**Purpose:**

This document explains the process of setting-up and using GIT as BCSG code repository is going to be migrated to GIT from SVN.

Since we are already well-versed with SCMs like CVS and SVN, this document focuses on those points which are new or different as compared to SVN.

**Setting-up GIT:**

Set-up is straightforward with dropping EGit Eclipse plugin in the ‘*dropins’* folder of Eclipse.

If using STS (SpringSource Tool Suite), GIT plugin is by default installed.

Key points to know before using GIT

* Distributed version control system unlike SVN where client connects to a central repo

Instead, individual developer will have own local repository where source code is checked-in and checked-out and then pushed to a central remote repo shared among developers

* By default, local repo is created in USER\_HOME/git/ directory if you haven’t specified GIT HOME in your system variables
* Location where your project resides is called workspace. When you commit changes in project (Workspace), they are moved from workspace to local repo. Moving changes from local to remote repo is called ‘Pushing to upstream’ and taking changes from remote to local repo is called fetching/pulling.
* A *fetch* on remote repo fetches all changes to local from remote repo while a *pull* on remote repo fetches changes from remote to local and then to your workspace
* In Eclipse, you can work with EGit in regular Package Explorer/Navigator views. Additionally,

EGit provides Synchronizing, Git Staging and Git Repository views. Use of synchronizing view is discouraged due to few issues associated except to comparing files.

* *‘Add to Index’*  in GIT is equivalent for *‘Mark as Merged’* in SVN/CVSwhile resolving conflicts

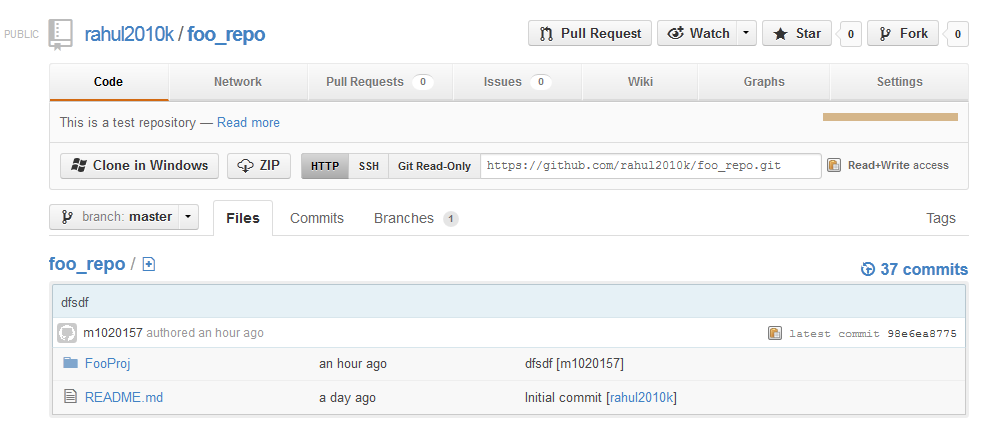
**A step-by-step guide to show creating and accessing GIT repositories:**

1. At Github.com, you can create free account and host your remote repository.

e.g. [***https://github.com/rahul2010k/foo\_repo.git***](https://github.com/rahul2010k/foo_repo.git)

Here, *rahul2010k* is the account name and *foo\_repo* is repo name.

Refer to the screenshot from github.com below.



1. In Eclipse, create a new java project e.g. FooProj. In navigator window, right click > team > share project to add the new project to the above remote repo.
2. To simulate a dev environment, create a new workspace for developer Dev1 e.g. git\_wkspc1. Create another workspace for developer Dev2 e.g. git\_wkspc2.
3. In git\_wkspc1 for Dev1, right click > import > Git > projects from Git > URI > Next. Enter the above remote repo url and click Next and again Next. In the destination directory, change the default location to {user\_home}\git\**foo\_repo to** {user\_home}\git\**foo\_repo1.**

So, foo\_repo1 is the local repo for Dev1 that’s connected to remote repo.

Follow the same step in git\_wkspc2 to create foo\_repo2 for Dev2.

So, in your user\_home/git directory you will see two local repositories

1. Now, in Dev1’s project, create a java source file with some code. Right click on the project > team > commit. Add comment and click Ok. This will commit the source in Dev1’s local repo i.e. foo\_repo1. However, to be able to share the source with Dev2, it has to be moved to remote repo. In navigator, you will see an up arrow beside the project icon indicating there are files committed but yet to be pushed to upstream.

So, again right click > team > push to upstream. This will move your changes to remote repo i.e. foo\_repo

1. Now, in Dev2’s workspace, right click on project > team > fetch from upstream (You can use Pull as well. Difference explained below). This will bring Dev1’s changes from remote to local repo for Dev2 i.e. foo\_repo2. However, changes are not yet in Dev2’s workspace (project). Again right click > team > synchronize workspace. This will open new view showing incoming changes. Right click and click on merge to take incoming changes.

**Resolving conflicts:**

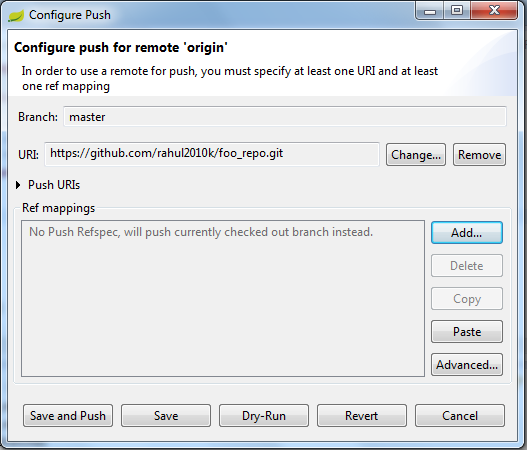
1. In case of conflicts, GIT tries to merge wherever possible otherwise you need to do it manually.

Change the source code for Dev1 and Dev2 such that it will create conflicts.

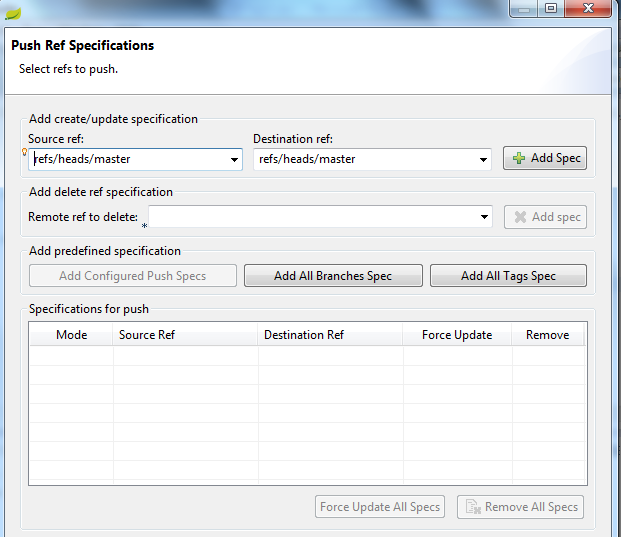
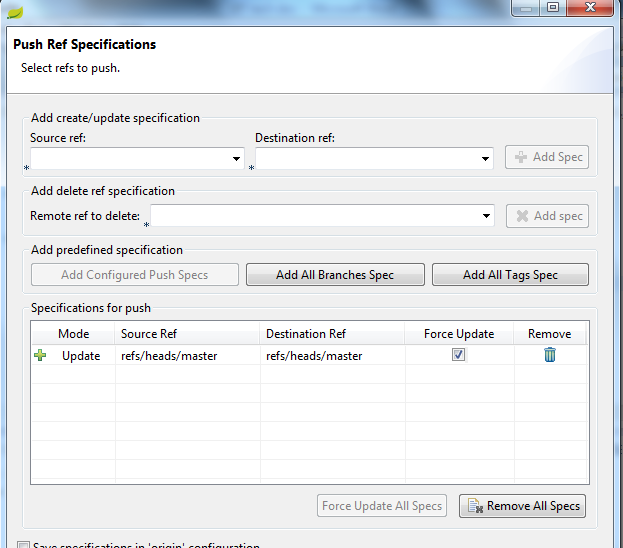
1. Commit the changes for both developers into their respective local repos.
2. Push the changes upstream into remote repo for one of the developers e.g. say Dev1

Now, Dev2’s changes are committed but yet to be pushed to upstream. So, when Dev2 pushes the changes to upstream, GIT shows error messages like **‘rejected: cannot fast-forward’** telling Dev2 that there is incoming and conflicting code that first has to be merged manually.

1. To manually merge, first fetch from upstream into local repo and synchronize the workspace. This will open new view highlighting conflicting files in red. Open the conflicting files in compare view and manually merge the files.
2. After merging, right click on project > team > Add to Index (equivalent of ‘Mark as Merged’).
3. You will need to commit the changes again to the local repo
4. Now to push the merged changes from local repo to remote, simply clicking on ‘Push to upstream’ won’t help. Follow below steps:-
5. Right click on project > team > remote > configure push to upstream. This will open a dialogue shown below.



1. Click on advanced to open new dialogue shown below. Select ‘master’ from two drop-downs and click on ‘Add spec’. Select the check-box ‘force Update’ and finish.



1. You will be taken to first dialogue. Click on ‘save and push’ to push your merged changes to upstream.

More to follow…