**Design:**

Decide what Eclipse projects should be logically grouped together in your Git Repository. A repository usually builds a single artifact (EAR, WAR, JAR, ZIP, etc). A typical JEE Application would consist of a Build, an EAR, a WAR and optionally multiple EJB / Utility projects.

Typically, you will migrate both the short-term code path and the long-term code path into Git. These instructions begin with the short-term path = production = master branch in Git. Then the long-term path = develop branch in Git, will be added to the Git repository. A single Git repo contains both code paths. These instructions will require you to have a local file system representation of both code paths. This will be usually achieved by doing a Harvest “check-out for browse” to a local location (e.g. C:\temp)

**RAD Workspace:**

The recommended way of getting a Harvest code base into Git via RAD is to start with a workspace that has no knowledge/reference to the projects you wish to migrate. You can and will certainly have other projects in this workspace. The workspace doesn’t need to be completely clean, just clean with regards to the projects you are migrating.

**Cleanup of the Harvest code bases:**

Before putting a code base under Git source code versioning, it’s recommended you remove any reference to harvest from these projects.

At a minimum you will need to do:

* Change the read-only flag on each file (>attrib -R /s)
* Remove all harvest.sig files (>del /s [/a:h] harvest.sig)
* The .harvestignore or .scmignore files can be kept temporarily until you create a .gitignore file.

**Obtain a Git Repository:**

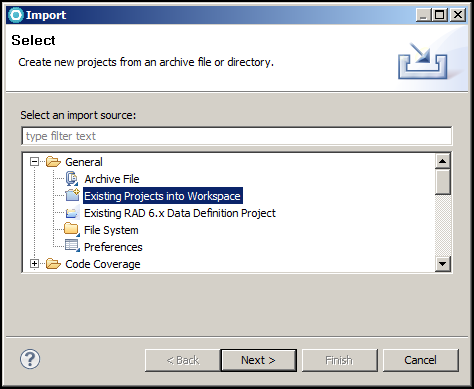
Each Stash project will have one or more “Project Administrators”. Have one of these administrators create you a Git repository.

**Minimum Git Setup in RAD:**

Ensure you have followed the Developer Setup instructions on the Wiki. There are five (5) documents that you need to follow.

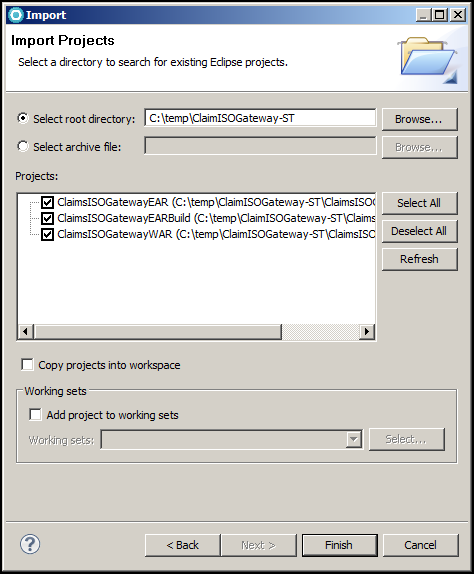
**Importing the short-term code base into RAD:**

* File > Import…



Click Next.

In the dialog below, the C:\temp\ClaimISOGateway-ST represent the location on my local file system where short-term code base from Harvest has checked out and cleaned. Make sure you uncheck the “Copy projects into workspace” option. This will avoid an unnecessary file system copy.



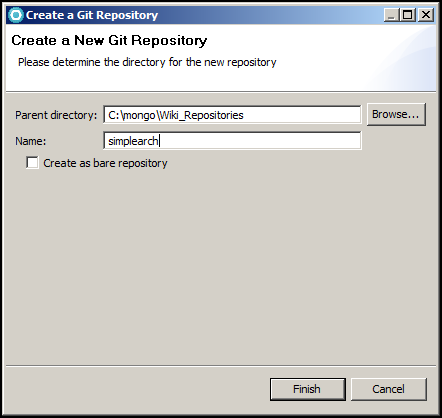
Click Finish

**RAD Projects:**

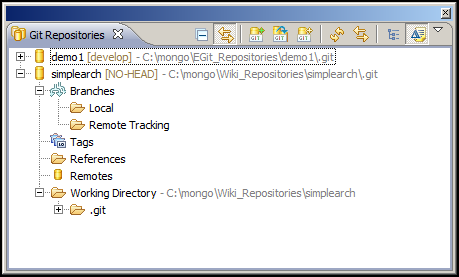
At this point, you need to make sure your workspace builds clean. I would suggest testing your code with a build or deploying to a local server.

**Create a new Git Repository:**

* In stash, click in the SSH URL for your repo, then Ctrl-C to copy it to the clipboard.
* In the Git Repository view, click “Create a new Git Repository”. The dialog should look similar to the following:
* Keep the name all lowercase, punctuated with dashes. This is your repository name. It doesn’t have to match your Stash repository, but often it does.



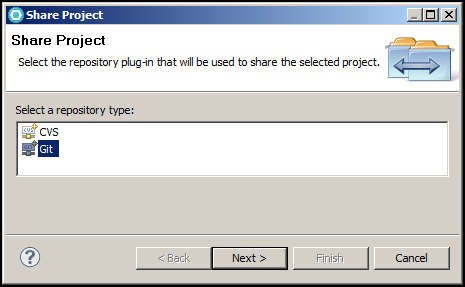
Finish



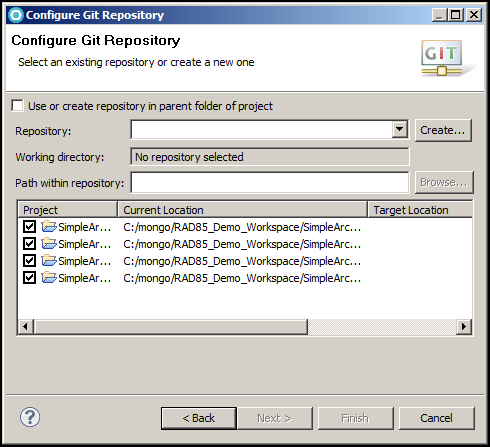
If your Git Repository view doesn’t look similar to above, STOP, contact your Team’s Git Admin or the Architecture Team’s Git Expert to resolve / fix your differences.

Share the Projects with Git:

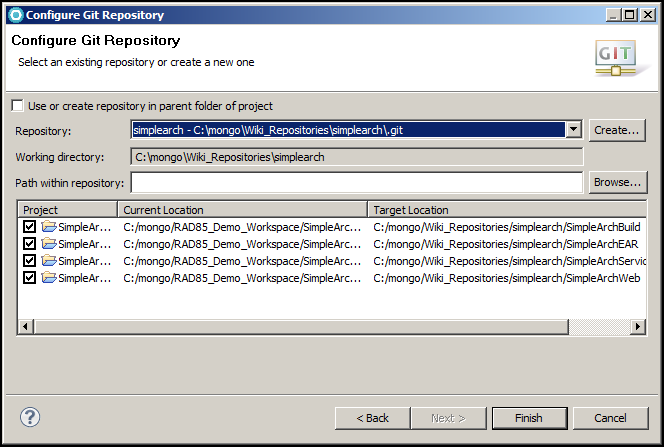
Select all projects, right-mouse, Team > Share project…



Make sure Git is selected. Next >



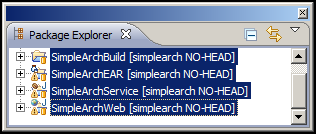
Click the Repository drop-down. Select your Repository.



VERY IMPORTANT!!!

Before clicking Finish, ensure the Target Location is correct. EGit is about to move your projects out of your Workspace folder into the default repository folder. Make sure it’s correct.

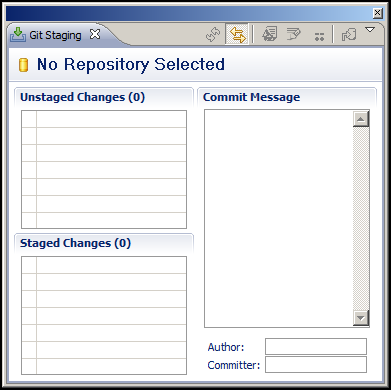
Click Finish.



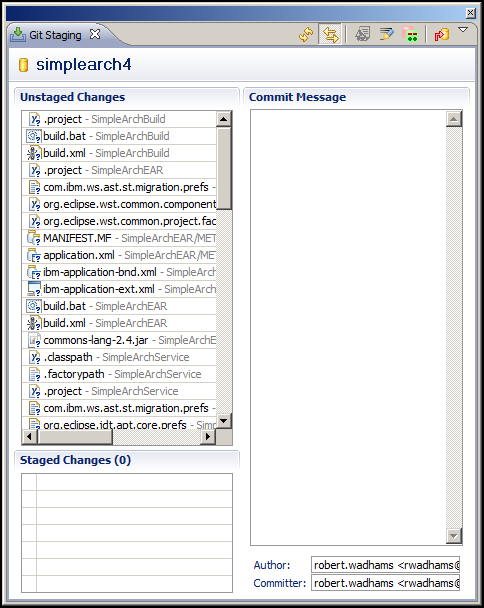
The label decoration may show either NO-HEAD or master.

**Stage and Commit your code locally:**

* Click on the Git Staging view



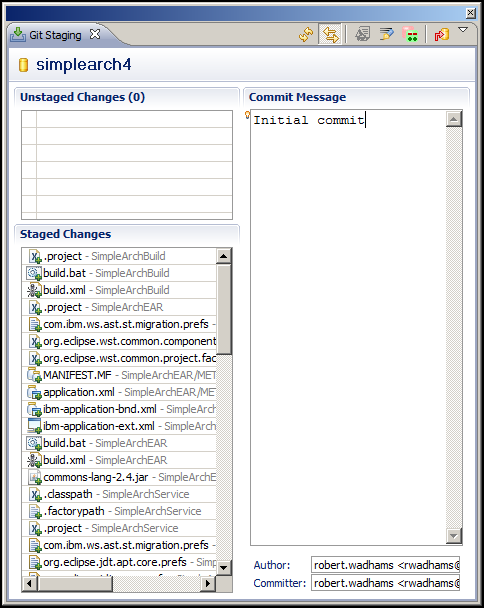
If you see everything empty, you need to select the repository in the Git Repository view before clicking Git staging.



At this point, you are preparing for your first commit into the Git repo.

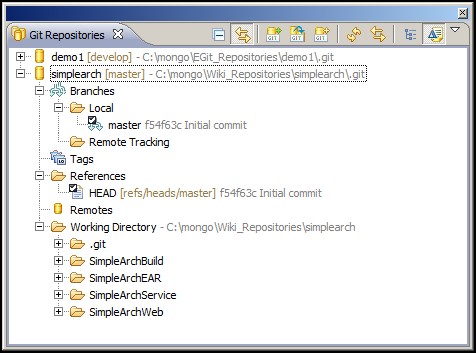
Take your time and carefully select what files you want under version control.

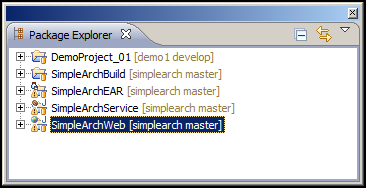
* Files and folders that shouldn’t be version controlled (e.g. bin, dist, build, testResults, etc) should be added to a .gitignore file. To do this return to the Package Explorer or Navigator view, right-mouse on the specific resource to ignore, Team > Ignore.
* Alternatively, open the .harvestignore(s) or .scmignore file(s) with a text editor and inspect it for accuracy. Save as… .gitignore. Now you should delete the Harvest ignore files as they are no longer required.
* Drag and drop files from the “Unstaged Changes” window to the “Staged Changes” window. Alternately, select files in the “Unstaged Changes” window, right-mouse, Add toGit Index. This will move them to the lower window.



Once you have all your code staged, type a commit message (mandatory) and click the red arrow commit icon.

Your Git Repository view and Package Explorer view should now look similar to below:

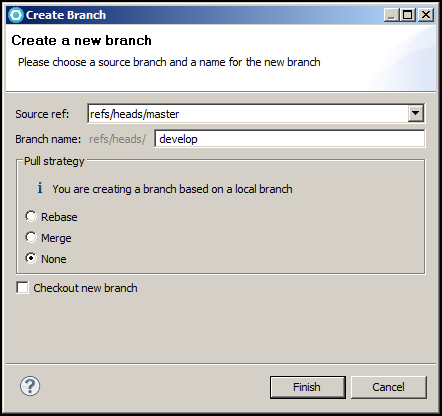




**Create a “develop” branch based off the “master” branch:**

The “master” branch represents the Production code base including short-term development and hot-fixes. The “develop” branch represents long-term development. We will create the “develop” branch now, but not bring in your long-term code base until later. We will push and configure the “develop” branch.

* In the Git Repository view, right-mouse, Local > Switch To > New Branch…
* Type: develop
* Uncheck > Checkout new branch
* Finish



**Interactions with a Remote:**

There two (2) primary interactions that you will do with a Remote:

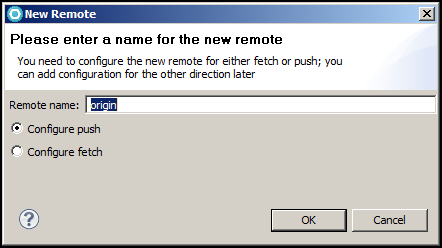
1. Fetch the latest changes from the Remote. Then Merge those changes into your local repository. Git provides a convenience command that combines both a Fetch and Merge called Pull.
2. Push your local changes to a Remote. Since you have at a minimum two (2) branches (master and develop) you need to configure your local master to push to the remote master and similarly for the develop branch.

At this stage you have:

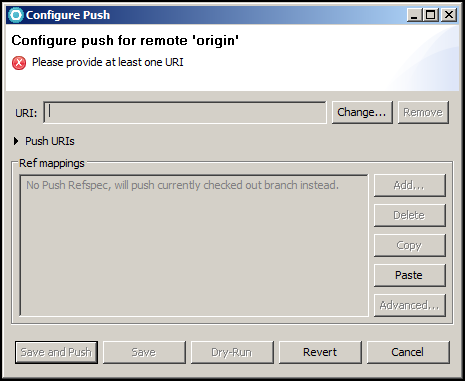
* An empty Remote repo. Check Stash to be sure.
* An initial commit.
* Two (2) local branches (master and develop). The HEAD of both these branches points to the initial commit.
* Nothing in our local Git repository ties it back to the remote Stash repository. We will setup the remote configuration now.

**Adding a Remote:**

* Return to Stash and copy the SSH URL of your repository to your clipboard (Ctrl-C). Return to RAD.
* In the Git Repository view, right-mouse Remotes > Create remote…

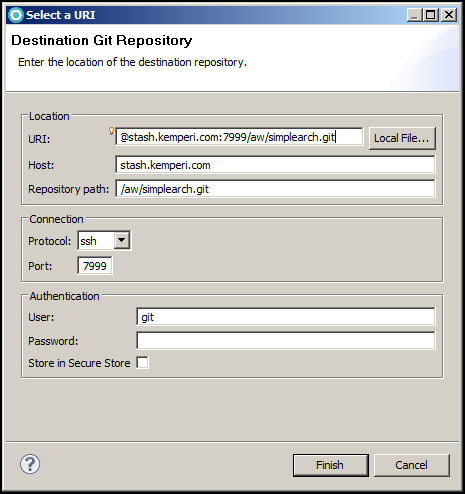


Click OK

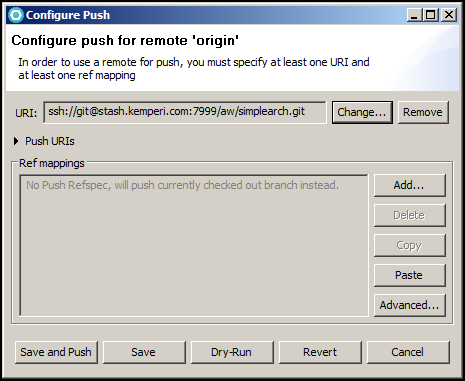


Click Change…

Paste the Stash repository URL into the URI: field.

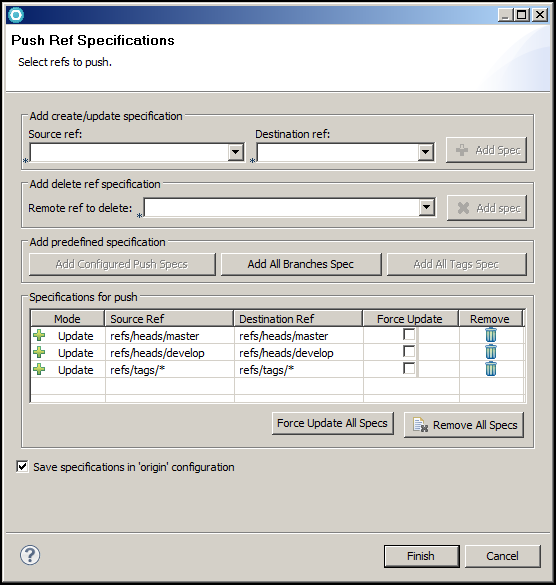


Click Finish

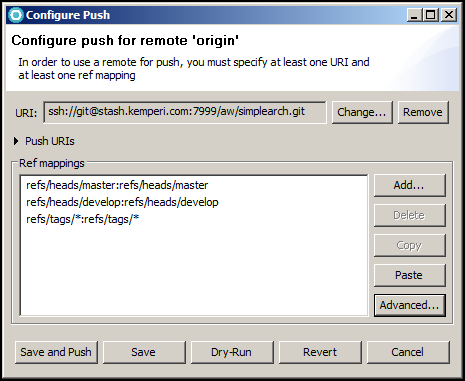


Click Advanced…

Make your Push Ref Specification look similar to below by using the “Add create/update specification” section and the “Add All tags Spec” button.

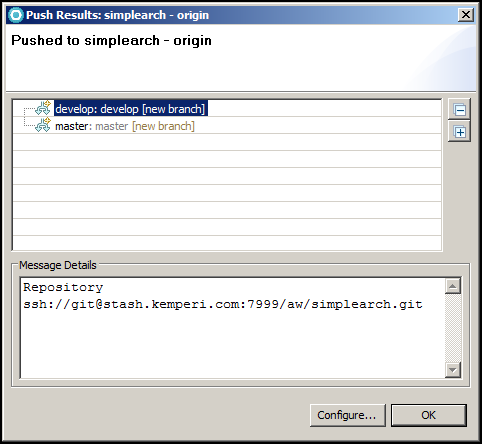


Click Finish



Click “Save and Push”

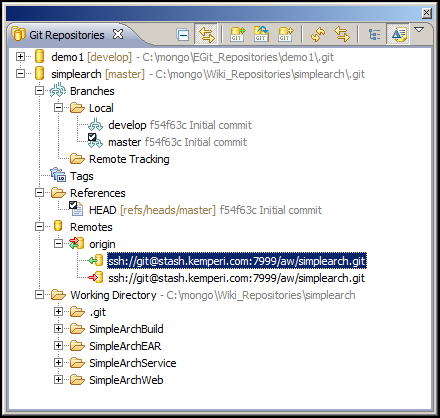
Click OK on any dialogs that may appear referring to SSH. This happens when you connect and use your SSH keys for the first time with a new remote.



Click OK

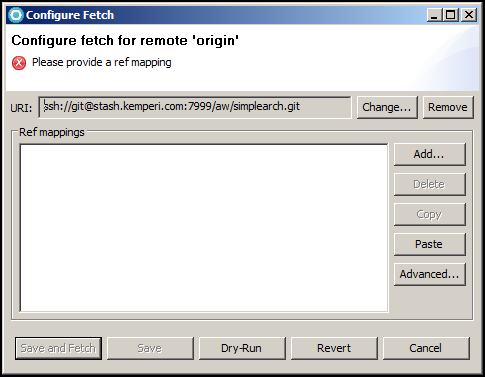
Return to Stash and refresh the page. At this point, you should see files, a single commit (initial) and two branches in the dropdown. Return to RAD.

Your Git Repositories view should now look similar to below. Note: the Remote Tracking folder under Branches is still empty. We will now configure Git to track our local master and develop branches to Stash’s remote master and develop branches.



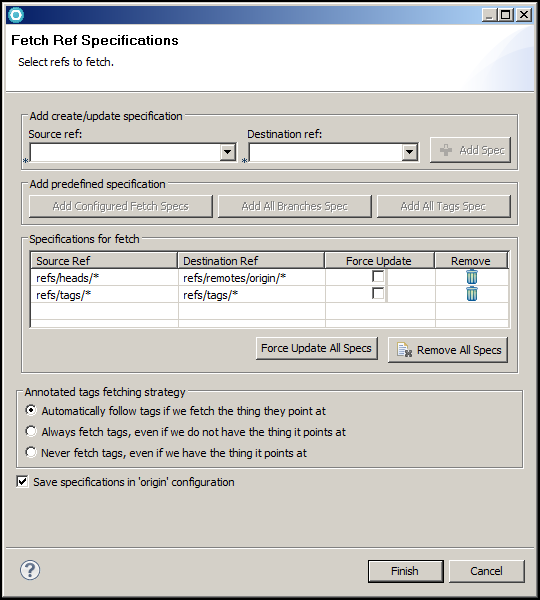
**Setup Fetch Configuration:**

* In the Git Repository view, expand Remotes, Expand origin
* A green Fetch URL should exist. Right-mouse, Configure Fetch…

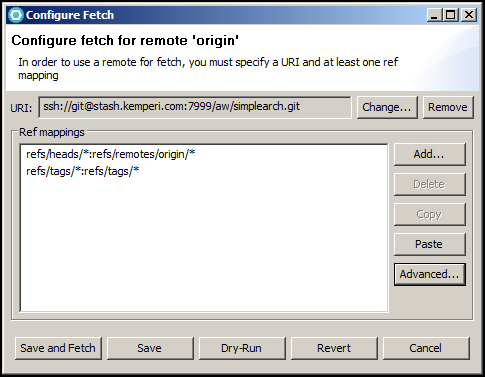


Click Advanced…

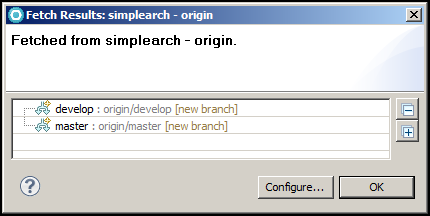
Make your Fetch Ref Specification look similar to below by using the “Add All Branches Spec” and “Add All tags Spec” buttons.



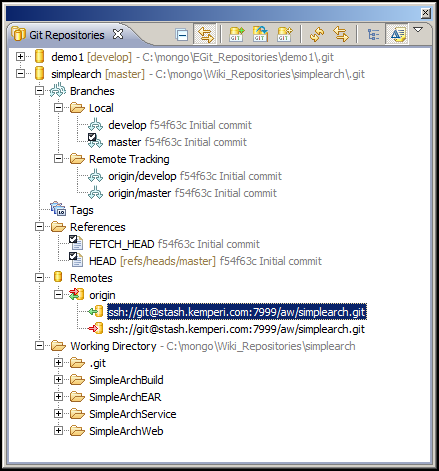
Click Finish



Click “Save and Fetch”

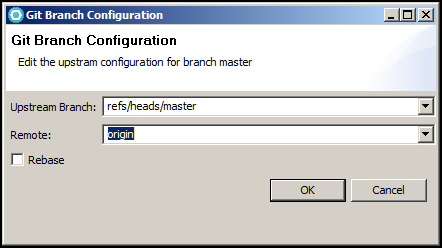


Your Git Repositories view should now look similar to below:



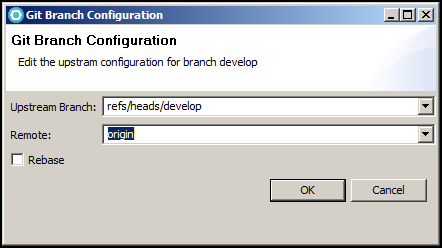
**Prepare each branch for Pull:**

* In the Git Repository view, expand Branches, Local, Right-mouse on the master branch, Configure Branch…
* Make your Git Branch Configuration look exactly as below.



Click OK

Do the same for the develop branch

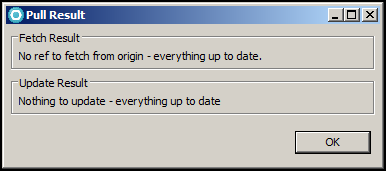


Click OK

**Verify “Pull” is configured correctly:**

* The Pull process works at the Repo level and only fetches and merges the currently checked out branch.
* In the Git Repository view,>
* Right-mouse on the Repo (top level folder), > Pull

You should see a dialog similar to below. Since there are no changes on the remote compared to locally, The “Pull Result” dialog reports that everything is up-to-date.

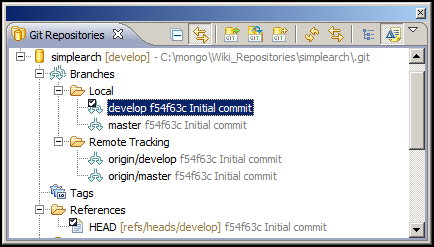


**Bringing in the long-term code base:**

VERY IMPORTANT!!!

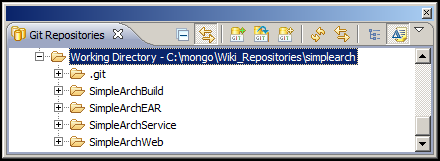
Make sure you switch to the “develop” branch (git checkout) before doing any file system changes. See below.

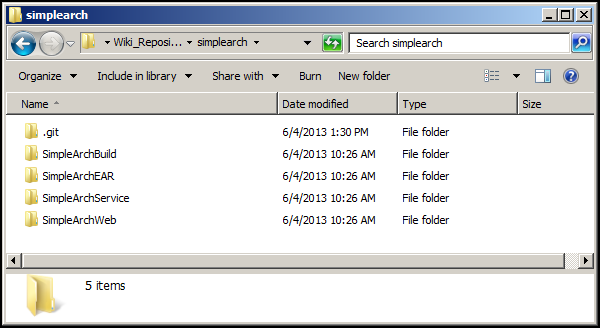
* In the Git Repositories view,
* Right-mouse the local “develop” branch > Checkout
* The tick mark should now move to the “develop” branch.



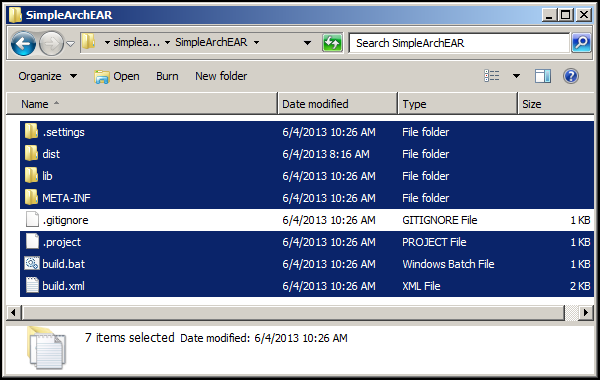
Now bring up Windows Explorer in preparation for replacing the existing code base with the cleaned long-term code base.

* The working directory is available in the Git Repositories view. Navigate to that location.





In this example, all 4 projects need to have all their contents deleted, except the .gitignore file. See below:



Delete the highlighted files.

For each project, you need to copy the long-term code base back in.

You should have a location on your local file system (e.g. C:\temp\ClaimISOGateway-LT) where your long-term code base from Harvest has been checked out and cleaned. Copy these files into the appropriate project folders.

Now return to RAD and refresh your file system (F5) and every view. Your staging area should reflect all the changes from your long-term path.

* Stage the changes by dragging them from “unstaged” to “staged”.
* Create a commit.
* Push the branch to the remote (Stash).

**Stash:**

Go back to Stash and refresh the repository page.

* Under Files,
  + Branch drop-down with both master and develop.
  + Folder structure representing your projects.
* Under Commits,
  + Author, commit SHA, commit message and commit date.

**Other Considerations:**

1. If you added a new Eclipse project into your workspace. The project is known to RAD (compiles, tests, etc), but Git has no knowledge of it. Follow the procedures above starting with Team> Share projects…