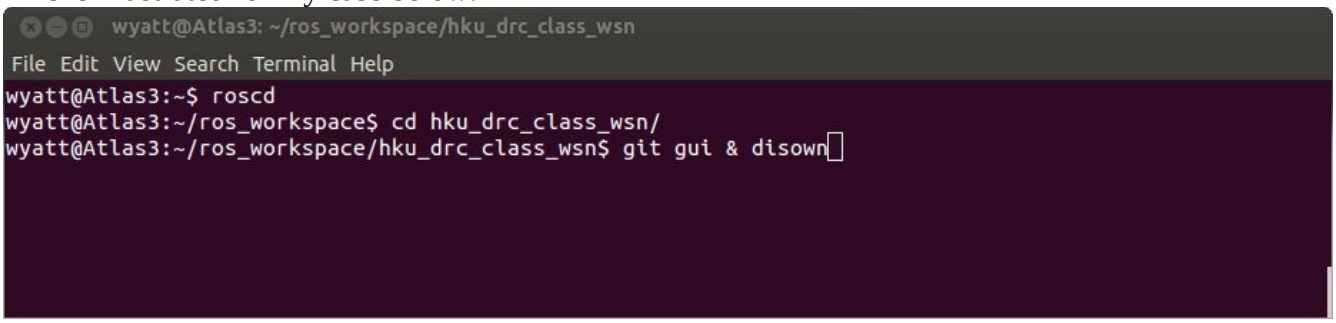
When you hit "Enter", this will start the download process, making a complete copy of your repository on the local machine.

If you will be returning to the same machine, or if you have persistent access to your filesystem, then you will only need to do the cloning operation once. If you are using a guest session, you will need to

clone the repository every time you login.

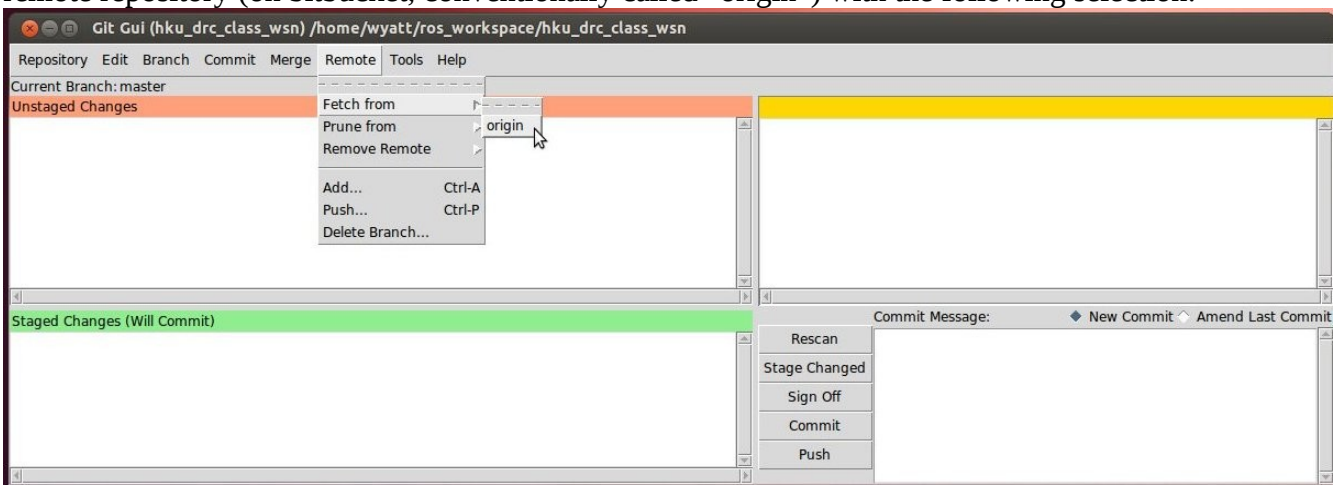
Using Git Gui:
If you are returning to work with your existing clone of the repository, the first thing you should do when starting your work session is to sync your local copy of the repository with your remote copy of the repository. If you are the only person working with this repository, it *should* be the case that the local and remote versions are already in sync (as this is the state you should leave it in before you log off). If you are working collaboratively, there may well be other changes to the repository since you last updated. In either event, you should sync the versions before starting your new work. To do this, I recommend using the graphical tool "git gui".

To start up git-gui, open up a terminal and navigate to your repository. That is, change directory until the current directory is somewhere inside your team code. Then enter: git gui
Optionally, enter: git gui & disown
The second version of the command will launch git gui, but then free up the terminal for other work. This is illustrated for my case below:

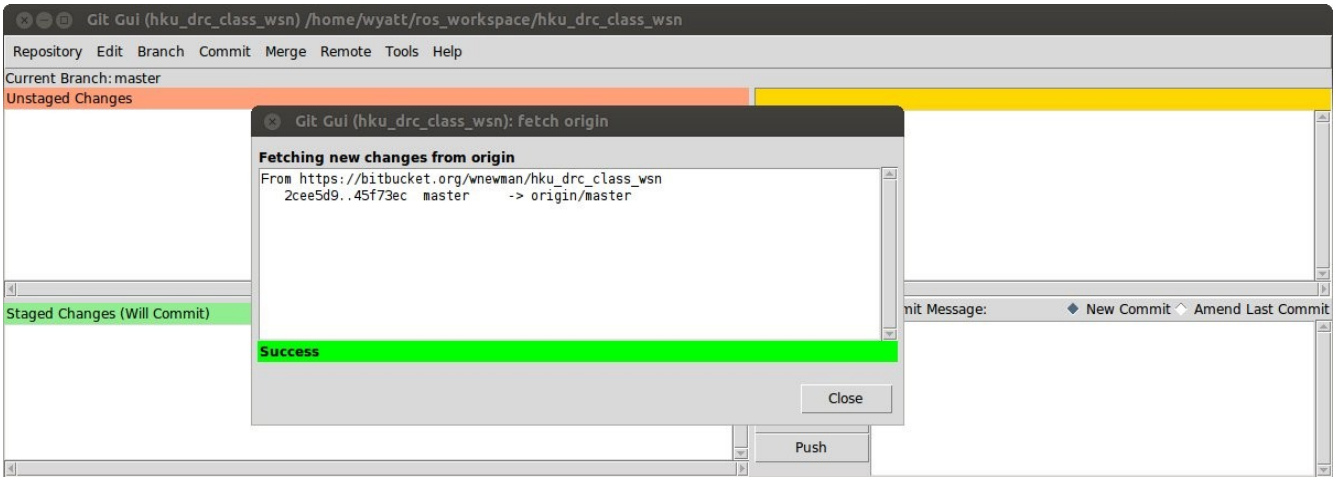


In the above, the first command "roscd" navigates to your ROS workspace (normally called "ros_workspace"). The second command navigated to the sub-folder with the same name as my bitbucket repository, which is the same name as my local clone of this repository.

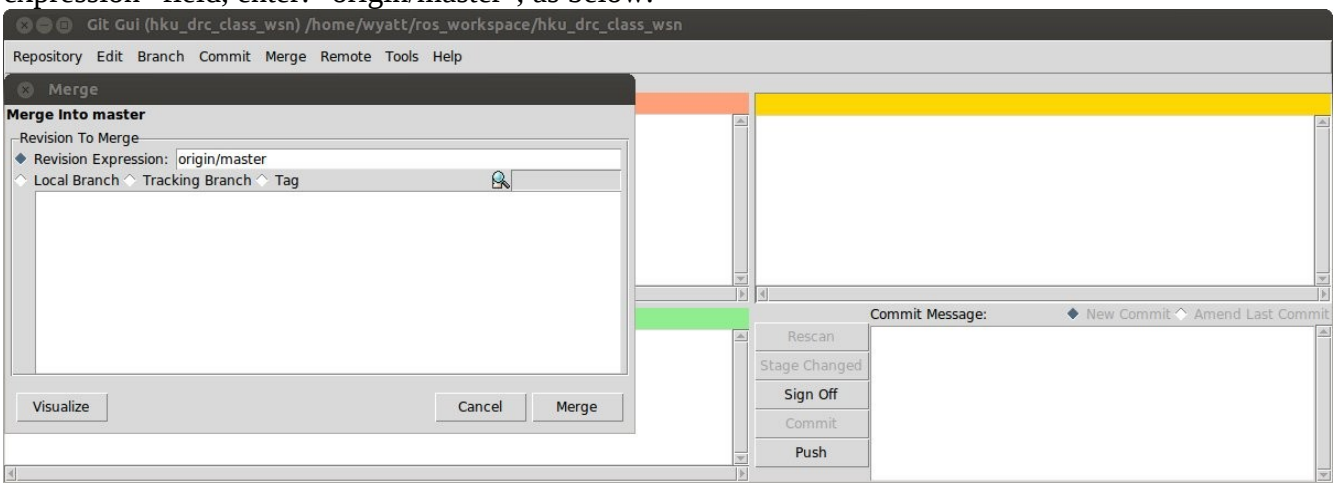
Once the current directory is within a valid repository, you can launch git gui. Fetch a copy of the remote repository (on bitbucket, conventionally called "origin") with the following selection:



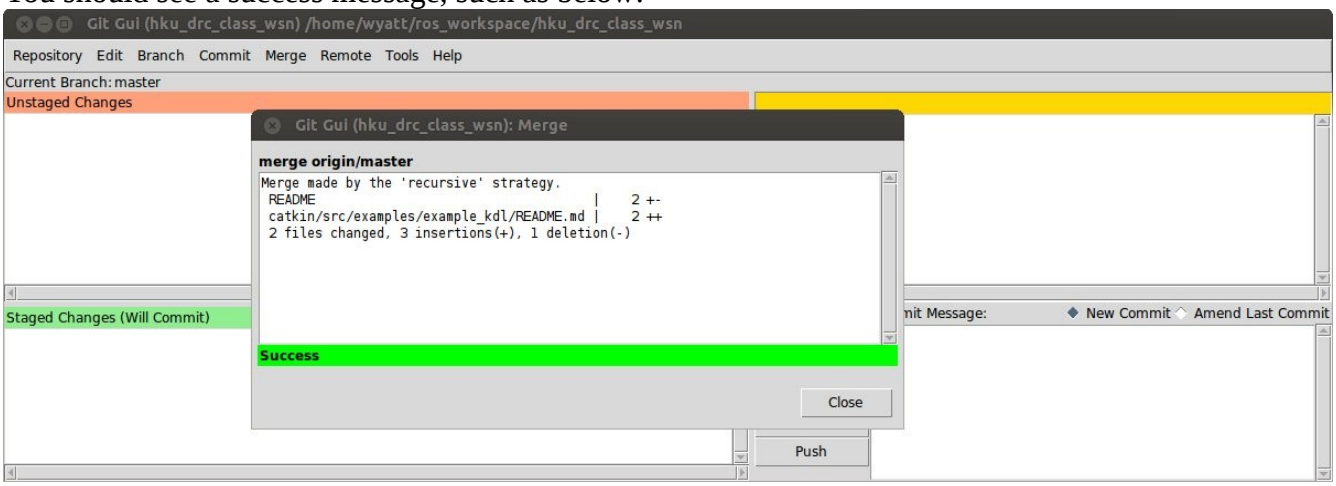
You will be prompted for your bitbucket password. After the fetch operation, you will see a success message, something like the following:



After fetching the remote repository, “merge” it into your local copy. To do this, select the “merge” menu along the top menu bar of git gui, and select the option “local merge”. In the “revision expression” field, enter: “origin/master”, as below:

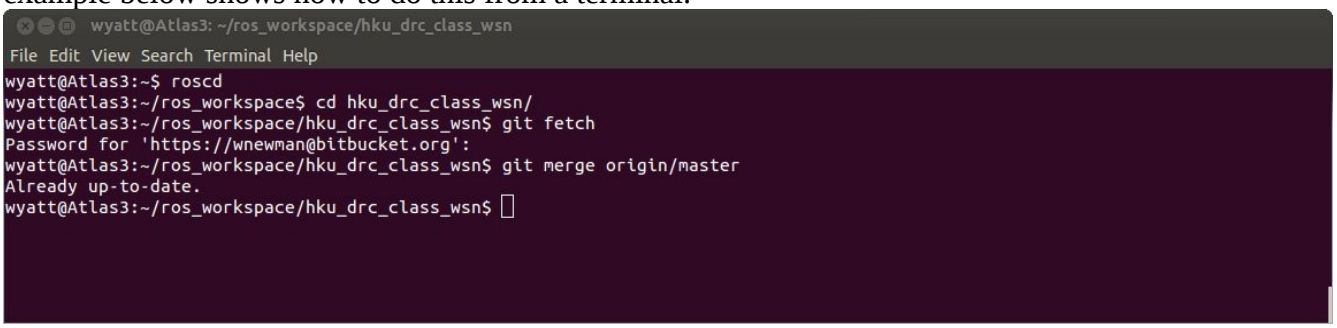


You should see a success message, such as below:



At this point, your local repository is up to date, i.e. in sync with the remote repository.

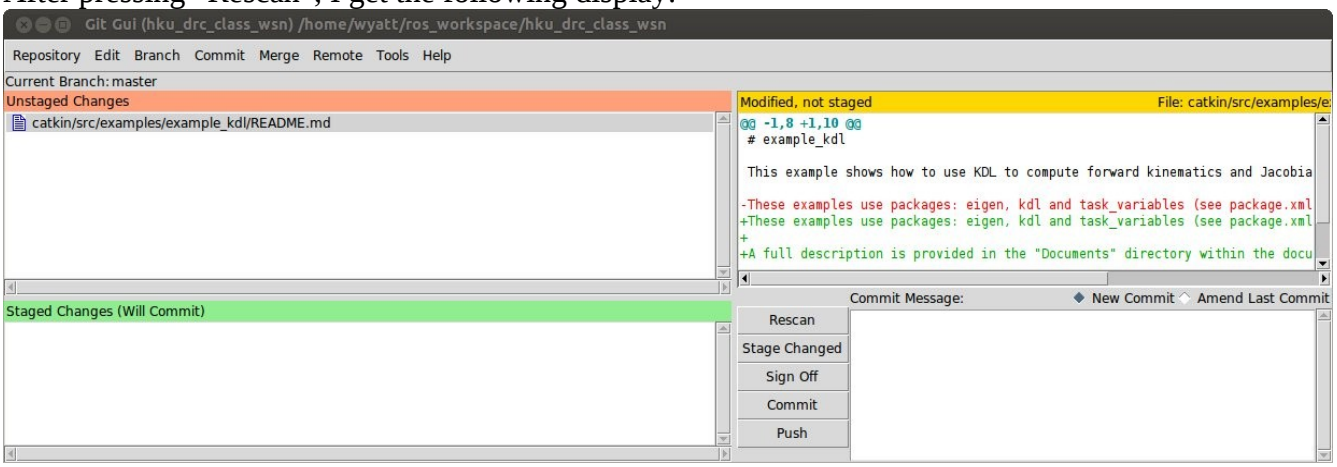
The steps to fetch and merge the remote repository are actually simpler as command lines. The example below shows how to do this from a terminal:



First, make sure the terminal has its current directory set to be within your repository. In the above example, this is done with `roscd` followed by `cd hku_drc_class_wsn`. The remote repository is then fetched with the command: `git fetch.` The merge is then performed with the command: `git merge origin/master`.

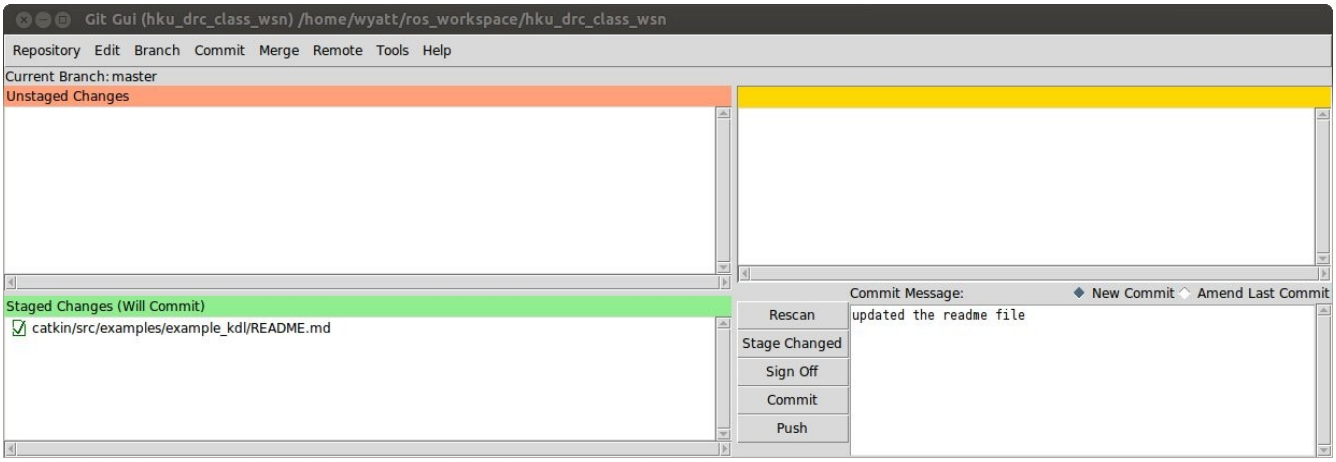
After bringing your local copy up to date, you can begin development work.

As you make progress, it is a good idea to "commit" changes that you consider valuable. "Commits" may happen many times during a work session. To do this, from git gui, click the button "Rescan". You will see a list of files that have been altered (upper left panel) and you can review the changes that were made (upper right panel). As an example, I edited a "readme" file in the package "example_kdl". After pressing "Rescan", I get the following display:



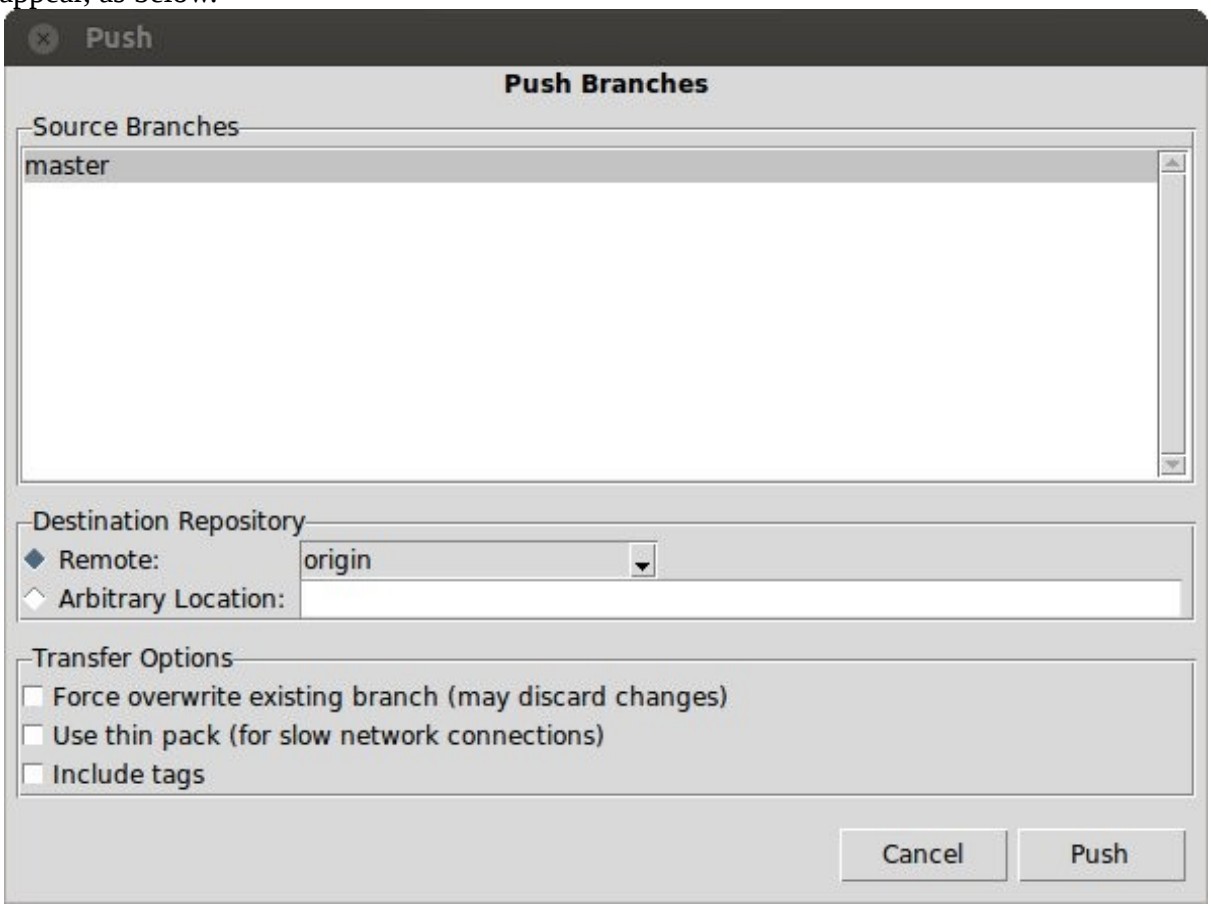
This shows that I have altered the expected file, and it shows me the changes made. I choose to commit these changes. To do so, first click on the file name. If there are multiple files altered, you can click on multiple files—but it is best to click on collections of files that belong together (e.g. files from within the same package).

After clicking on the file(s) of interest, it (they) will appear in the lower-left window (and disappear from the upper left window). The selected file(s) is (are) about to be committed. In the lower-right window, type in a message. This should be a meaningful reminder to yourself. For editing a readme file, there is not much to say. But if you are trying out a new algorithm or new parameters, you should describe what you have done. The example follows below, after selecting the readme file and filling in a commit message.

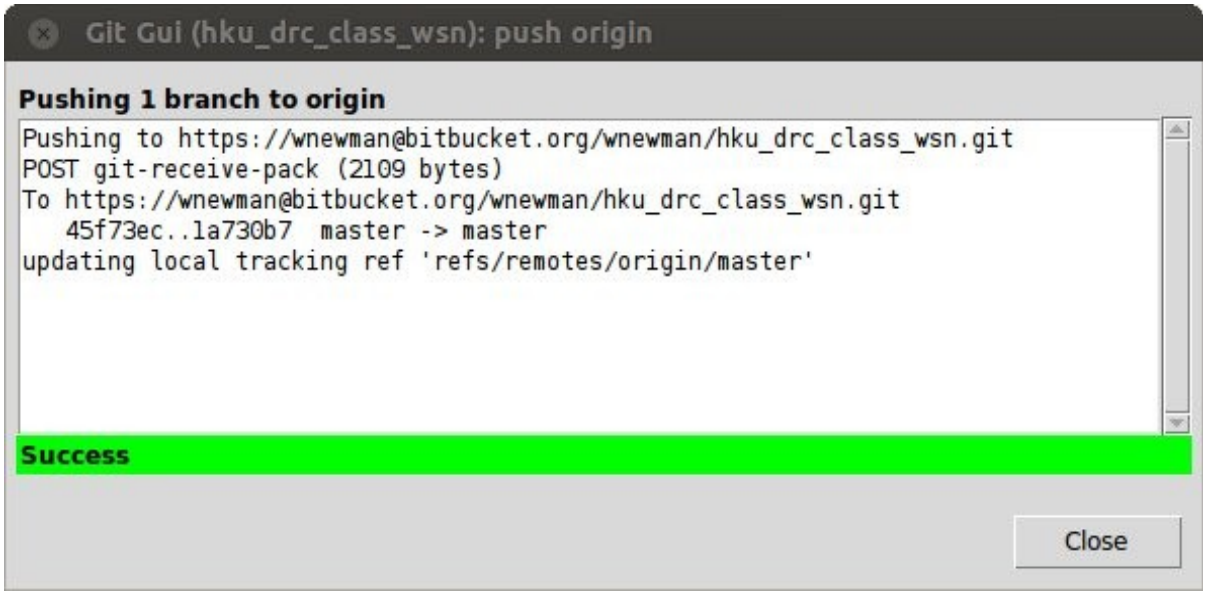


At this point, click the “Commit” button. The lower windows will clear. You can commit more files with more messages, at this point, if desired.

Finally, with all changes committed, you can “push” the changes back to the remote bitbucket repository, bringing the remote site up to date. To do so, click the “Push” button. A “Push” window will appear, as below.



You can accept the defaults (“remote” repository default name is “origin” and you are pushing to the “master” branch). Confirm by again clicking “Push”. You will again be prompted for your bitbucket password, and ultimately you should get a “success” message like the following:



Every time before you log out, you should push your valuable changes back up to the remote repository. Simply assume that if you have not done this, your changes will be lost the next time you come back.

Additional Comments on Git and Revision Control:
The HKU repositories already have a ".gitignore" file in each repository, which specifies types of files (and even directories) that should *not* get pushed to bitbucket. Notably, binary code is never stored on bitbucket. Binary (executable) code tends to be large. Further, it gets out of date quickly, and it can always be re-created by compiling the source code (which *is* stored on bitbucket). At the start of a work session, after merging in the remote repository, you should re-compile your source code. (Or, at least, the packages you need).

There is much more functionality in "git" than has been presented here. Most notably, "git" supports "branches." Branches are useful for spinning off work on separate topics. The branches are generally not expected to be stable or even functional. This is typically code in progress. There is no harm to saving erroneous code in a branch. But once such developments constitute real progress, the branch should be merged into the "master."

Another crucial aspect of bitbucket and git is its support for teams working on a common project. By sharing a repository, your team can all fetch from the same repository and push to the same repository. Importantly, your team should be responsible for making sure the entire repository can be compiled cleanly before pushing new changes. That is, the repository should always be left in a state where any team member can fetch it and compile it cleanly. If you have work in progress that does not compile, you can disable the compilation instruction in your CMakeLists.txt file, so your partners will have no problems compiling.

Another level of protection that is possible is to push code from a forked repository to its parent repository. Your team could have separate forks originating from the same parent. To get new code back into the parent repository, one issues a "pull request." That is, team members cannot force (push) changes into the parent repository, but they can request that the owner of that repository "pull" (fetch from a fork) those changes. In this way, changes to the master repository can be reviewed before acceptance.