

Using Git in CCS - A guide for Students

Contents:

Using Git in CCS - A guide for Students	1
0 - Finding Git in CCS:	2
0.1 - 'Git Repositories' view in CCS:	2
1 - Making a repository available in CCS	4
1.1 -Add an existing local repository to the CCS view	4
1.2 - Clone a remote Git repository:	5
1.3 - Creating a new Git repository:	8
2 - Importing projects from a repository	10
3 - Put your files into the repository	12
3.1 - Share a Project into a repository	12
3.2 - Adding Files, Ignoring Files	14
4 - Save your Local Changes	16
4.1 - Committing Changes	16
4.2 - Pushing to Remote repository	20
5 - Load Remote Changes	22
5.1 - Pulling from Remote repository	22
5.2 - Dealing with conflicts	26
6 - Being clever with developing multiple things	31
6.1 - (WIP) Making a branch, switching between branches	31
6.2 - (WIP) Stashing, and restoring a stash	31

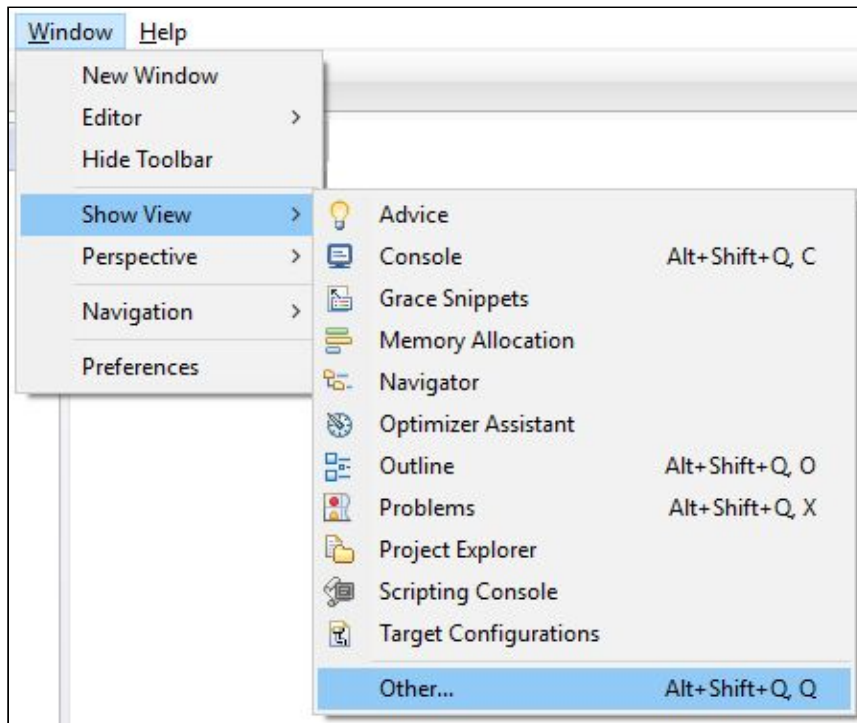
0 - Finding Git in CCS:

Before you can use Git in CCS, you may need to open the “Git Repositories” view, and connect with a local repository, or clone a remote one.

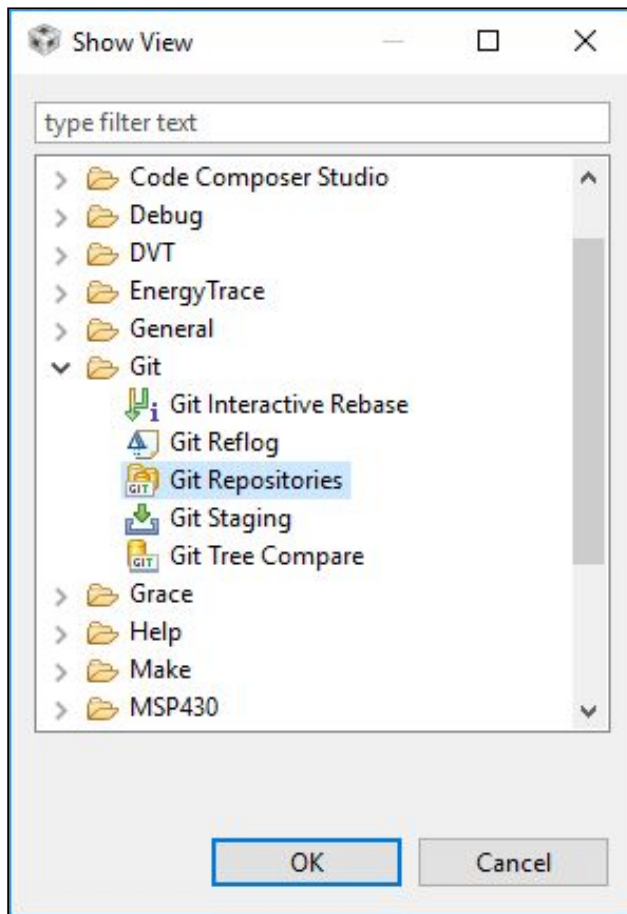
0.1 - ‘Git Repositories’ view in CCS:

Just like the ‘Project Explorer’ view, there is a useful little window view for Git Repositories. Here’s how to open it:

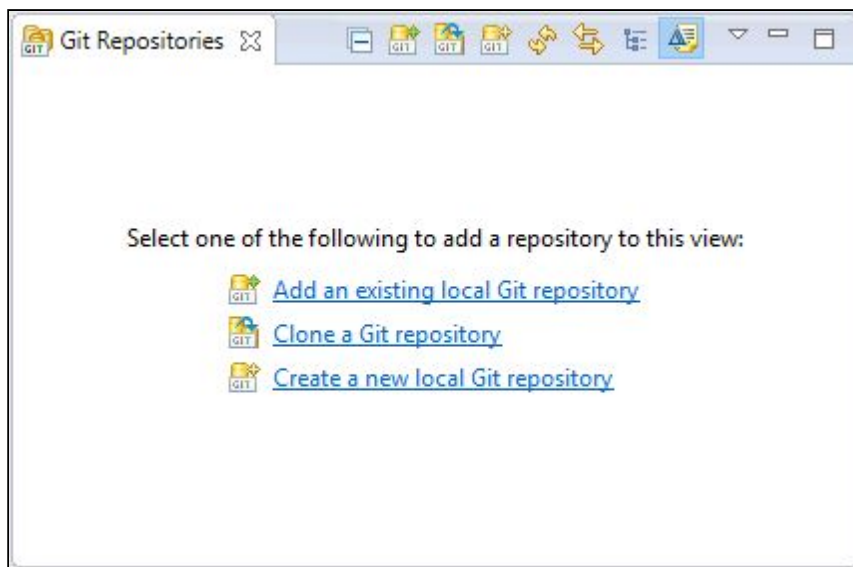
- 1) Go to “**Window > Show View > Other...**”



- 2) Click ">" next to "Git" to expand the tree, then select "**Git Repositories**" and OK.



- 3) Result:

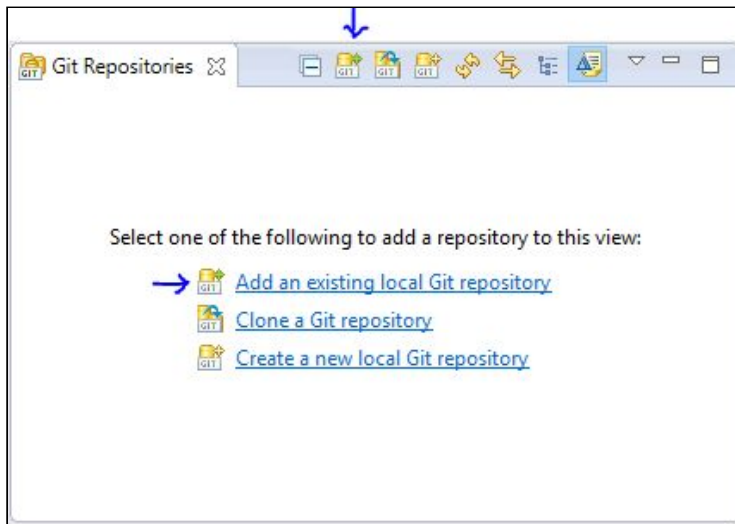


1 - Making a repository available in CCS

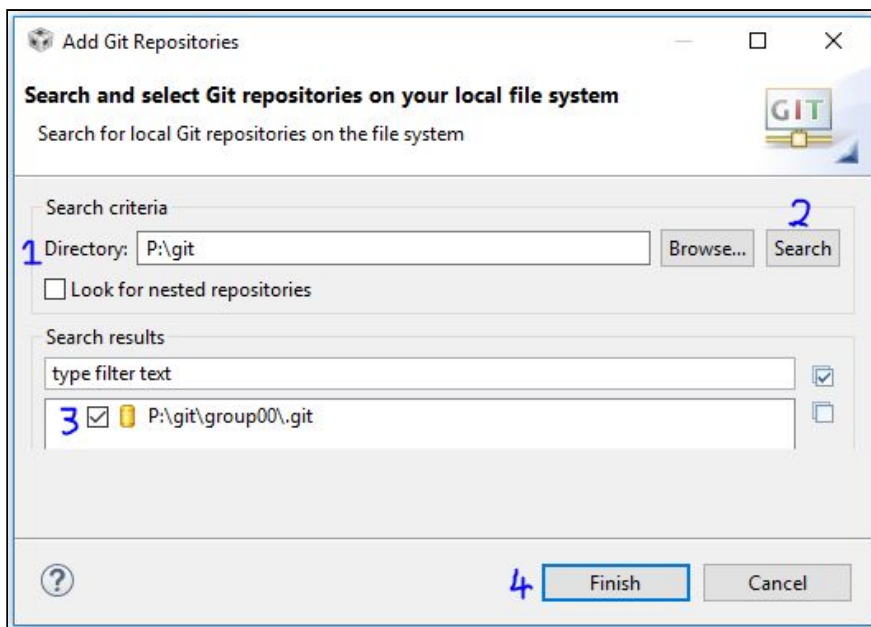
Before you can use Git in CCS, you will need to connect with a local Git repository (1.1), clone a remote one (1.2), or create a new local one (1.3). This section provides a guide to doing this.

1.1 -Add an existing local repository to the CCS view

- 1) Open the Git Repositories view, if you haven't already.
- 2) Click **"Add an existing local Git repository"** (options for this are marked below):



- 3) Navigate to your git directory (1), click search (2), select the repository you want to add (3), and finally click Finish (4).

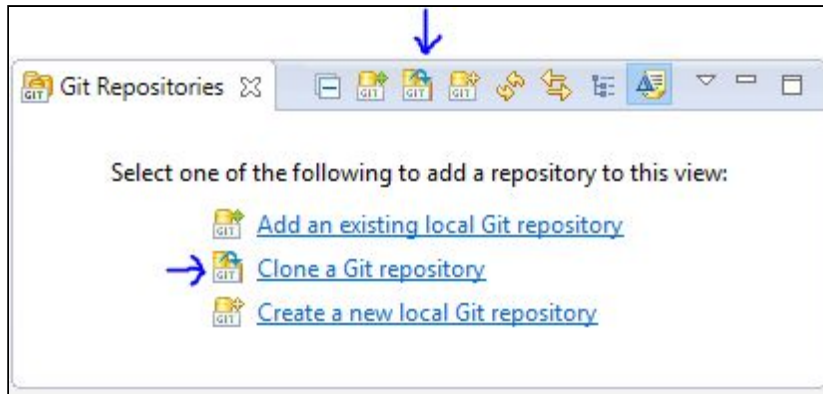


- 4) The repository will have appeared in your Git Repositories view:

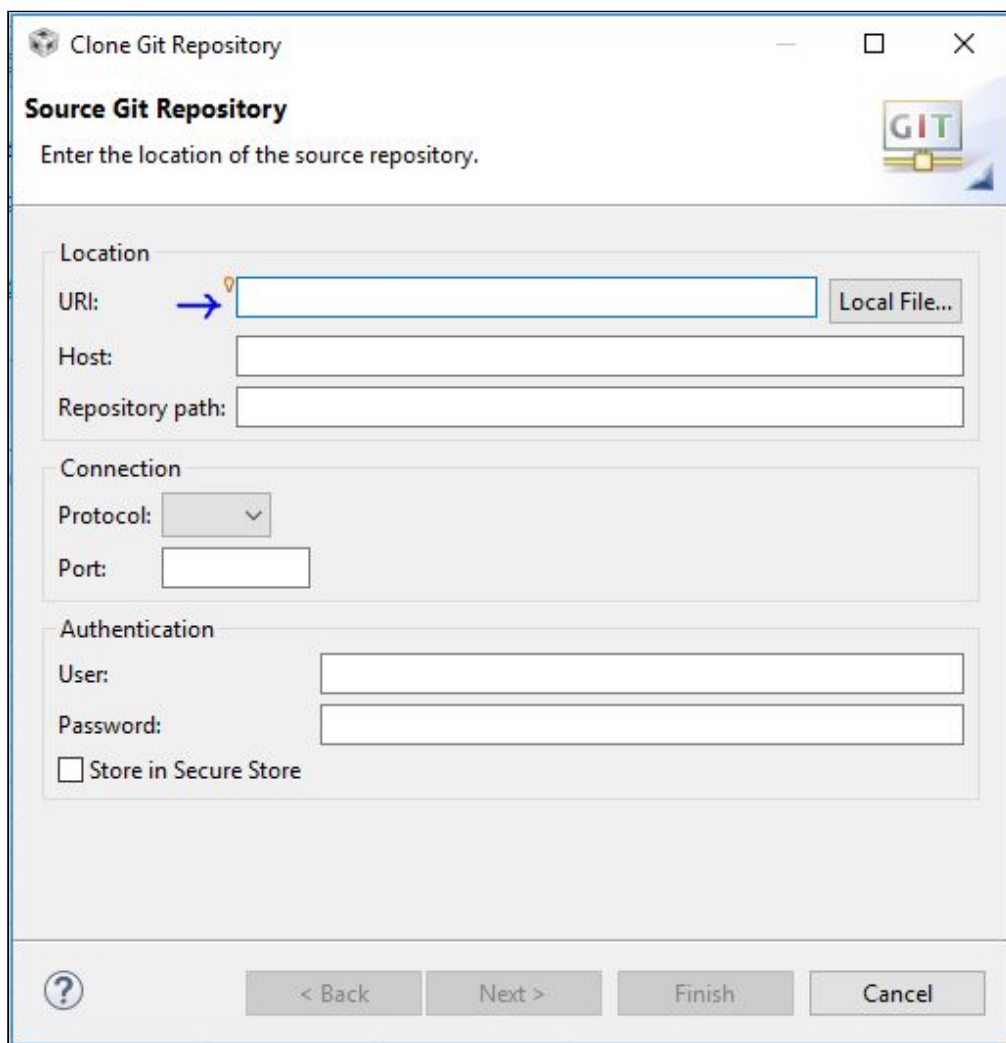


1.2 - Clone a remote Git repository:

- 1) Open the Git Repositories view, if you haven't already.
- 2) Click "**Clone a Git repository**" (options for this are marked with arrows below):



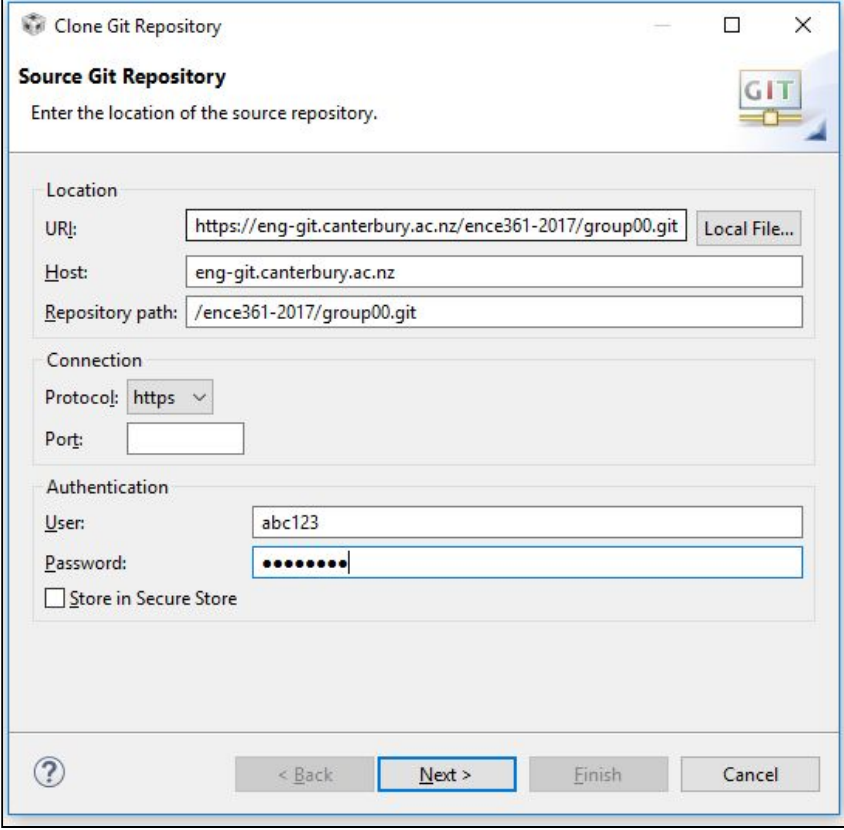
- 3) In the "Location" section, enter the address of the git repository you want to clone in the "URI" field. E.g.: <https://eng-git.canterbury.ac.nz/ence361-2017/group00.git>
The "Host", "Repository path", and "Protocol" fields will fill automatically, based on the address.



The 'Clone Git Repository' dialog box is shown. It has a title bar with a question mark icon and standard window controls. The main section is titled 'Source Git Repository' and contains the instruction 'Enter the location of the source repository.' Below this, there are three sections: 'Location', 'Connection', and 'Authentication'. The 'Location' section has a 'URI:' label with a blue arrow pointing to the text input field, which contains the URL 'https://eng-git.canterbury.ac.nz/ence361-2017/group00.git'. To the right of the URI field is a 'Local File...' button. Below the URI field are 'Host:' and 'Repository path:' labels, each with a text input field. The 'Host' field contains 'eng-git.canterbury.ac.nz' and the 'Repository path' field contains '/ence361-2017/group00.git'. The 'Connection' section has a 'Protocol:' dropdown menu set to 'https' and a 'Port:' text input field. The 'Authentication' section has 'User:' and 'Password:' text input fields, and a checkbox labeled 'Store in Secure Store' which is currently unchecked. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

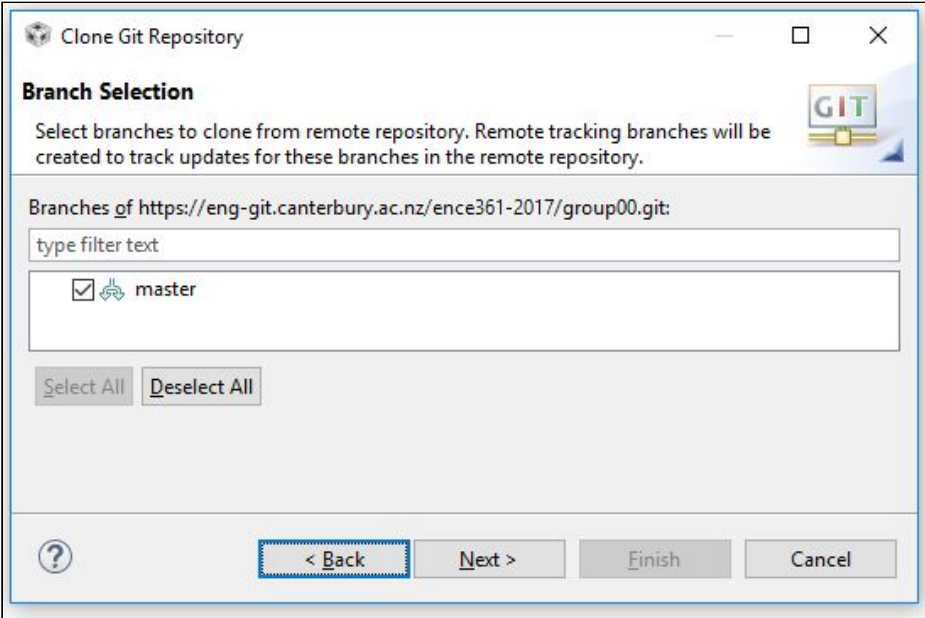
- 4) In the Authentication section, enter the User and Password associated with the host of the repository you are cloning (e.g., eng-git.canterbury.ac.nz). Click “Next >”.

Note: The User account must have sufficient privileges in the git project to ‘pull’ a copy of the repository, or this cloning process will fail. In “eng-git” there are roles with different privileges and “Reporter” is the minimum level required to pull. Developer is the minimum level required to push.



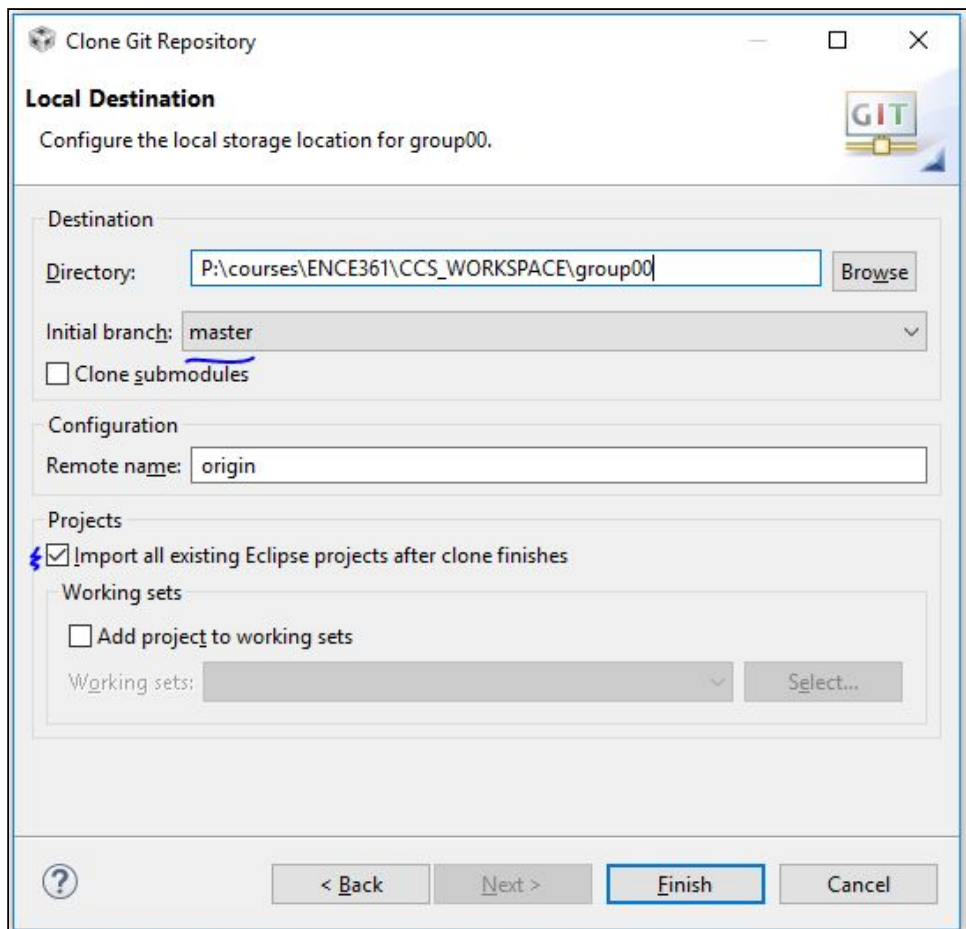
The screenshot shows the 'Clone Git Repository' dialog box with the 'Source Git Repository' tab selected. The dialog has a title bar with a question mark icon, a close button, and a maximize button. The main content area is titled 'Source Git Repository' and contains the instruction 'Enter the location of the source repository.' Below this, there are three sections: 'Location', 'Connection', and 'Authentication'. The 'Location' section has three text boxes: 'URI:' with the value 'https://eng-git.canterbury.ac.nz/ence361-2017/group00.git', 'Host:' with 'eng-git.canterbury.ac.nz', and 'Repository path:' with '/ence361-2017/group00.git'. There is a 'Local File...' button next to the URI box. The 'Connection' section has a 'Protocol:' dropdown menu set to 'https' and an empty 'Port:' text box. The 'Authentication' section has a 'User:' text box with 'abc123', a 'Password:' text box with masked characters, and a checkbox for 'Store in Secure Store'. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

- 5) Branch Selection: If the repository has more than one branch, you can select which branches you wish to pull. It is usually fine to just pull them all, for small projects. Click “Next >”

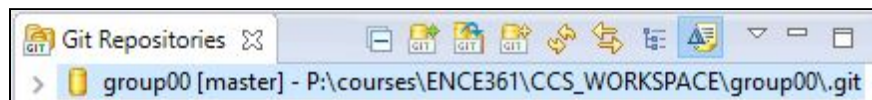


The screenshot shows the 'Clone Git Repository' dialog box with the 'Branch Selection' tab selected. The dialog has a title bar with a question mark icon, a close button, and a maximize button. The main content area is titled 'Branch Selection' and contains the instruction 'Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.' Below this, there is a text box showing 'Branches of https://eng-git.canterbury.ac.nz/ence361-2017/group00.git:' and a 'type filter text' input field. A list box below shows a single entry 'master' with a checked checkbox and a branch icon. At the bottom of the list box are two buttons: 'Select All' and 'Deselect All'. At the very bottom of the dialog, there are four buttons: '?', '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

- 6) Choose where you want to clone the repository to. If it is a CCS project repository, you can clone it directly into your workspace (See 'Making a new CCS workspace', if you want to try it in a safe way). If it is just source files, you could clone the repository to a generic git directory, such as the default "git\Project".



- a) Choose the "Initial branch:" to be the branch most convenient for you.
- b) Tick 'Import all existing Eclipse projects after clone finishes' if you know the repository contains projects you want to import and use. This is likely, as you are following this guide.
- c) Click "Finish" when you are happy with the local destination settings and the cloning will be carried out.
- 7) Verify that you have your project and repository.
- a) The repository should be present in your "Git Repositories" view:



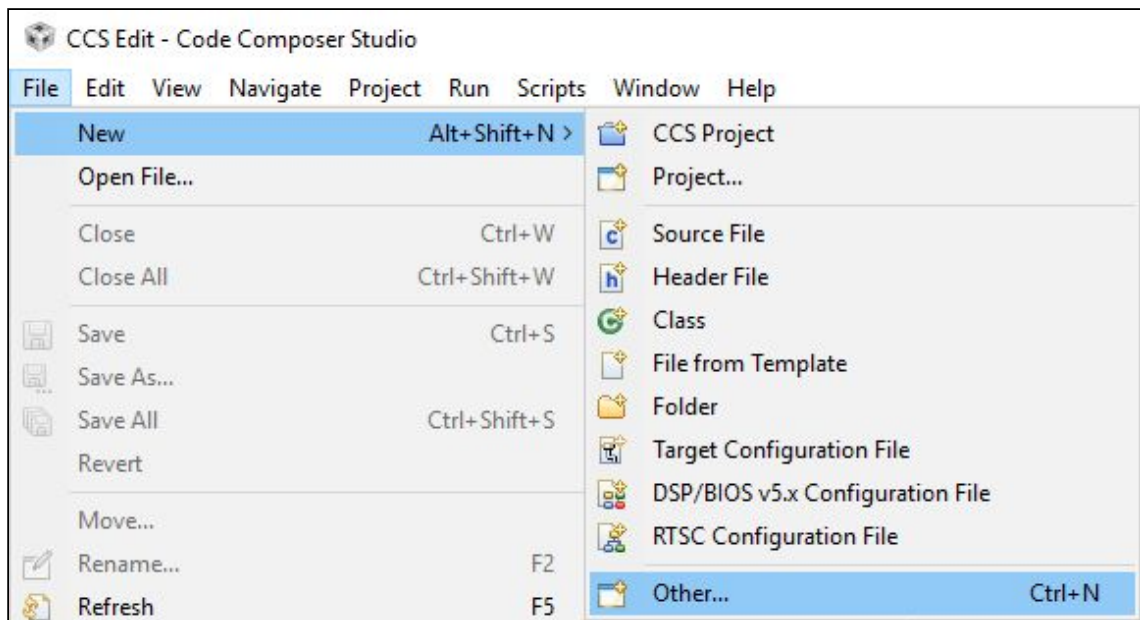
- b) The project should be present in your "Project Explorer" view:



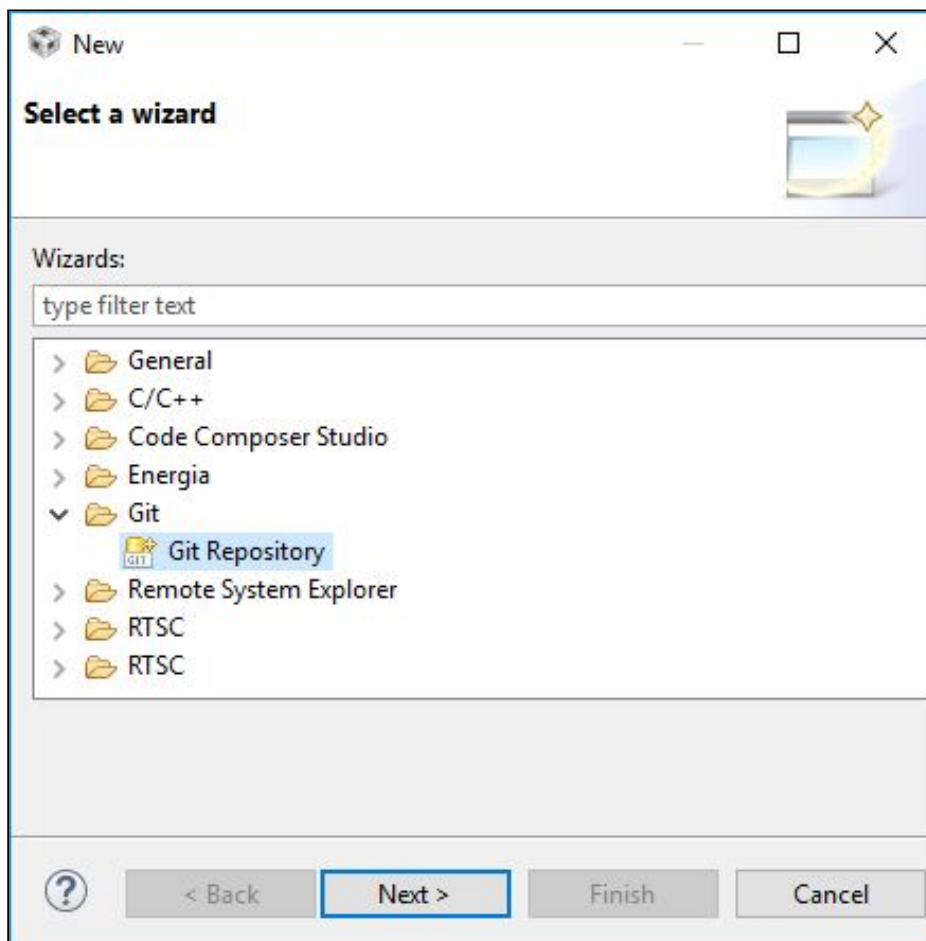
That's it for cloning a git repository in CCS.

1.3 - Creating a new Git repository:

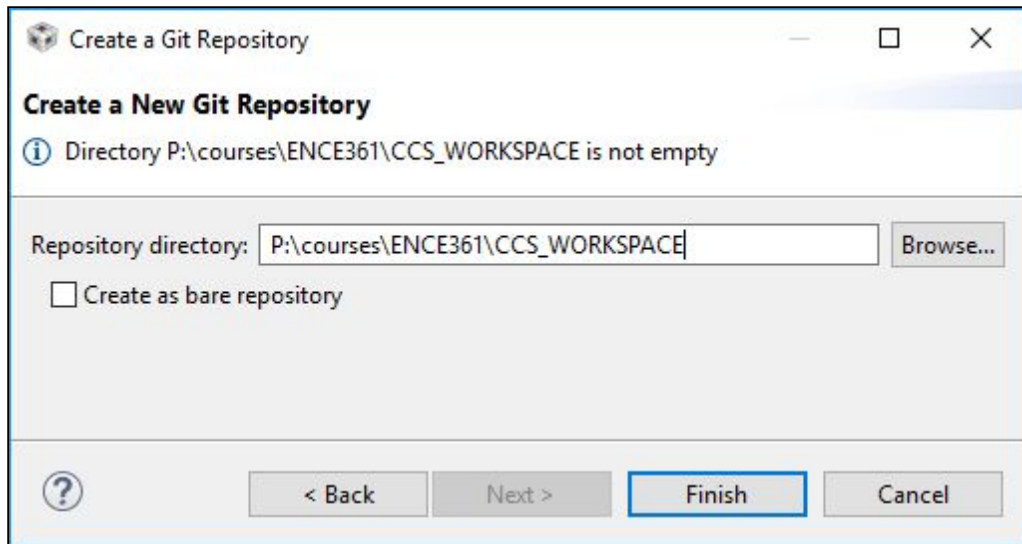
- 1) Go to “**File > New > Other...**”:



- 2) Click “>” next to “*Git*” to expand the Git tree, then select “**Git Repository**” and click “Next >”



- 3) Browse to, or type in the path to, the directory where you would like to create your new Git repository. Some people find it easier to keep track of things if you put it in your workspace folder. Click “Finish” to confirm.

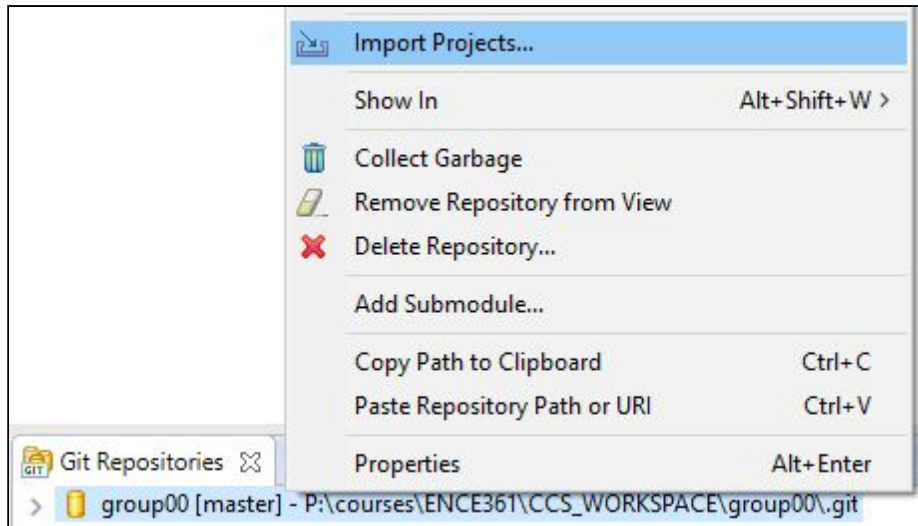


The screenshot shows a Windows-style dialog box titled "Create a Git Repository". Inside, the title "Create a New Git Repository" is displayed. Below it, an information icon (i) is followed by the text "Directory P:\courses\ENCE361\CCS_WORKSPACE is not empty". A text field labeled "Repository directory:" contains the path "P:\courses\ENCE361\CCS_WORKSPACE", with a "Browse..." button to its right. Below the text field is a checkbox labeled "Create as bare repository" which is currently unchecked. At the bottom of the dialog, there is a question mark icon on the left, and four buttons: "< Back", "Next >", "Finish" (which is highlighted with a blue border), and "Cancel".

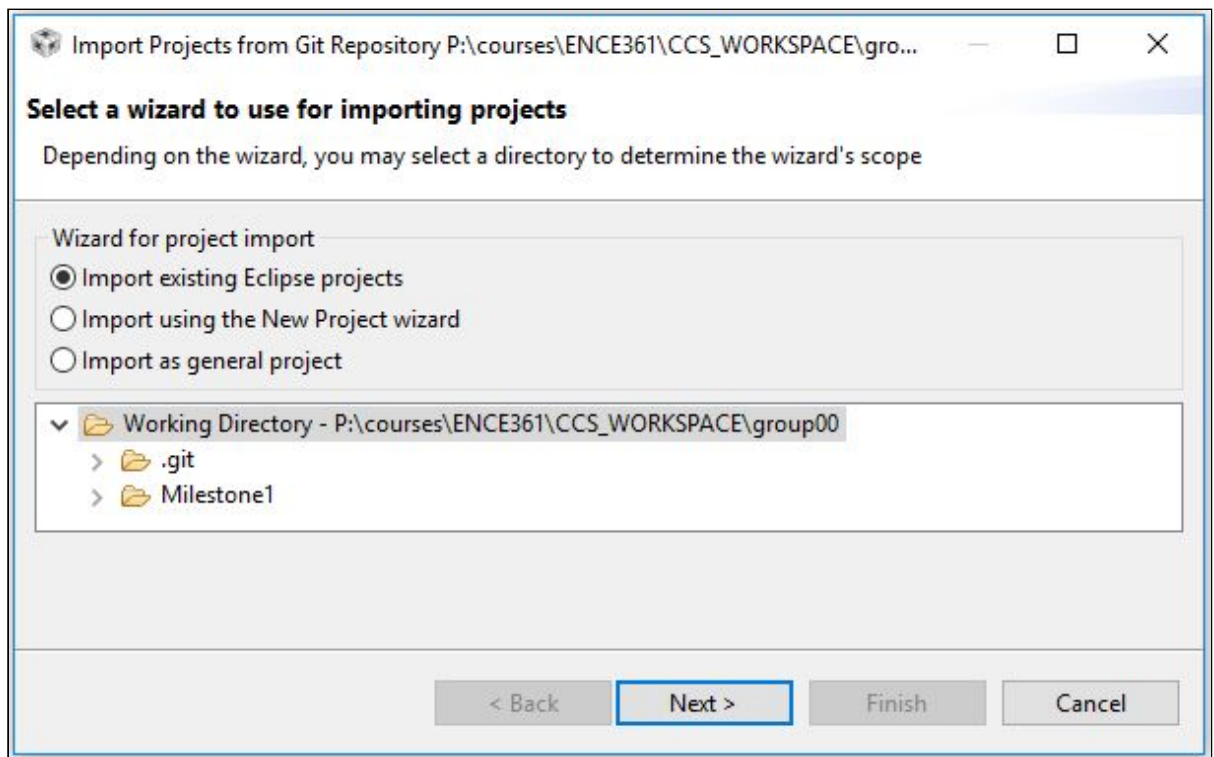
2 - Importing projects from a repository

If you have an existing local Git repository with a CCS project

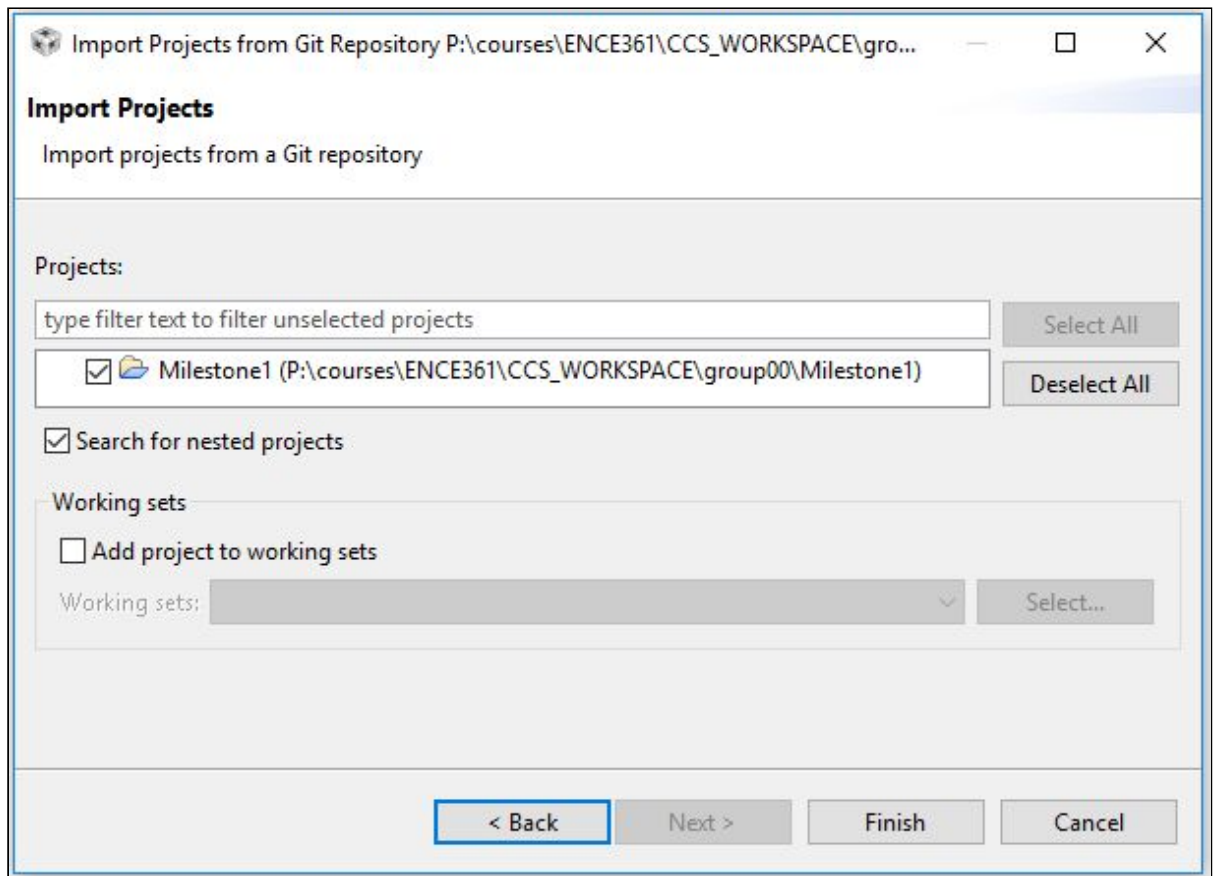
- 1) Right click on the repository, click **"Import Projects..."**



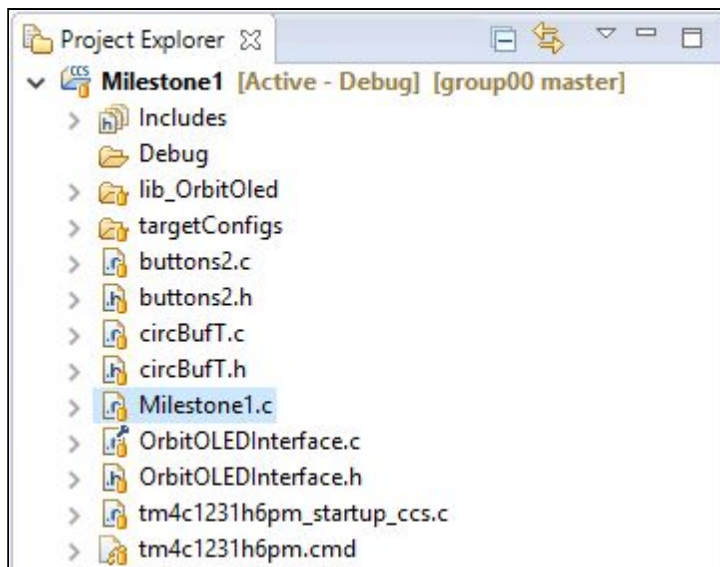
- 2) Hit **"Next>"**. Importing existing Eclipse projects is fine (assuming your repository has .project files along with the source files) CCS should recognise the projects.



- 3) Select the project(s) you want to import from the repository, then click “Finish”.



- 4) CCS will import the project, and it will be connected to the Git repository. The result is shown below. The little orange labels on the file icons indicate that the file/folder is tracked in the Git repository.



3 - Put your files into the repository

3.1 - Share a Project into a repository

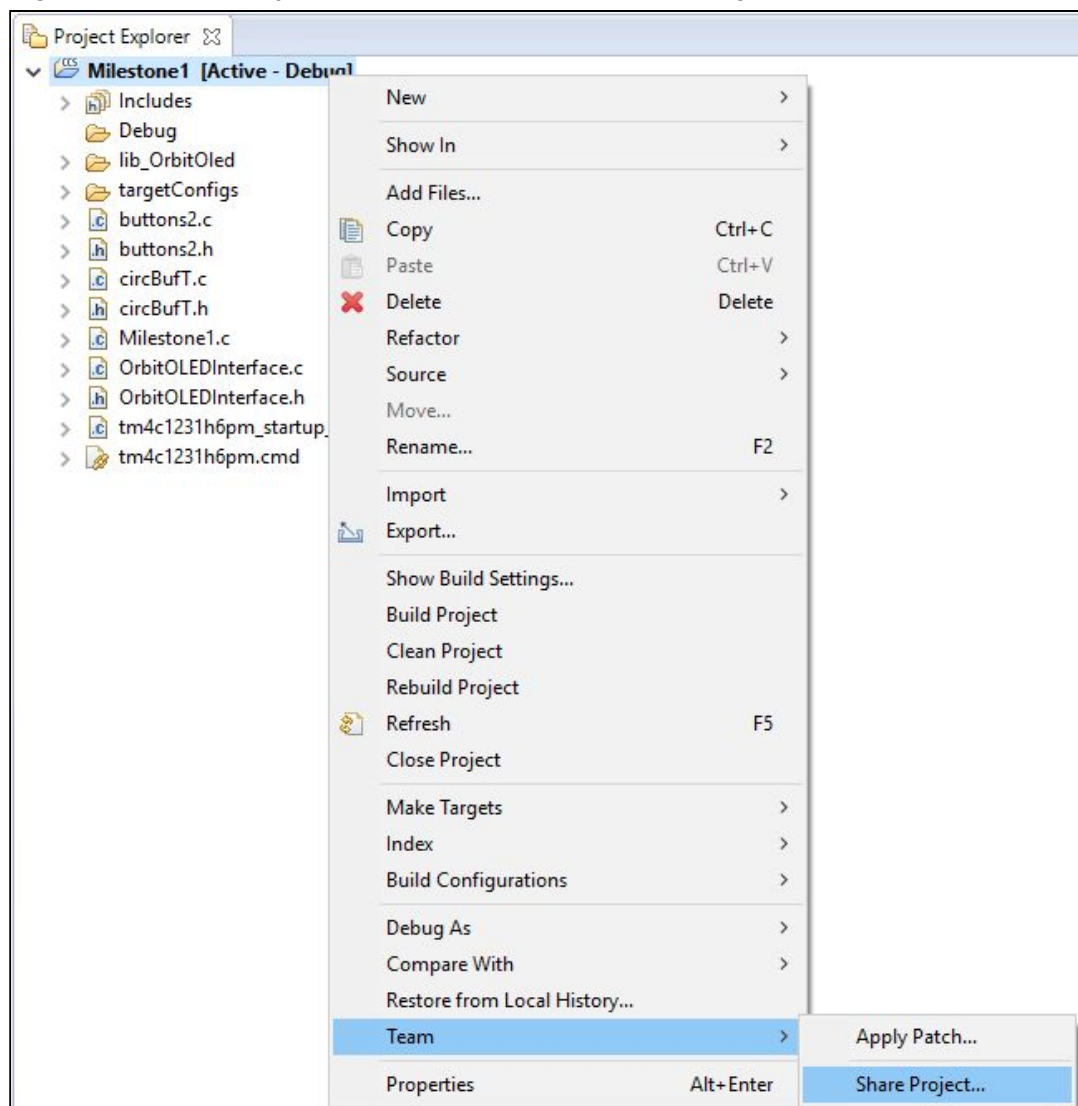
If you have a git repository with no CCS project in it, you can use CCS to move an existing project from your workspace, into the git repository.

If you have a project that you would like to be included in a Git repository, CCS can do that.

Assuming:

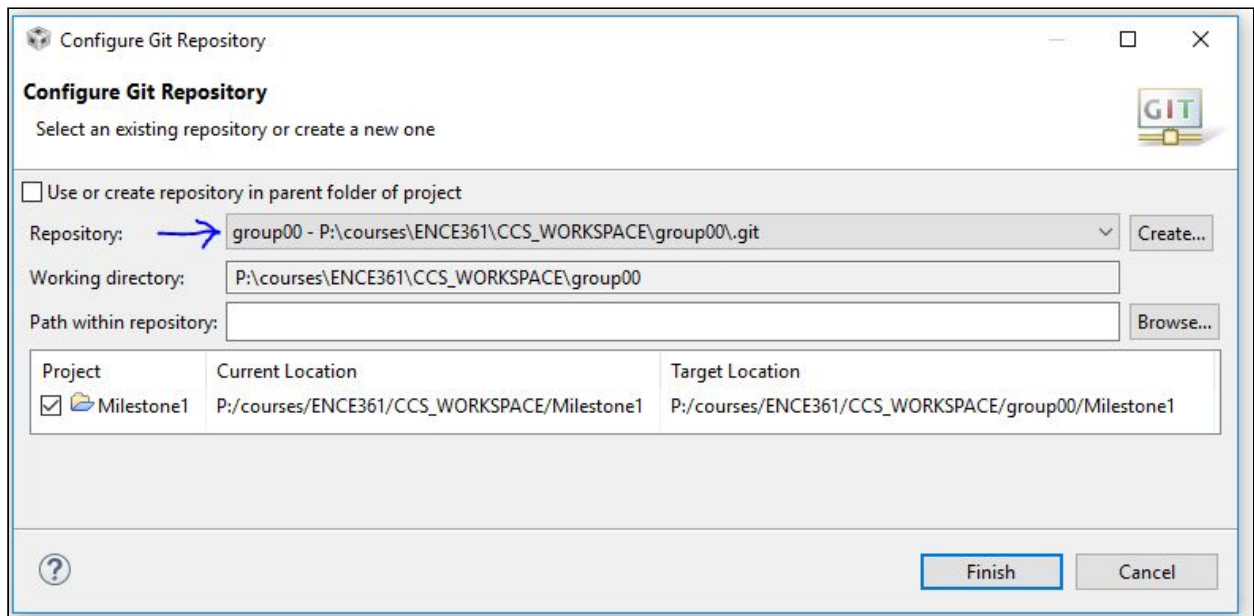
- You have a project.
- You have already connected CCS to a repository by creating a new repository, cloning a repository, or by adding an existing local repository (Add a repository to the CCS Git repositories view).

- 1) Right click on the project, and select **“Team > Share Project ...”**



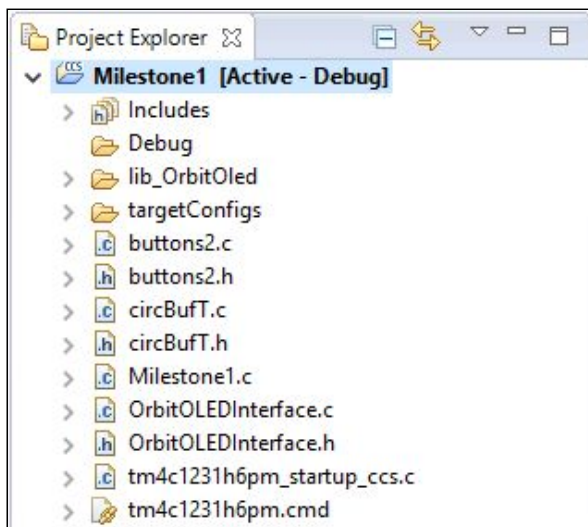
- 2) Click in the “Repository” field and select the repository that you have added to the repository view in CCS. If all the directories and locations are correct, then hit “Finish”

and CCS will move the project into the Git repository directory.

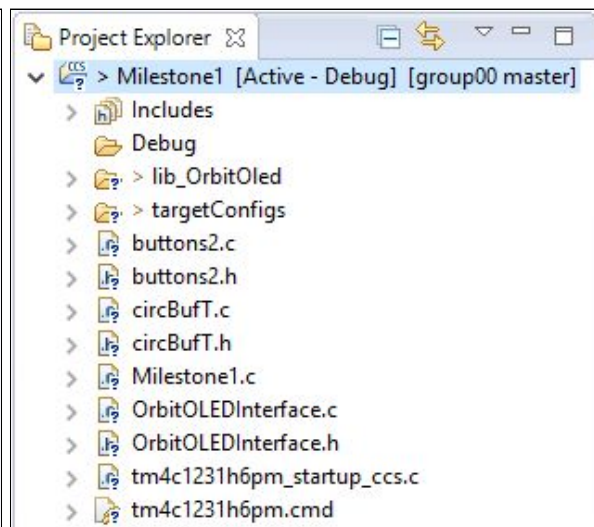


Once this is done, you can start performing Git related tasks, and Git related icons will appear in your “Project Explorer” view.

Before:



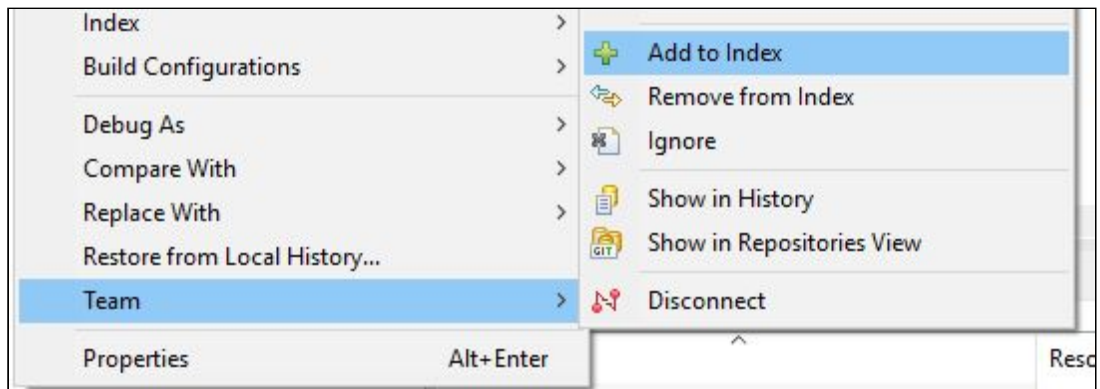
After:



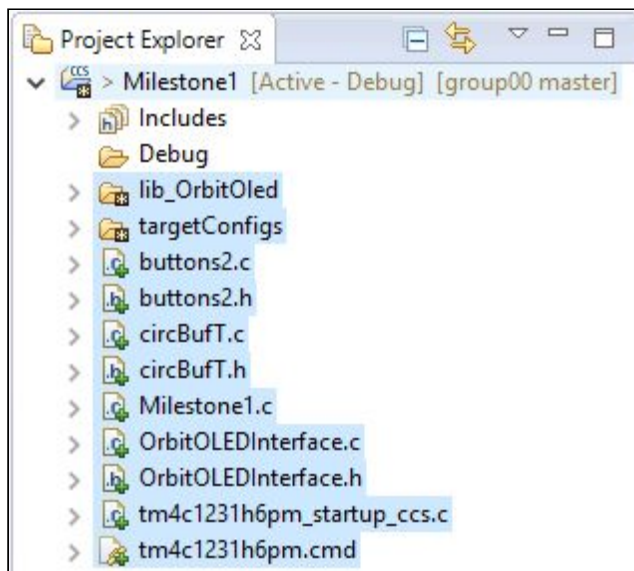
3.2 - Adding Files, Ignoring Files

Once you created a new repository and want to add your files to it, or you have created new files in project in an existing repository, you'll need to tell CCS that you want them tracked by Git. In general for your CCS projects, you'll want to track all the source files, the 'targetConfigs' folder (or the target configuration xml file), the startup_ccs.c and .cmd files.

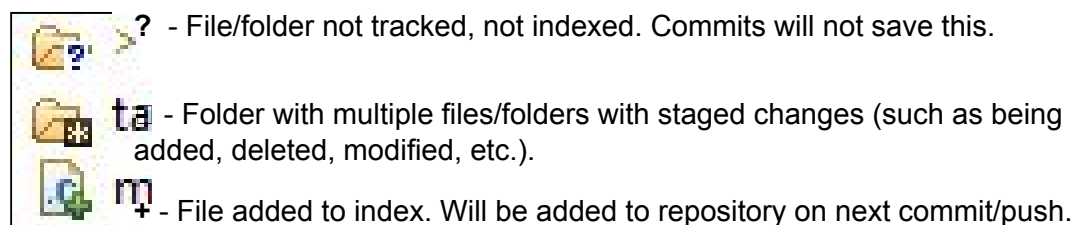
- 1) **Adding Files:** - Right click on the file(s) and/or folder(s) that you want to track, in the "Project Explorer" view. Go to "**Team > Add to Index**".



Result:



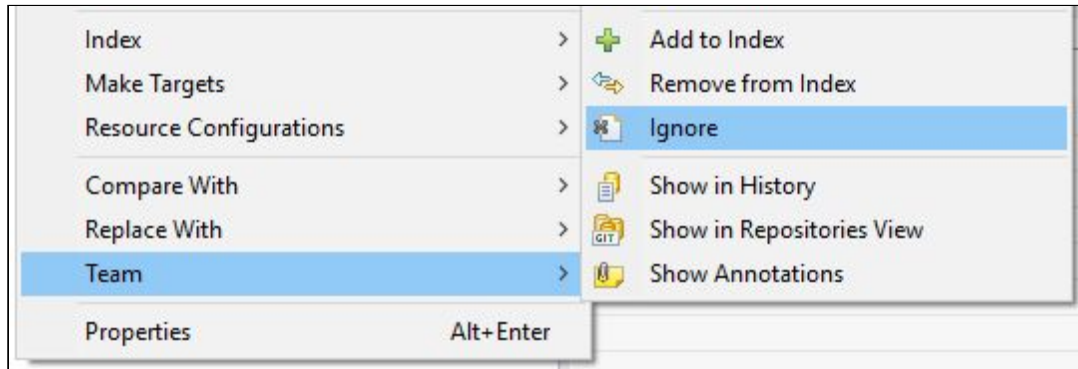
- 2) Meaning of Icons:



Don't bother adding the "Debug" folder. It is just full of generated files, you shouldn't ever need to share these with other team members (unless you need the .out or .bin program

file). If you want the project to be portable, you will also need to add the “.project”, “.cproject”, and “.ccsproject” files. These are usually hidden, but will show up when you try to make a commit. You can also access these by opening the ‘Working Directory’ in the Git Repositories view in CCS.

- 3) **Ignoring Files:** - Right-click on a file or folder you never want to track in the Git repository (such as the Debug folder in your project):

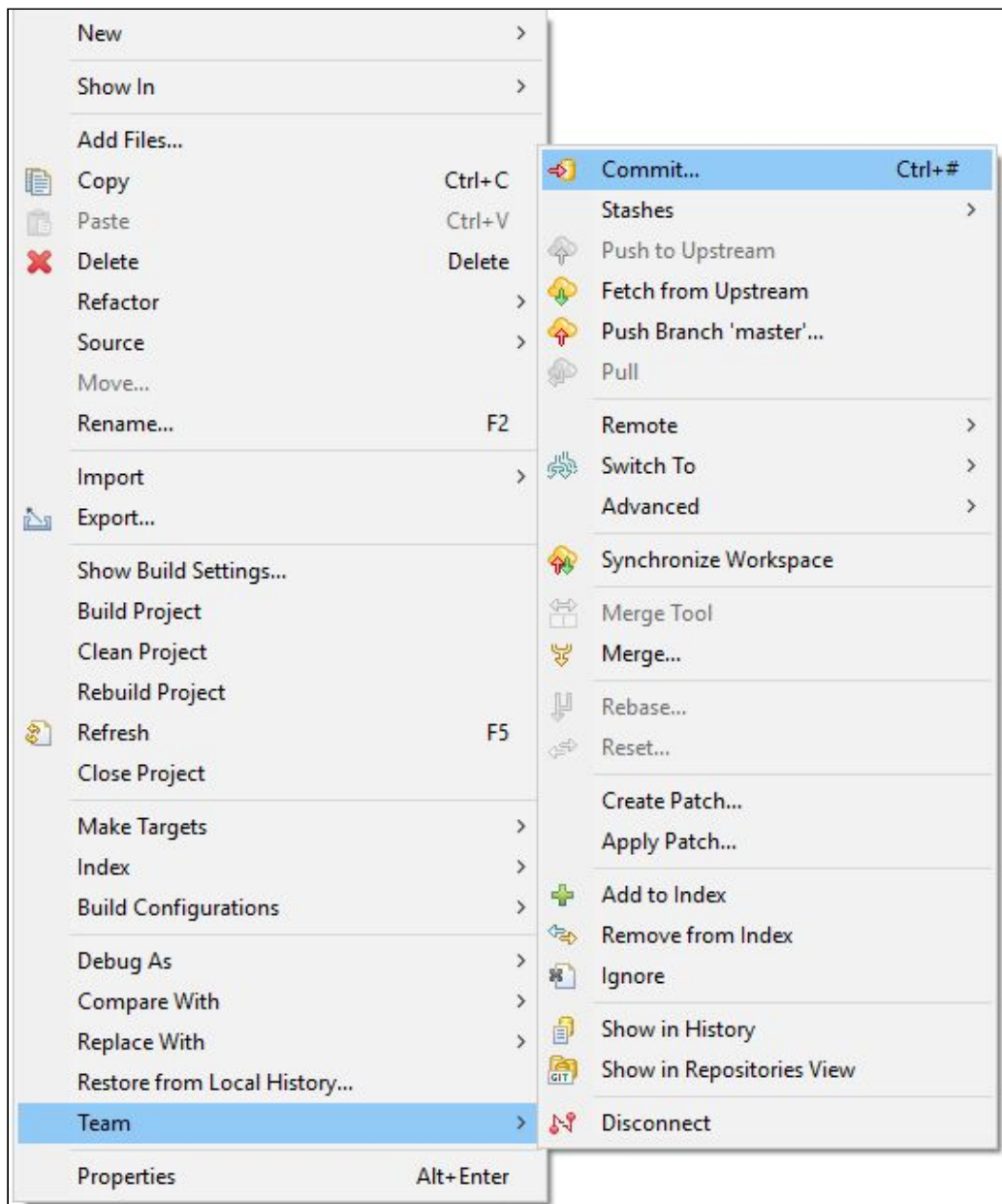


4 - Save your Local Changes

4.1 - Committing Changes

So you have your CCS project tracked in a Git repository, and you have changes that you want to commit (save).

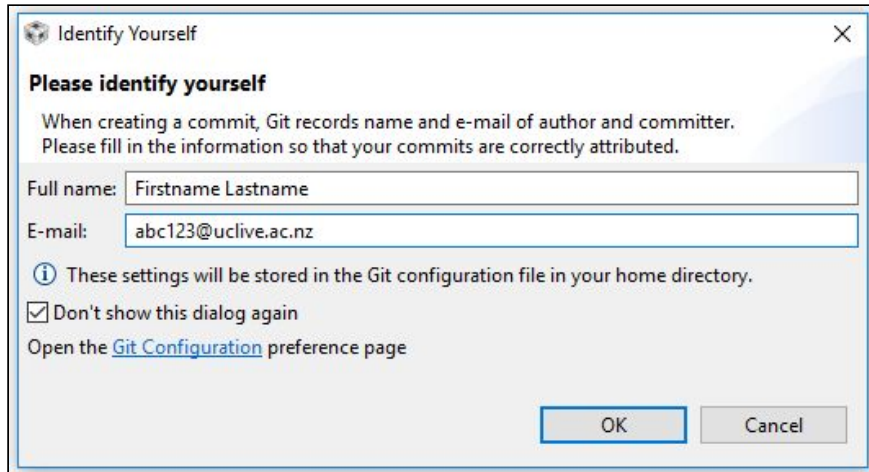
- 1) Right click on the project, go to **"Team > Commit..."**. If this is your first time using the repository, CCS will ask you to identify yourself, so that work done by you will be correctly linked to you (this helps you take credit).



- 2) Fill in an appropriate identity and click OK. It should be similar details as you have for you eng-git account, if that is where your repository is stored.

Note: These details will be stored in the `".gitconfig"` file in the root of your P drive (or whatever drive that you have the workspace in).

If you need to find the “Git Configuration” preference page later, it is in “Window > Preferences -> Team -> Git -> Configuration”.



The image shows a standard Windows-style dialog box titled "Identify Yourself". It has a close button (X) in the top right corner. The main text reads "Please identify yourself" followed by an explanatory paragraph: "When creating a commit, Git records name and e-mail of author and committer. Please fill in the information so that your commits are correctly attributed." Below this, there are two text input fields. The first is labeled "Full name:" and contains the placeholder text "Firstname Lastname". The second is labeled "E-mail:" and contains the text "abc123@uclive.ac.nz". Below the input fields, there is an information icon (i) followed by the text "These settings will be stored in the Git configuration file in your home directory." Below that is a checked checkbox with the label "Don't show this dialog again". At the bottom left, there is a link that says "Open the [Git Configuration](#) preference page". At the bottom right, there are two buttons: "OK" and "Cancel".

Identify Yourself

Please identify yourself

When creating a commit, Git records name and e-mail of author and committer.
Please fill in the information so that your commits are correctly attributed.

Full name: Firstname Lastname

E-mail: abc123@uclive.ac.nz

i These settings will be stored in the Git configuration file in your home directory.

☒ Don't show this dialog again

Open the [Git Configuration](#) preference page

OK Cancel

- 3) Type in an appropriate commit message to say something useful about the reason for this commit. Also, consider selecting the `“.ccsproject”`, `“.cproject”`, and `“.project”` files in the bottom view of the window. These files are not essential source files, but including them allows others to use CCS to import the project with all its settings intact, as opposed to adding the files to a fresh project.

Commit Changes

Commit Changes to Git Repository

Commit message

Added project to repository
First!

Author:

Firstname Lastname <abc123@uclive.ac.nz>

Committer:

Firstname Lastname <abc123@uclive.ac.nz>

Files (27/30)

type filter text

Status	Path
<input checked="" type="checkbox"/>	Milestone1/Milestone1.c
<input checked="" type="checkbox"/>	Milestone1/OrbitOLEDInterface.c
<input checked="" type="checkbox"/>	Milestone1/OrbitOLEDInterface.h
<input checked="" type="checkbox"/>	Milestone1/buttons2.c
<input checked="" type="checkbox"/>	Milestone1/buttons2.h
<input checked="" type="checkbox"/>	Milestone1/circBufT.c
<input checked="" type="checkbox"/>	Milestone1/circBufT.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/ChrFont0.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/FillPat.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/FillPat.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/LaunchPad.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitBoosterPackDefs.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOled.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOled.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOledChar.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOledChar.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOledGrph.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/OrbitOledGrph.h
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/delay.c
<input checked="" type="checkbox"/>	Milestone1/lib_OrbitOled/delay.h
<input checked="" type="checkbox"/>	Milestone1/targetConfigs/Tiva TM4C1231H6PM.ccxml
<input checked="" type="checkbox"/>	Milestone1/targetConfigs/readme.txt
<input checked="" type="checkbox"/>	Milestone1/tm4c1231h6pm.cmd
<input checked="" type="checkbox"/>	Milestone1/tm4c1231h6pm_startup_ccs.c
<input checked="" type="checkbox"/>	Milestone1/.ccsproject
<input checked="" type="checkbox"/>	Milestone1/.cproject
<input type="checkbox"/>	Milestone1/.gitignore
<input checked="" type="checkbox"/>	Milestone1/.project
<input type="checkbox"/>	Milestone1/.settings/org.eclipse.cdt.codan.core.prefs
<input type="checkbox"/>	Milestone1/.settings/org.eclipse.cdt.debug.core.prefs

Open [Git Staging view](#)

Commit and Push

Commit

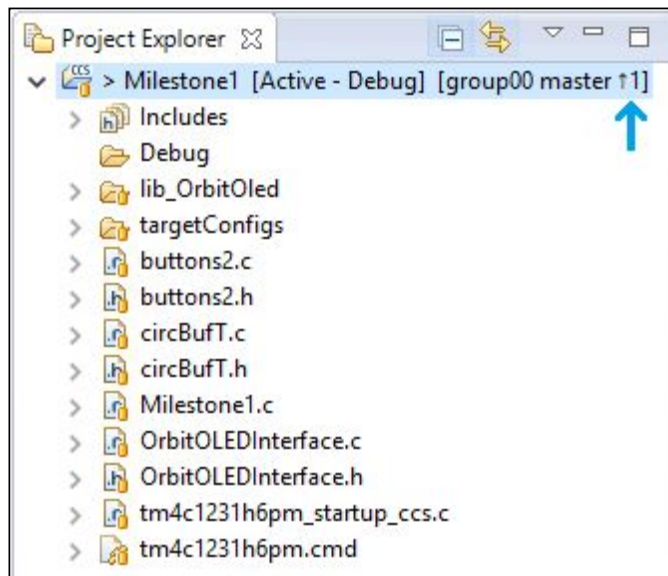
Cancel

- 4) Click Commit to store the changes locally, or “Commit and Push” to store the changes locally and also send them to the online repository (pushing is necessary for your teammates to download your changes).

Result:

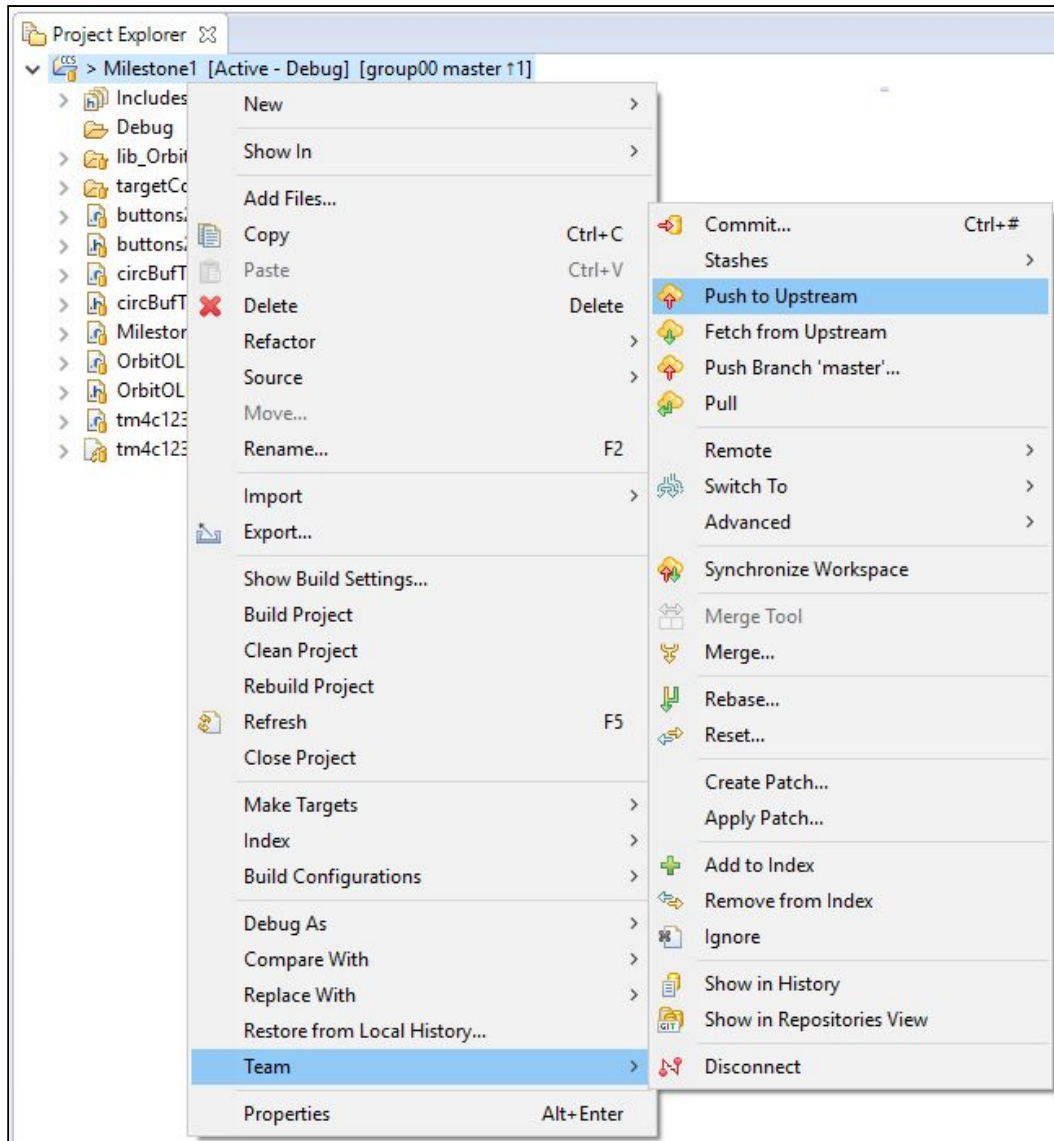
Note the little up arrow with the number next to it. This indicates your local repository is “1 commit ahead of the remote repository”. Sometimes it might have a down arrow, indicating you are using a commit that is not the most recent.

Also note the the icons now have little orange cylinder labels (🔧). They indicate that the item is being tracked, and has no changes staged.

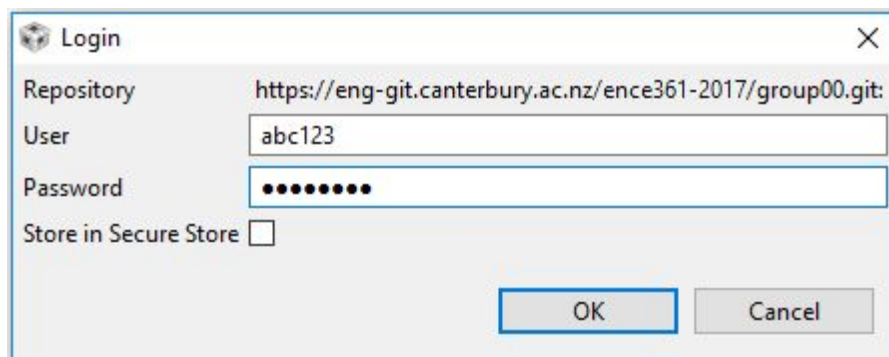


4.2 - Pushing to Remote repository

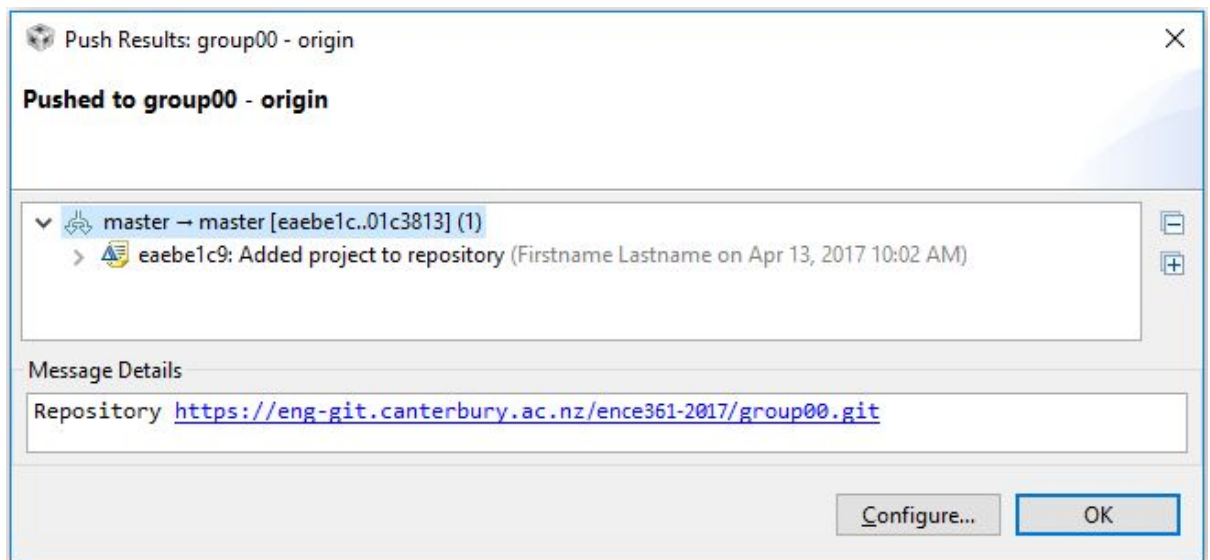
- 1) Assuming you commit changes you'd like to commit, right click on the project, and go to **"Team > Push to Upstream"**.



- 2) Enter your credentials (you need permission to push to the repository - Developer role or better)



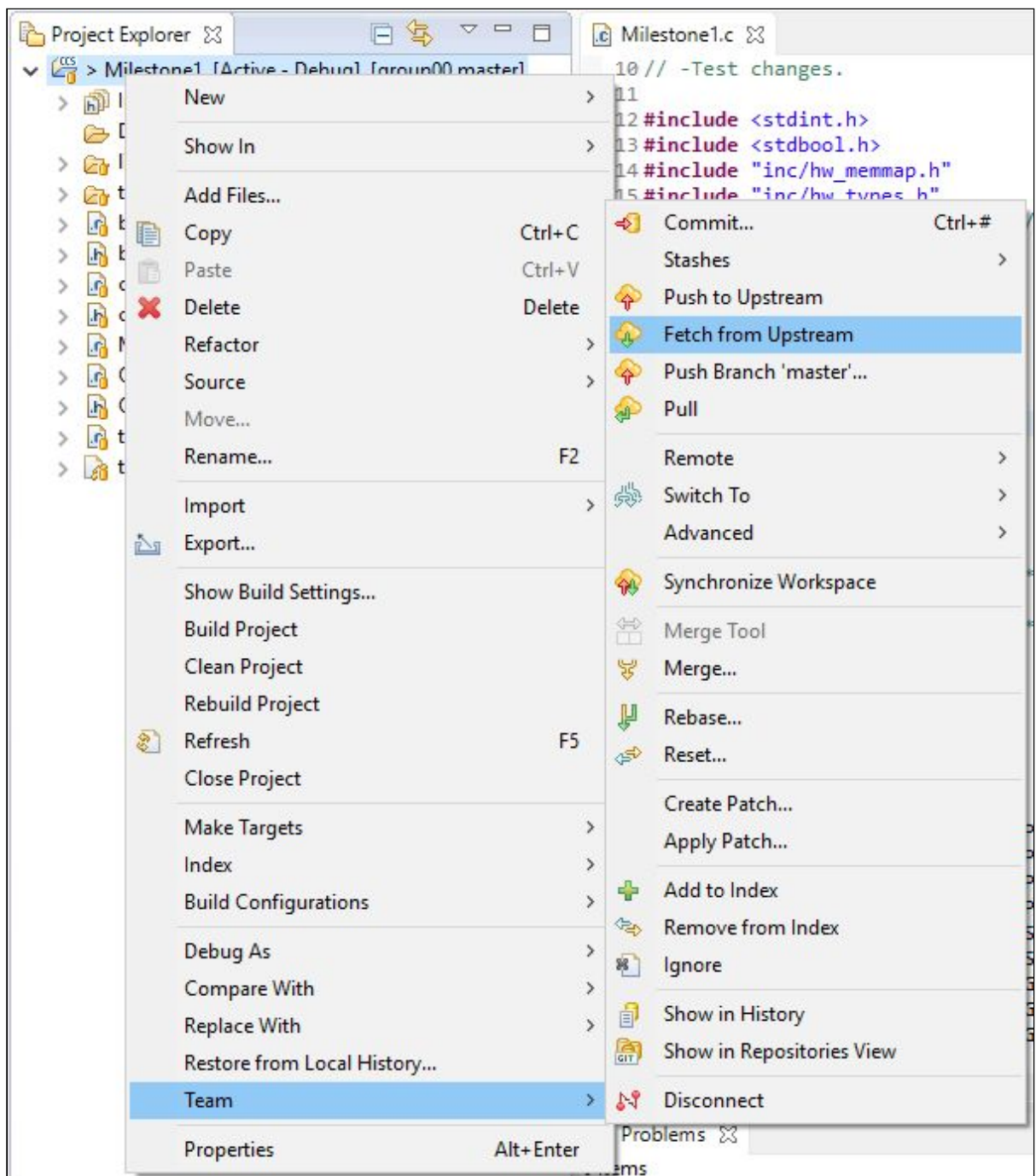
3) A summary will be shown once the pushing is complete.



5 - Load Remote Changes

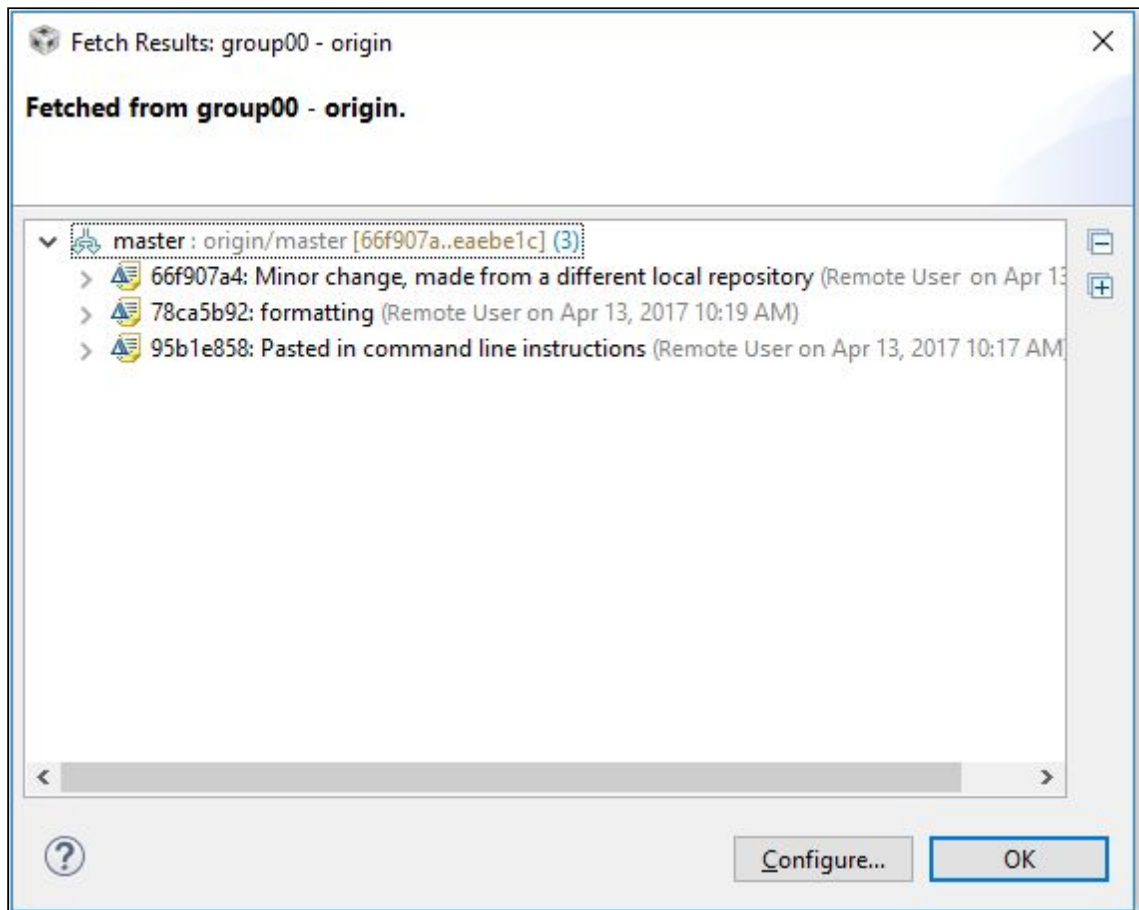
5.1 - Pulling from Remote repository

- 1) Git and CCS don't magically know if there have been changes to the remote repository. You can ask for it to fetch details of any changes, by asking it to "**Team > Fetch from Upstream**".

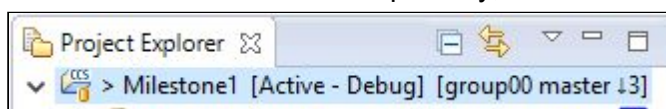


- 2) After entering your Authorisation details (username and password for your repository), the fetch request is processed, and the results are shown. Each commit is listed separately, and if you want to know more about what changes were made in

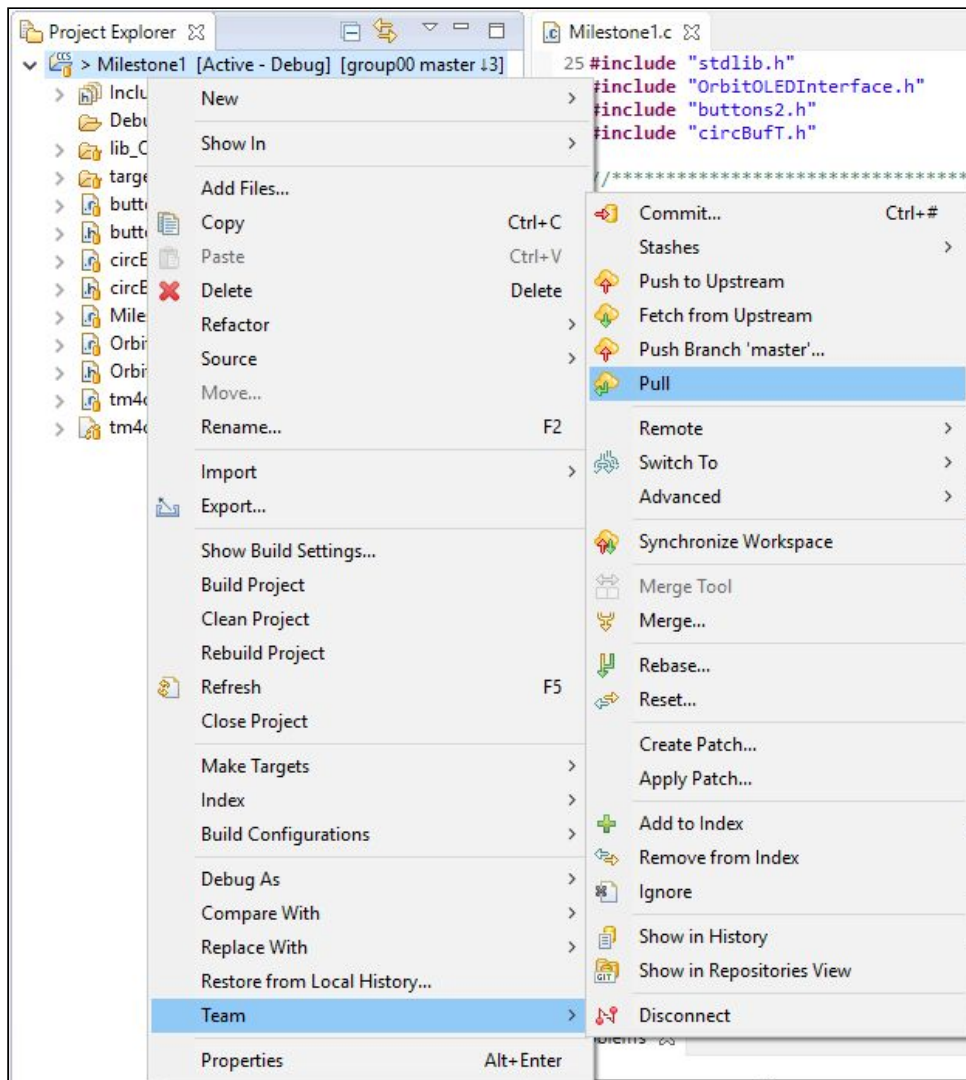
each commit, you can click the “>” next to a commit to expand it into a list of file changes (edited file, added file, removed file).



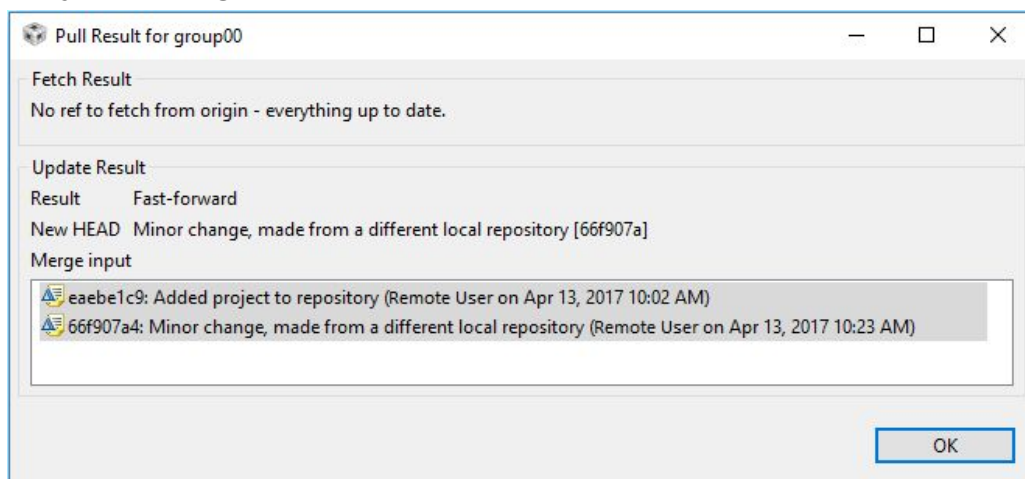
- 3) Now that we have downloaded information about remote changes, note the project name now has a down arrow, with the number of commits that our local repository is behind the the remote repository. In this case, the example local repository is three commits behind the remote repository, so we should try to pull those changes.



- 4) Right click your project, go to “Team > Pull”. Enter your username and password for your repository.



- 5) The pull is executed, and the results are shown in a summary window. The pull process performs a fetch and an update. Because we fetched first, the summary says for “No ref to fetch from origin - everything up to date”. **Skipping the fetch step and just running a pull would have been fine.**



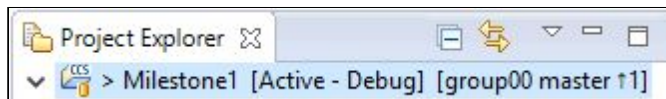
That's it for pulling remote changes. If you fetch first, you will know whether there are changes to pull. If you pull first, it will pull any changes that are available and attempt to merge them with your local version. Sometimes, remote changes will conflict with changes you have made locally.

5.2 - Dealing with conflicts

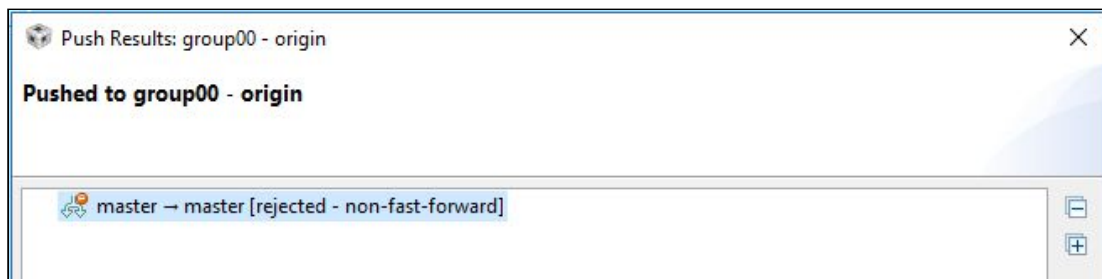
Let's start by assuming we have made changes to a file, and someone else has made changes to the same file, and we want to pull their changes. Conflicts will result in our local repository. It will stop us from pushing our changes, but it won't affect other users of the remote repository.

What does a conflict look like? A simple example is two different values for the same variable or #define. Let's say that we have been working on the Milestone1 project with a teammate, and we both wanted to increase the buffer size, but chose different values. We chose to change it from 20 to 30, and our teammate chose to change it from 20 to 25. They have already pushed their changes to the remote repository, and now we try to push our changes.

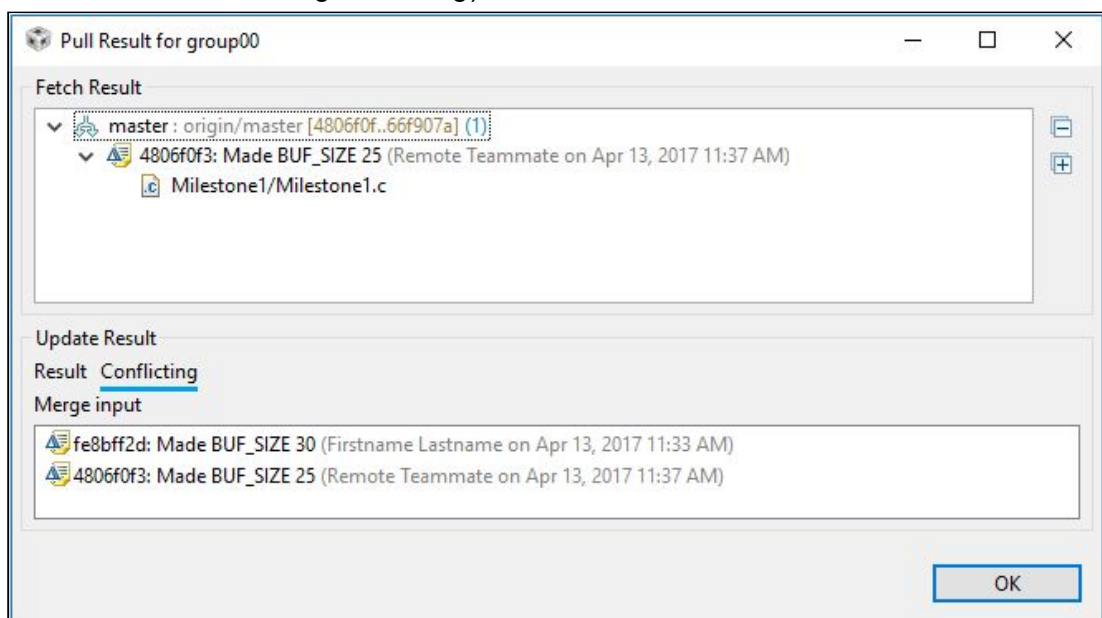
- 1) We have changes we want to push, and haven't fetched from remote. Our project looks like this:



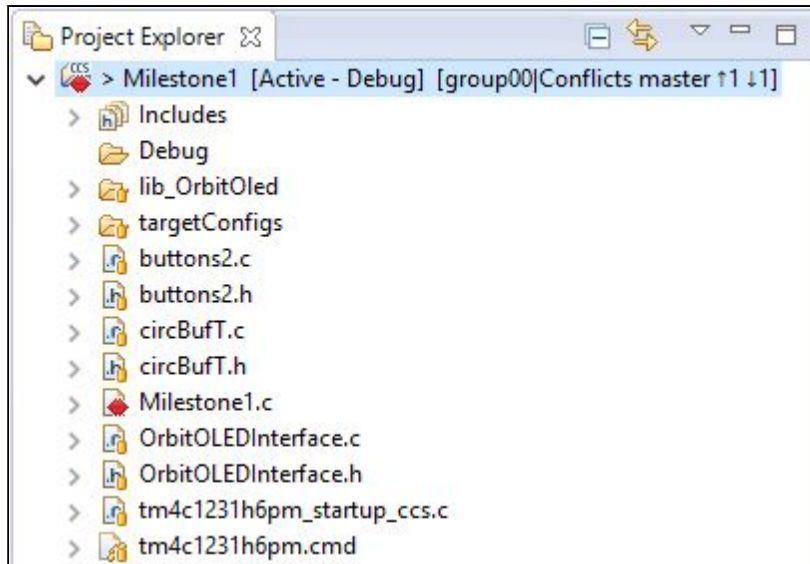
- 2) When we try to Push to Upstream, this happens:



- 3) That message isn't particularly helpful. When we try to Pull instead, thinking that maybe we are just behind the remote repository, we get this Pull Result (*note the mention about something conflicting*):



- 4) Now our project looks a bit different - there is a new red label on the project, and on the file that has conflicts (🔴). This label indicates that there are conflicts. There is also a mention about conflicts in the name section of the project (next to the arrows indicating that you are now both one commit ahead and one commit behind the remote repository).

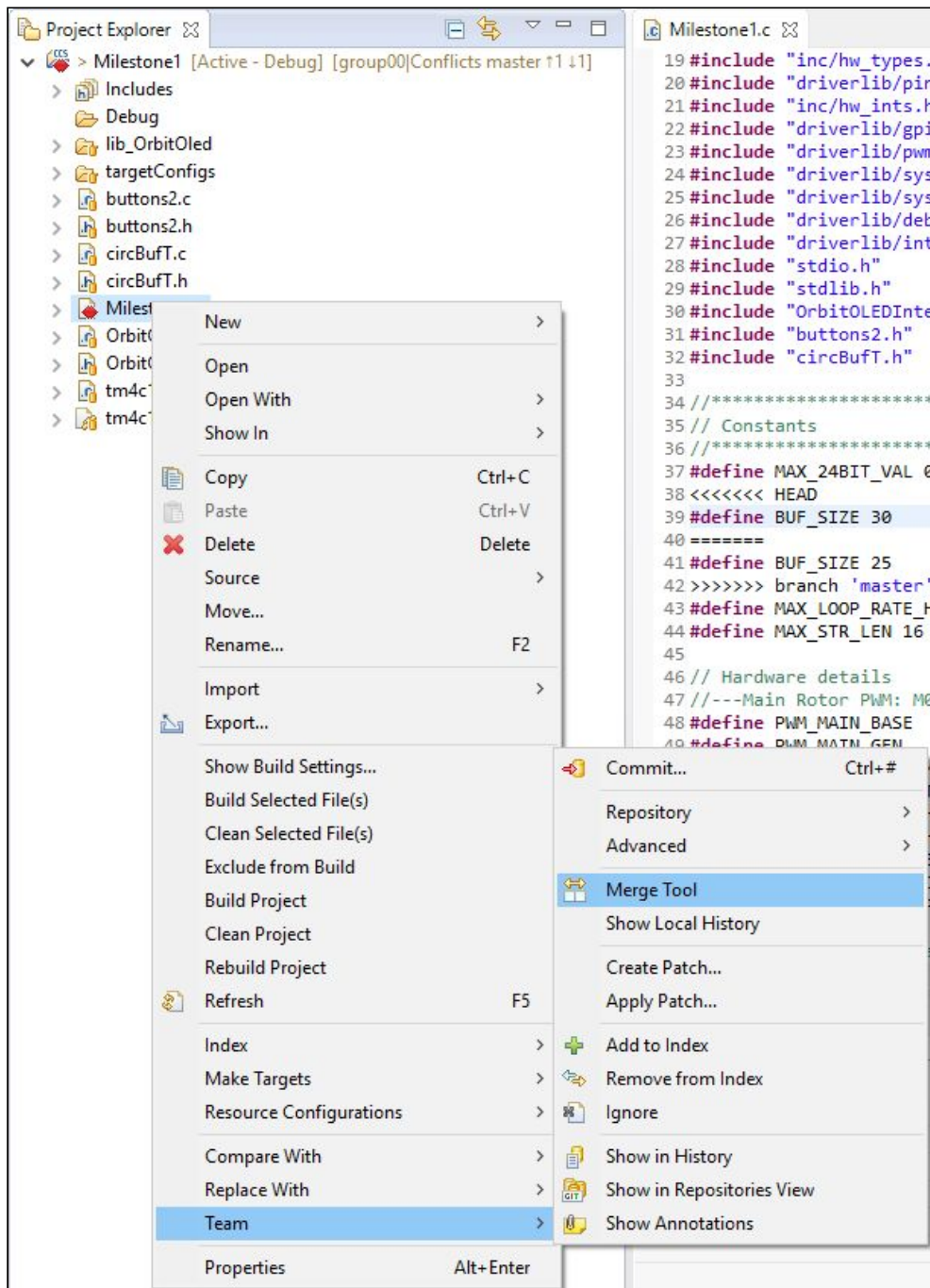


- 5) The conflict is in the file "Milestone1.c", shown below.
The conflict is bordered by "<<<< HEAD" and ">>>> branch ..."
The two conflicting entries are separated by "====". Above that line of '=' are the entries from the local HEAD (our changes), and below the line are the entries from the branch from the specified repository.

```
38 <<<<<< HEAD
39 #define BUF_SIZE 30
40 =====
41 #define BUF_SIZE 25
42 >>>>>> branch 'master' of https://eng-git.canterbury.ac.nz/ence361-2017/group00.git
```

- 6) CCS has a tool included to help process these conflicts. It is called "Merge Tool". It will help show the difference between your local version and the pulled version. Right

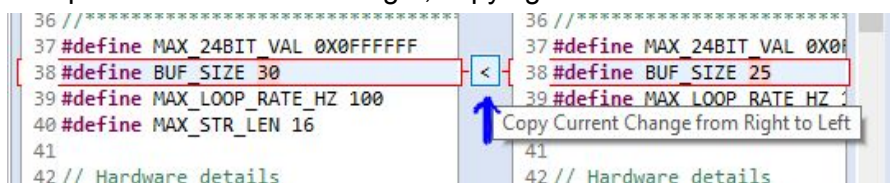
click on the file with conflicts, and go to “Team > Merge Tool”



7) The left side is the local version, and the right side is the remote version. You must make and save at least one change on the left in order to resolve the conflict.

8) We have two options for fixing the conflict here.

a) Accept the version from the right, copying it to the left



- b) Manually edit the code on the left (perhaps we decide BUF_SIZE 20 is better).

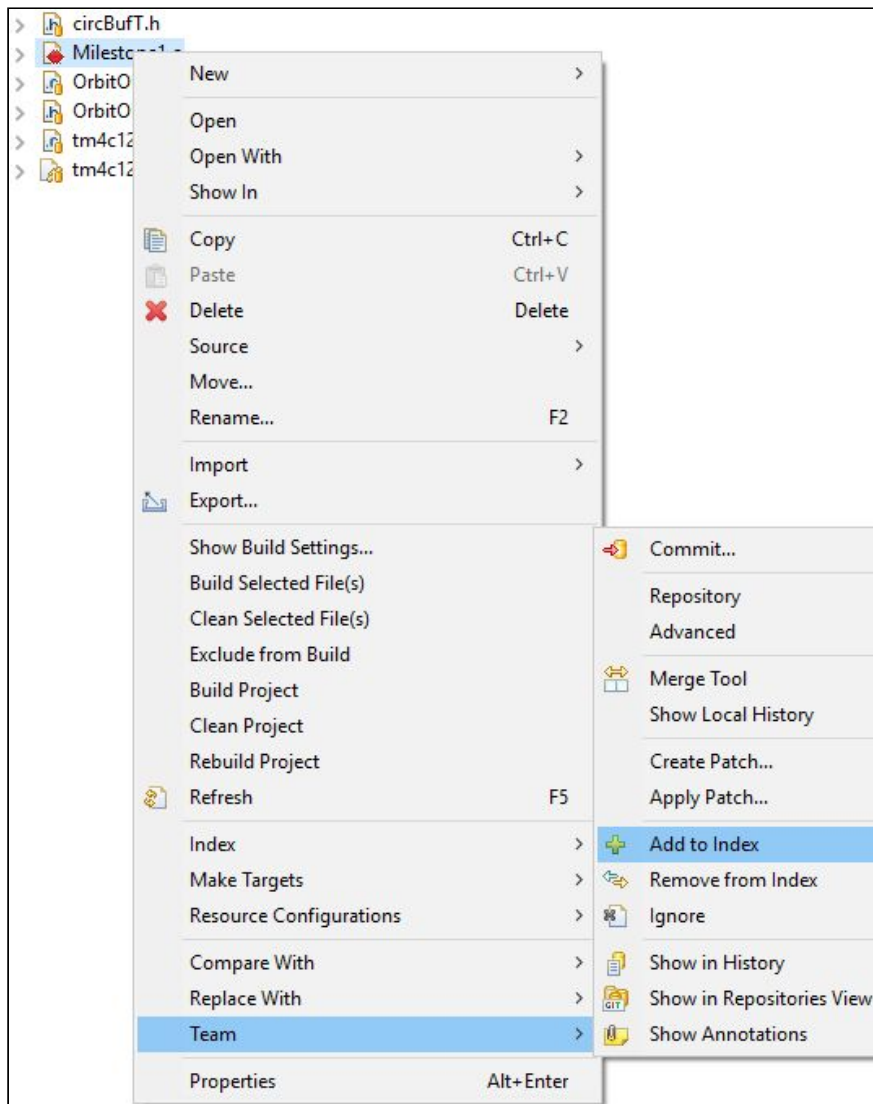
```
36 //*****
37 #define MAX_24BIT_VAL 0X0FFFFFFF
38 #define BUF_SIZE 20
39 #define MAX_LOOP_RATE_HZ 100
40 #define MAX_STR_LEN 16
41
42 // Hardware details
```

- 9) Save the changes in the Merge Tool (using CTRL+S, or click save from the toolbar or File menu). This will apply the changes to the conflicted source file (Milestone1.c). Note that the <<<HEAD, ==, etc. are gone. The merge has been successfully applied to the file.

```
35 // Constants
36 //*****
37 #define MAX_24BIT_VAL 0X0FFFFFFF
38 #define BUF_SIZE 20
39 #define MAX_LOOP_RATE_HZ 100
40 #define MAX_STR_LEN 16
41
```

- 10) Add the file to the index to mark its conflicts as resolved (this must not be done until *after* you have actually fixed the conflicts, or else you will all the conflict markers to

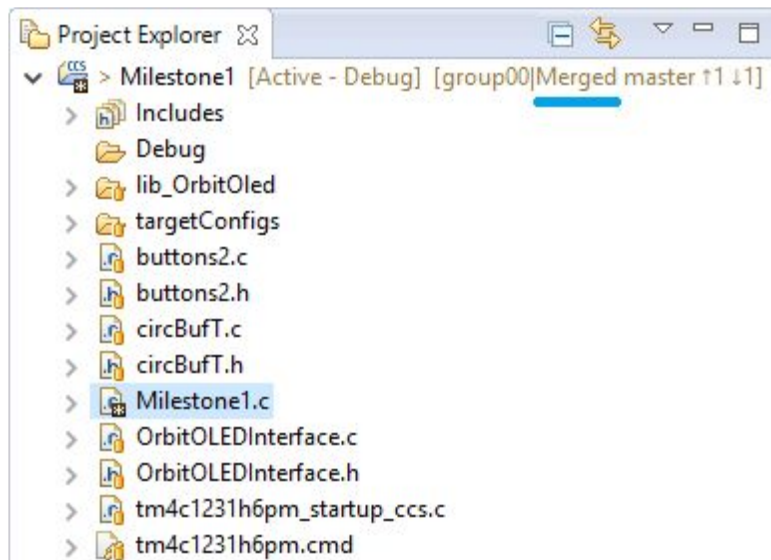
be added and committed).



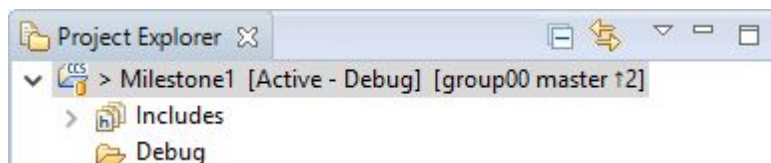
Result of adding:

Note that the project status now says “Merged”, and the “conflict” labels (red) have been replaced with “staged” labels (black square with star). The project should now be ready to

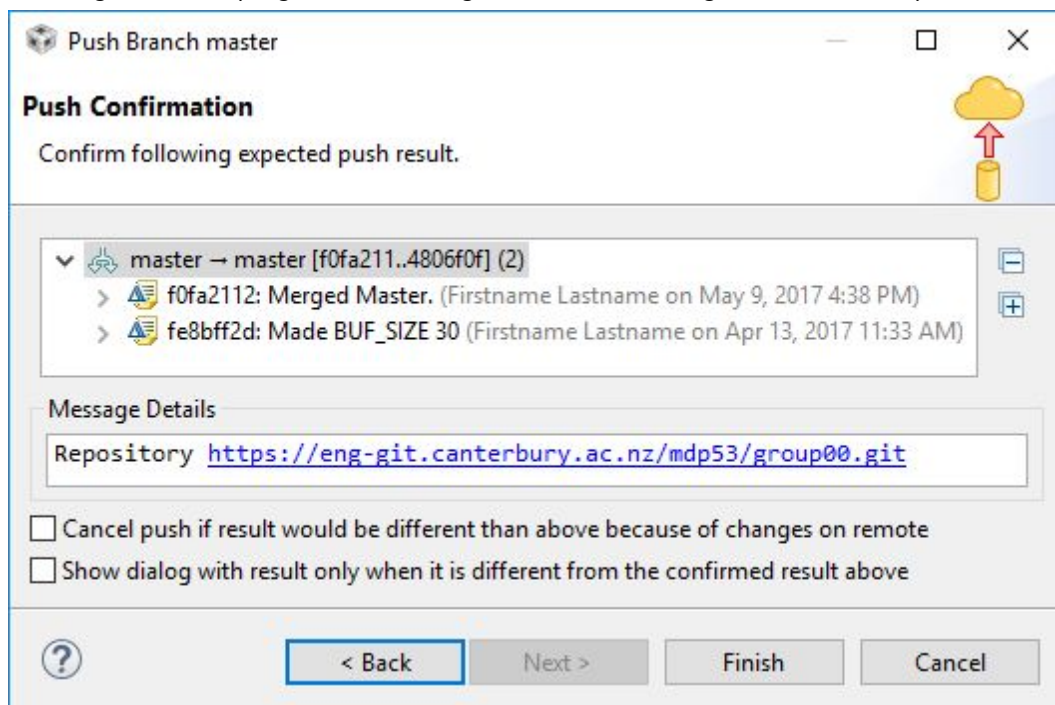
commit, and push.



Committed merge changes:



Pushing commits (original conflicting commit, and merge result commit):



That's it for a resolving a simple merging conflict.

6 - Being clever with developing multiple things

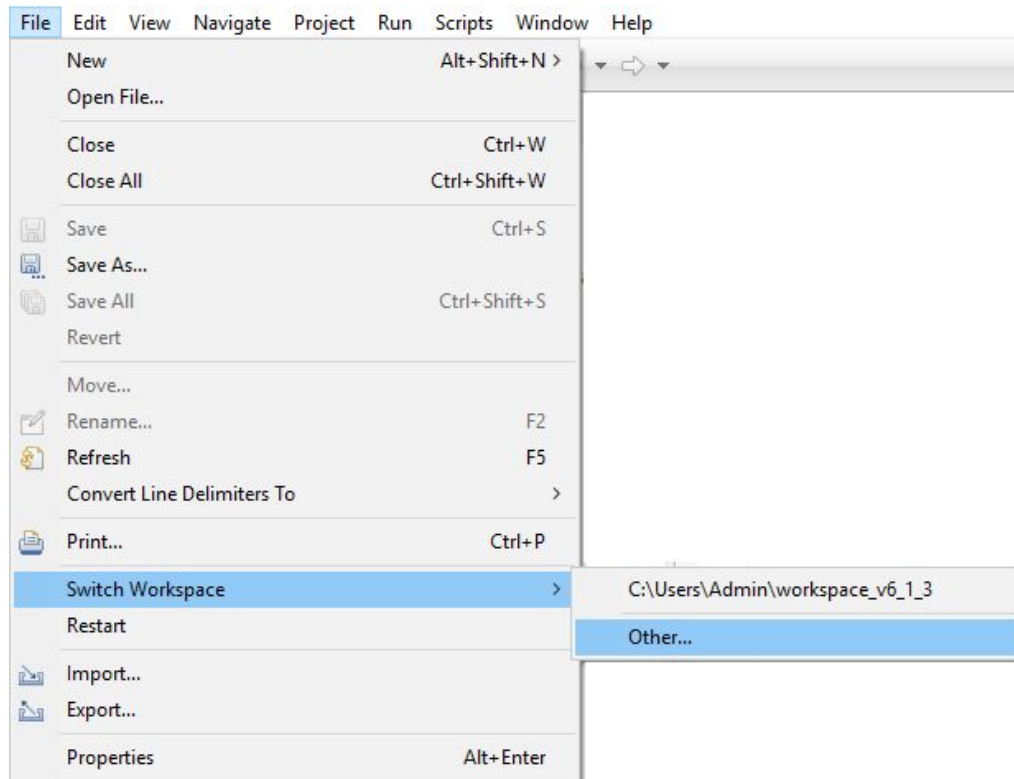
- Being clever with diversions and such:
 - Making a branch, and switching between branches.
 - Stashing, and restoring a stash

6.1 - (WIP) Making a branch, switching between branches

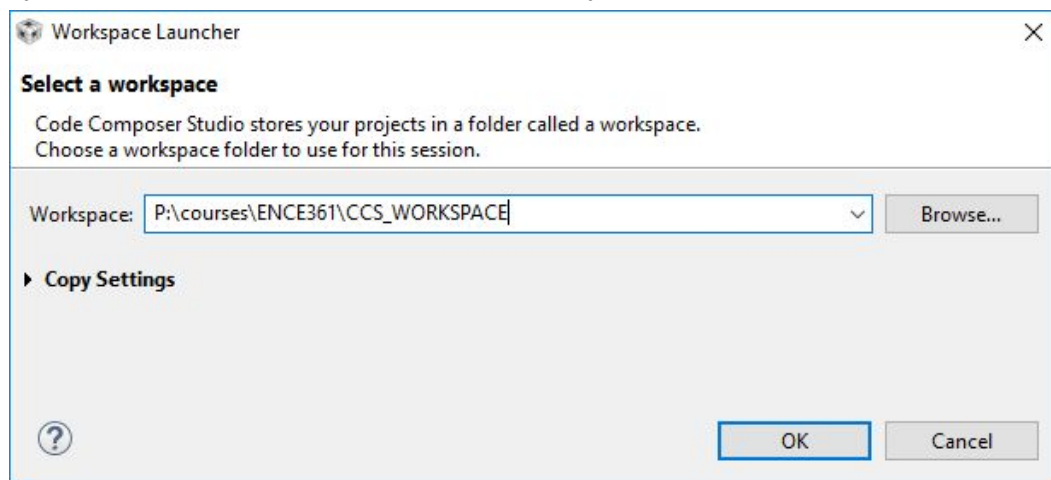
6.2 - (WIP) Stashing, and restoring a stash

Optional: New CCS workspace

- 1) Go to “File” > “Switch Workspace” > “Other...”:



- 2) Type in the path to, or browse to, the directory of the new workspace:



- 3) Click “OK” and Code Composer Studio will restart, configured to use that workspace.