


Setting up Eclipse for the CS QuaCS Teaching Lab in H110 (git edition)

These instructions are for Haverford CS courses that use the git system to obtain and submit lab work and the Eclipse development environment; if your course uses CVS (e.g. CMSC 105 or 106 **before Fall 2017**), see [these instructions](#) instead.

- **NOTE:** If you manage to erase your current working files or disconnect them from git, see the [instructions below about reverting/reconnecting](#), but note this will only get back the last things you committed **and pushed**, so commit and push often.
- Log in on the computer
 - a. If in the lab, use the name and password (initially provided by the instructor; use the "kpasswd" command to change it)
 - b. If outside the lab or using your own laptop inside the lab, ssh to the lab as per Option 1 of [this document](#) if possible.
- Start Eclipse - It may be in your dock, or you can start a command-line terminal and type "eclipse". If not in your dock and you want the icon, you can open the applications menu (click the 9 dot icon at the bottom left of the screen) and search for it. The terminal is also an application you can find in this way.
 - a. If, at some point, Eclipse pops up a window about changing the settings for something you're not using and that's not mentioned in these instructions, e.g. "Scala", you should be able to just cancel or close that pop-up window (ask your instructor, to be sure).
- If this is the first time you've done work for a particular course, there are several one-time steps to complete:
 - a. When asked for a workspace, change from the default: select the folder in your home directory that's named after the course (i.e., /home/dwonnaco/cs105 for user dwonnaco in cs105). In future sessions, it should offer to start with this (the most recently-used workspace). You may have to click the **"Launch"** button.
 - Note: If Eclipse goes automatically into a workspace for a previous course, choose "Switch Workspace" from the File menu
 - b. When you first enter a workspace, there should be an orange arrow marked **"Workbench"** at the top right of the window, click it
 - c. Before accessing your first repository or project for the course, do some Eclipse set-up steps
 - Click **preferences** in Eclipse's Window menu
 - Click the **triangle by the word General**
 - Click the **word "Workspace"** under the now-open General category
 - Click the **Refresh using native hooks or polling** checkbox
 - Click the **word "Build"** under the "Workspace" category
 - Click the **Save automatically before build** checkbox (This is important for C++, and maybe Java, but not Python)
 - If you want to change the font, etc., do it now (e.g., Appearance→Colors and Fonts→Basic→Text Font→Edit)
 - Click **Apply and Close**
 - **Quit** Eclipse and re-start (this will ensure your changed settings are saved even if Eclipse crashes)

Congratulations! At this point, you have completed the "one-time-per-course setup" steps. The two steps below will be used to access the repository(s) and project(s) for your course. Note that you may be able to see, or even edit, your files without those steps, but unless you do *both* the ["add a git repository"](#) and the ["import the project"](#) steps below, *for each project*, you may not be able to run or debug your projects in Eclipse, and adding/importing may even become harder if you make changes to the files. So, please do **not** use "File->Open" in Eclipse to start editing without following *both* of the two upcoming steps!

- **Add the git repository** for each project you'll work on (do this and the "import" step below to get the project into eclipse)
 - a. Switch to **git perspective** (e.g., via the "Window → Open Perspective → Other" menu option or using the icon in the Quick Access bar)
 - After this step, the leftmost pane of the Eclipse window should be labelled "Git Repositories"; if this pane is missing, right-click on the word "Git" in the upper right of the Eclipse Window (where it is identifying that you're in the Git perspective), and choose "Reset" and reset the perspective (this will not change your files, just the layout of the Eclipse window).
 - b. Choose **"Add an existing local git repository"** option in the "Git Repositories" pane, *if you are using a QuaCS computer* (after the first time), this will be the first little yellow icon at the top of the Git Repositories pane ().
 - *If you're neither sitting at nor ssh'd to a QuaCS computer, but are instead editing (and perhaps running and debugging) programs on your own computer, see the ["git" section of Option 2](#) for [working on files outside of the QuaCS lab](#).*
 - c. **Set the "directory" box** In the "Add Git Repositories" pop-up window to the *full project name* in your folder, e.g. for the [ComputationalGeometry](#) project, user [dwonnaco](#) would use "browse" to get to (and click OK when found), or just type, [/home/dwonnaco/cs105/ComputationalGeometry](#)
 - d. At this point, there should be a line with an unchecked check-box in the lower pane (if not, perhaps hitting "search" would help?), which when checked will make the **"Add"** box active; Click on **"Add"**. The "add repository" window should close, and the project should now appear in your repository list

At this point, when you're in the "Git" perspective, the "Git Repositories" pane on the left should list your project, but the project won't be listed when you switch the development perspective (e.g., PyDev, C/C++, etc.). If the project is not listed in the "Git Repositories", review the steps above or ask for help. If it is listed there and you want it to be available for you to work on it in, e.g., PyDev perspective, go on to the next step.

- **Import the project** so that you can edit and run it

- a. While still in git perspective, right-click on the **name of the project** you want to import (e.g., ComputationalGeometry), where it is listed in the "Git Repositories" listing on the left of the screen
 - b. Choose **"Import Projects"** from the pop-up menu
 - Click the link "Show other specialized import wizards" which is blue and underlined
 - Click the triangle next to **General**
 - Choose the "Existing Projects into Workspace" option
 - Click **Next>**
 - c. The "root directory" default should be the directory you added above (e.g. /home/dwonnaco/cs105/ComputationalGeometry) and the box in the pane below should be already checked; click **Finish** to import that project
 - d. Switch to the appropriate perspective for the language you'll use, e.g. **PyDev** for Python (via the "Window → Open Perspective → Other" menu option again); if our course uses C++, see [this](#), or [this for HERA](#).
 - Note that for PyDev, it may ask you about configuration the first time. Choose "Advanced Auto-Config" and then select Python 3 from the list of possible interpreters. Click OK in that window and then in a second window.
 - If PyDev later presents a pop-up window titled "Default Eclipse preferences for PyDev", it should be fine to click "OK" to accept the defaults
 - e. Your program should now be listed in the left-hand pane that lists projects/packages, so that you can run it as described in class
- You should then be able to edit your files, run the program, etc., and generally work on the specific lab rather than configuring Eclipse :-)
 - **WHEN YOU WANT TO SUBMIT WORK, DO A COMMIT AND PUSH:**
 - a. Right-click on your project name in the list of projects in the left-hand pane of Eclipse
 - b. Choose the "Commit..." option within the "Team" menu
 - c. **LOOK AT** the list of files in the Git Staging window ... if any files you want to commit are listed under Unstaged Changes, use the green plus to move them to Staged Changes. (If there are distractions from other files listed there that you don't *ever* want to commit, you can cancel the commit and right-click on each file and choose "Remove from index" from the "Team" menu.)
 - d. After checking to be sure all relevant files will be committed, enter a message in the appropriate pane of the commit window, and click the "Commit and Push" button (it is occasionally useful to use "Commit", e.g. to keep a log of changes if "Push" is failing for some reason such as working on your laptop while off-line, but note that only Commit **and Push** will submit your work).
 - e.

NOTE: The rest of the instructions have not been updated to the Photon version of Eclipse in case there are differences

Advanced/Alternative Steps:

- If you want to know whether or not you've really submitted your work for grading:
 - a. In Eclipse, you can check for the signs of un-submitted work:
 - the file names should not have little "?" on the icons next to them; these can be hard to see, but they're important, as they mean files that have not been added to the git index, and thus never committed or pushed, *even if you don't see the signs below*;
 - the project name and file names should not have a ">" next to them; that means uncommitted work
 - the project name should not have an up-arrow (or down-arrow) next to it; these indicate un-pushed changes in your folder (or, for a down-arrow, changes that someone (you?) has pushed to your grading repository, which then need to be pulled and merged; you should ask for help with this).
 - b. On the command-line, you can type "git status". This will show you several things:
 - at the very top, it will tell you if you have unpushed commits, with a sentence like "Your branch is ahead of 'origin/master' by 1 commit (you should push them before the due date, if you want to get credit for them)"
 - it will then list files that have changed since your last commit, in an indented list under the heading "Changes not staged", with the word "modified" in front of each line;
 - finally, any new files that have never been committed will be listed under the heading "Untracked files"
 - Or, if you have committed and pushed all the files, you should see exactly this:


```
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```
- If the professor or T.A. updates the student master files, there is a way to use "pull" to get updates
 - a. Try to do this at a time when you could get help, and when you're not in the middle of other work, since it can create problems.
 - b. cd into your *project* folder for the project that was updated, and use git status to check for unsubmitted work (see the [note above](#))
 - If you have any, commit and push any un-pushed work
 - c. Get the update via either git's command-line or graphical interface like so:
 - to use command-line git, type git pull StarterFiles master
 - to use the graphical interface,, start by typing git gui (this is better than Eclipse for some things, worse for others)
 - Pick "Fetch From -> Starter Files" from the "Remote" menu
 - Pick "Local Merge" from the "Merge" menu
 - If you get a green bar in the result window, you should be all set
 - d. If you get an error message from the command line, or a red bar in the gui "merge" result window, something conflicted with your work, **in which case you should fix the conflicts right away**, since your project may not run and you shouldn't be able to commit it until you fix them.
 - If you have not done fixed merge conflicts before, it may be best to get help; with or without help, here are instructions:
 - If you merged with git gui (above), it should show you a list of conflicting files in the upper-left pane of the window; you

can also do a multi-file search (command-line or in Eclipse's "Search" pop-up window) for lines with "===="; those lines indicate merge problems. If you're using git gui, you can click on the file name and look at the merging that was done.

- Open each file in Eclipse or some other editor, find each occurrence of "====", and fix it as follows:
 - Look for a sequence of lines starting with a ">>>>>" mark, then having the "====" mark, and finally having a "<<<<<" mark at the start of a line. These demarcate two alternative versions of the file: one is yours; the other is the version you got from the merge. Your mission is to replace that entire sequence, including the marker lines, with one version that integrates both versions.
 - The easiest thing to do is usually to pick one version, e.g. between ">>>>>" and "====", and make any changes that are important from the other (between "====" and "<<<<<")
 - Then remove the *other* version (which you looked at but didn't edit) and the three marker lines, producing one good version.
 - After you've fixed *all* the conflicts on each of the conflicted files, click on "Re-scan" and then "Sign-off" in git GUI; if you get an error message, check the named file again, and make sure you *saved* it after editing
 - Click on *blue icon to the left* of the name of each conflicting file, to move it from "unstaged changes" to "staged changes"
 - Once all the files for your project have been staged, click "commit"
 - At the command-line, the following may work instead of the "re-scan"/"sign-off"/"commit" sequence.
 - Double-check to be sure you got rid of all the conflicts, as the command-line approach may not have as many built-in checks
 - Use "git add" for each file that you've de-conflicted (even though it's already in the project; normally you don't have to re-add files)
 - do a "git commit -a" after you've fixed, checked, and re-added the files.
 - e. If Eclipse shows uncommitted changes, it wouldn't hurt to do a "Team->Commit" at this point.
 - f. Before starting any new work, check to make sure the project still works at least as well as it did before the pull (if using C++, it's probably best to do a "Clean Project" and then "Build Project" and then run it). If something(s) have been broken during the merge, it is often best to fix those problems first, and then commit, and then start new work.
- See Dave or a git-fluent student if you want to use git to collaborate or work on your own computer (with something other than ssh); the above instructions are just for using the files from Eclipse *on the QuaCS lab machines*.
 - If you manage to remove **some files** you want, or mess up so badly that you want to go back to an earlier version of your project, you can use Eclipse's "Replace With" menu to do so for a project (or perhaps even for a single file?) in Eclipse, if the project is still connected to the git repository.
 - a. Right-click on the project (or file?) name in the project pane at the left side of the Eclipse window (if you want to revive a file you accidentally deleted, try creating an empty file of the same name and seeing if you can ask to have it replaced).
 - b. Select "Replace With → commit" from the pop-up menu, and look through the *authors* and *comments* of the various commits, to find the one you want (if you can't decide, it may be worth exploring "Compare To → commit" first and then doing the replace).
 - c. For more details, see the discussion in the [advanced use of Eclipse](#) document.
 - If you're not using Eclipse, or manage to remove **your project** entirely or disconnect it from the git repository, you can do the following:
 - a. If you still have a project file, rename it in the file browser or on the command line with e.g. (for a cs245 project named Stack-Scheme) the commands `mv ~/cs245/Stack-Scheme ~/cs245/Stack-Scheme-moved`
 - b. Use the commands
 - `cd ~/cs245`
 - `git clone /home/courses/students/${LOGNAME}/git-cs245/Stack-Scheme`to get a fresh copy of the project (e.g., Stack-Scheme in this case) that's synchronized with the latest *push* you did.
 - c. Use the '[add a git repository](#)' and the '[import the project](#)' steps above, as if you were accessing the project for the first time. This should give you access to the latest version of the files that you had committed *and pushed*.
 - d. If you want any of the program files that you edited since the last push, and still have them in the version you renamed in step a above, you can copy/move the files into the new project (but do **not** do this with the .git folder or its contents).