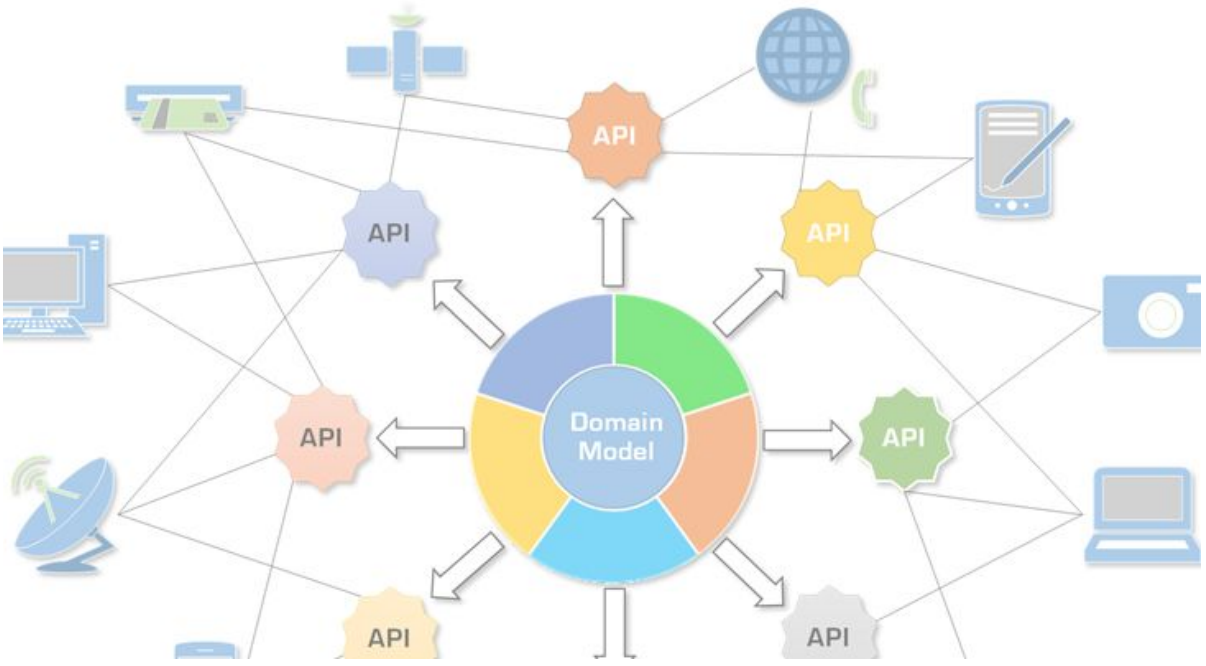
RepreZen™

Working with Git in RepreZen API Studio

RepreZen White Paper
July 2018



Contents

Git Workflow

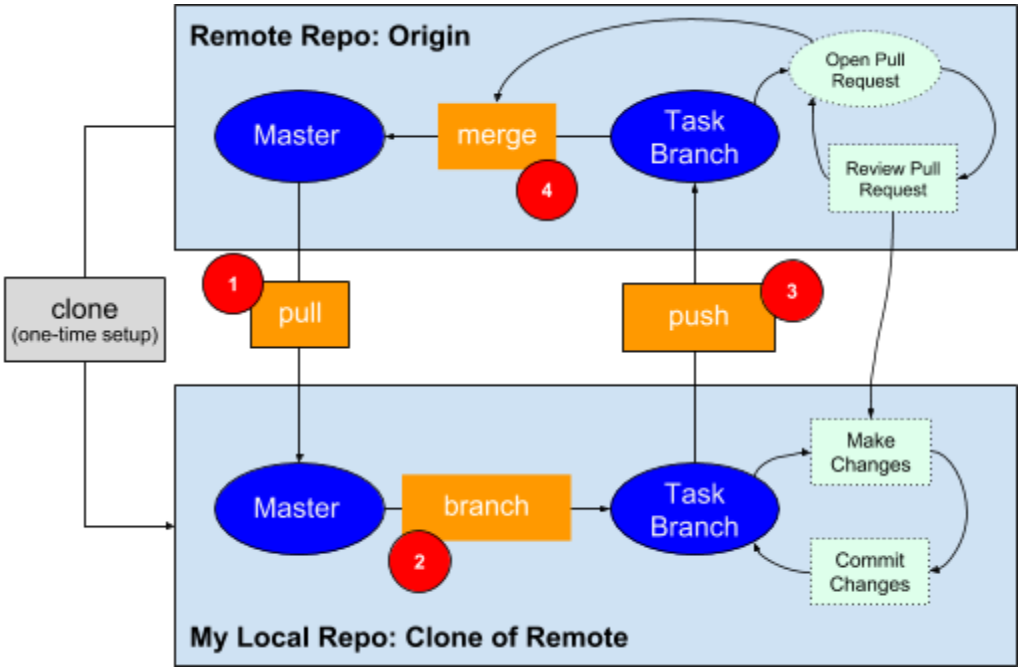
As a distributed version control system, or DVCS, Git differs from conventional version control systems in two important ways:

1. Each user works with a local repository, which may contain multiple branches and a detailed history of changes within each of those branches. There is still a central repository, called the *psjhjo/*. But you can retrieve historical versions, switch freely among branches, and do other essential version control tasks without accessing the remote origin repository.

2. Branching, which is a heavyweight, expensive operation in other version control systems, is very lightweight and efficient in Git. Development teams can make liberal use of branches for day-to-day tasks, which makes it much easier to manage distributed projects.

The 4/4 Rhythm of Task-Based Development

Task-based workflow in Git usually works as a 4-phase process:



1. Pull the most recent changes from the *nbtufs/* branch on the remote origin repository to your local repository.
    - Unless you've made local changes to your master branch that conflict with recent changes in the remote master, this should be fast and straightforward.

2. Create a local *ubtl!csbodi*, make the required changes to complete your task, and *dpnnju* those changes to the task branch.

- This forms a mini-cycle, wherein you may commit multiple incremental changes to the branch.

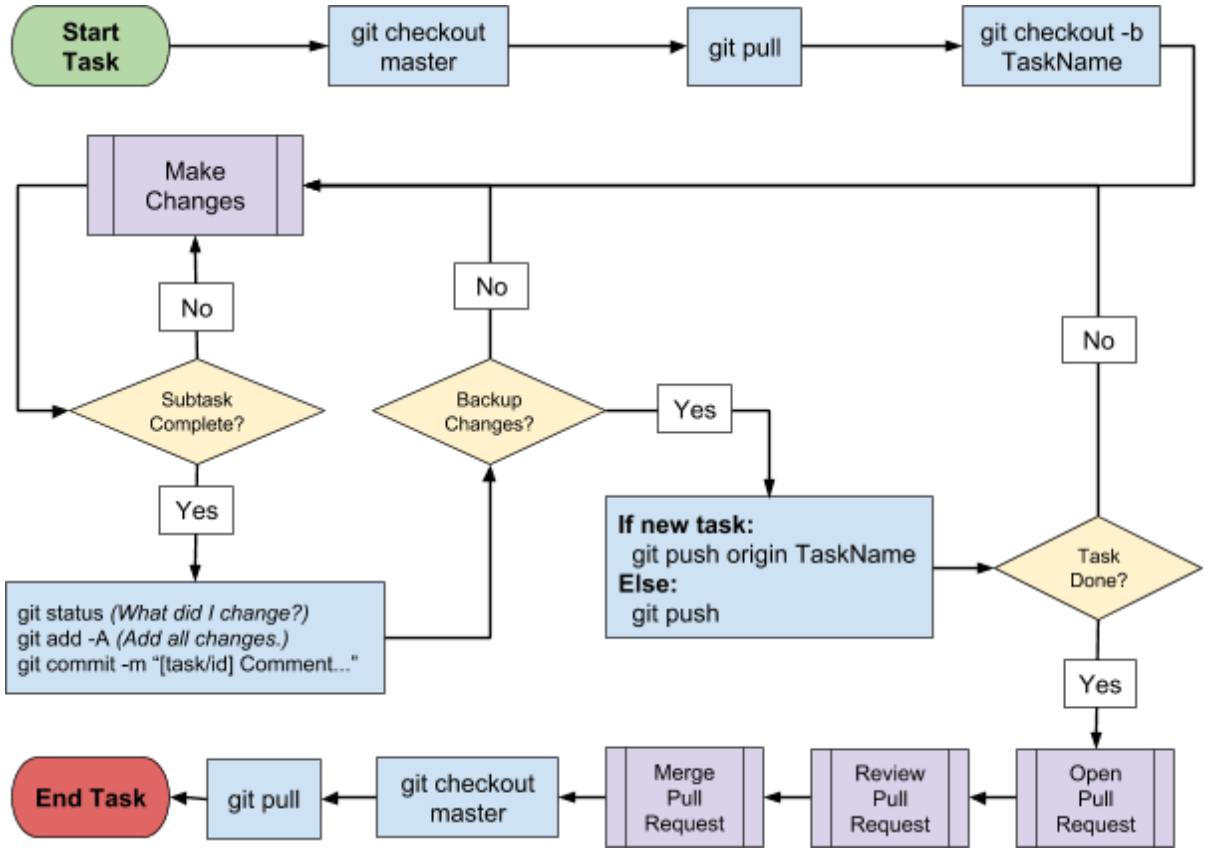
3. Push those changes from your local task branch to the remote origin repository. This creates or updates the corresponding branch on the remote repo.

4. Once your changes have been reviewed and/or tested, merge the task branch back to master. Github uses a *qvmmsfrvftu* to coordinate the review and merge process.
   - The review process can be incremental, extending the mini-cycle described in step 2.
   - Prior to merging, if code review and testing expose some problems, you can commit ongoing changes to the local task branch, and push those changes back to the origin, repeating this cycle as needed.

Over the course of a project, this high-level process sets a steady "4/4 rhythm" for the team.

Editing with Git Commands

Here's a more detailed workflow diagram with some essential git command included:



Using Git in RepreZen API Studio

RepreZen API Studio ships with EGit, a popular git client for Eclipse. While EGit has its own complete User Guide, this section provides a brief guide to setting up your environment to work with EGit, and performing essential git-based editing and workflow functions.

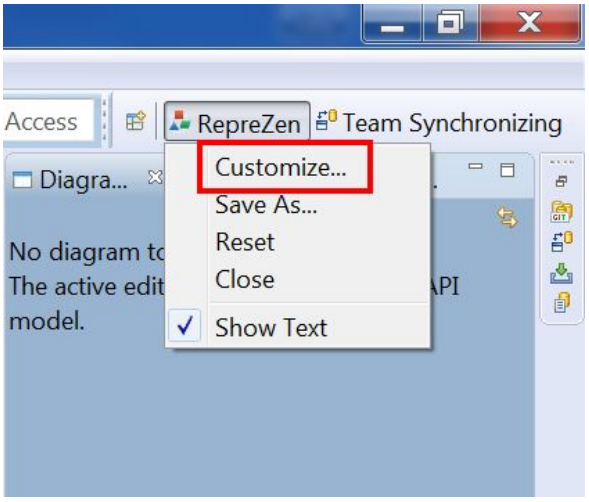
Opu f;!5Git works well with github as a central, remote repository. But it does not support github-specific features such as issues, pull requests, and review comments. The GitHub connector for Mylyn, also included, supports some of these features, but is not covered in this guide. See the knowledge-base article here for more information.

Configuring the Eclipse Environment

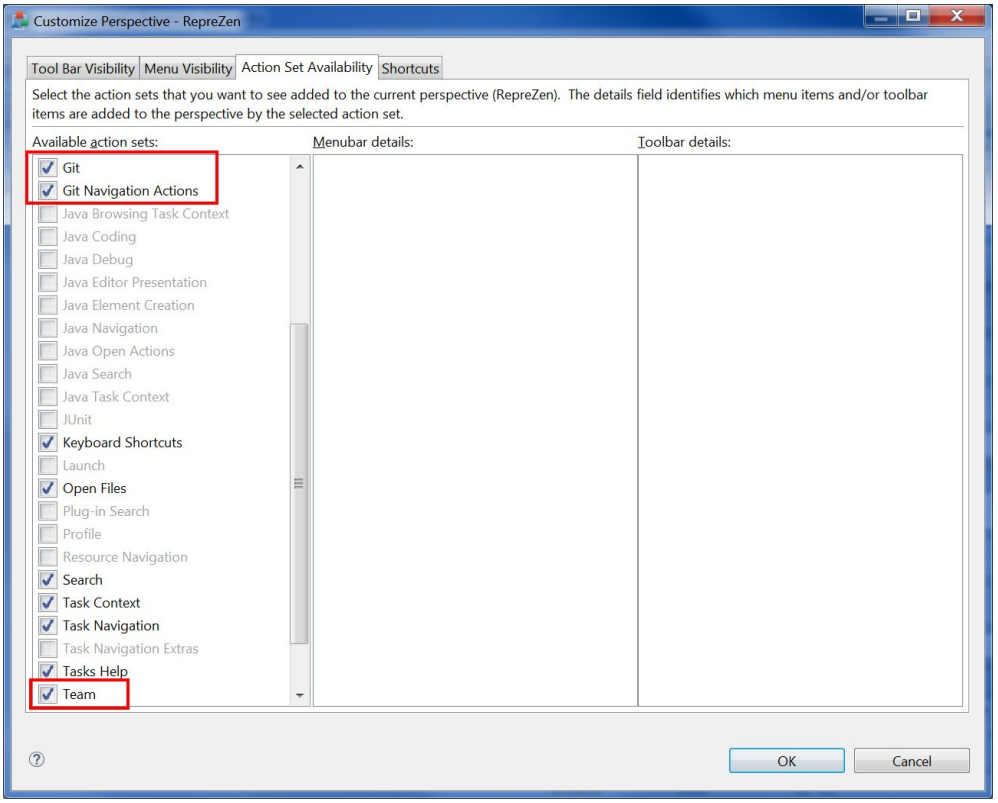
You may want to make some of the following changes, to make git easier to work with in API Studio.

Display the Git Toolbar

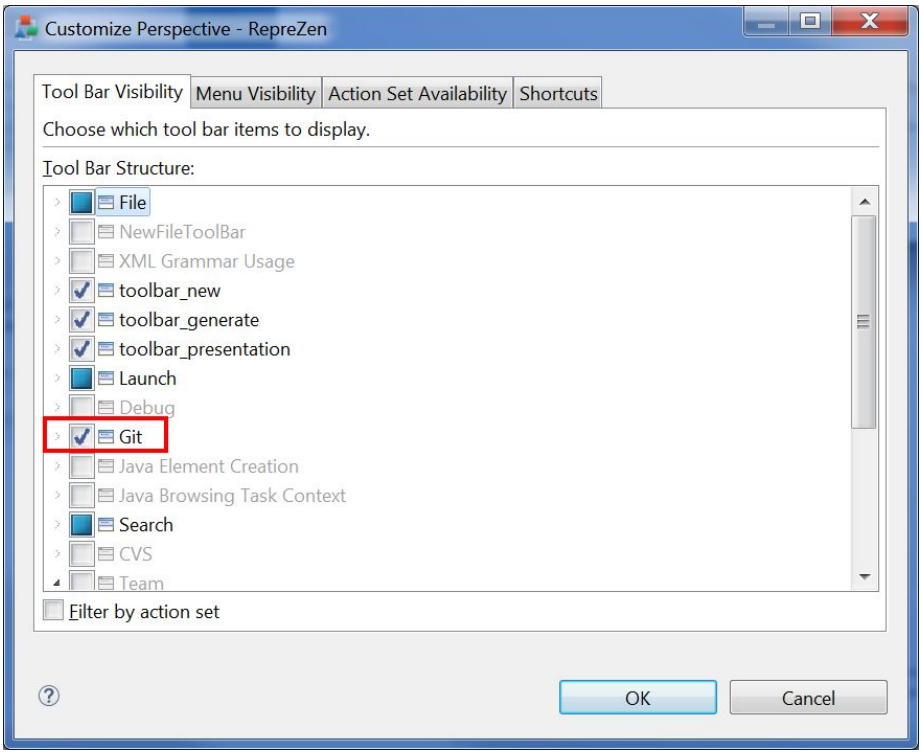
- Right-click the RepreZen Perspective button and choose Customize...



- Click the Action Set Availability Tab, and enable Git, Git Navigation Actions, and Team.

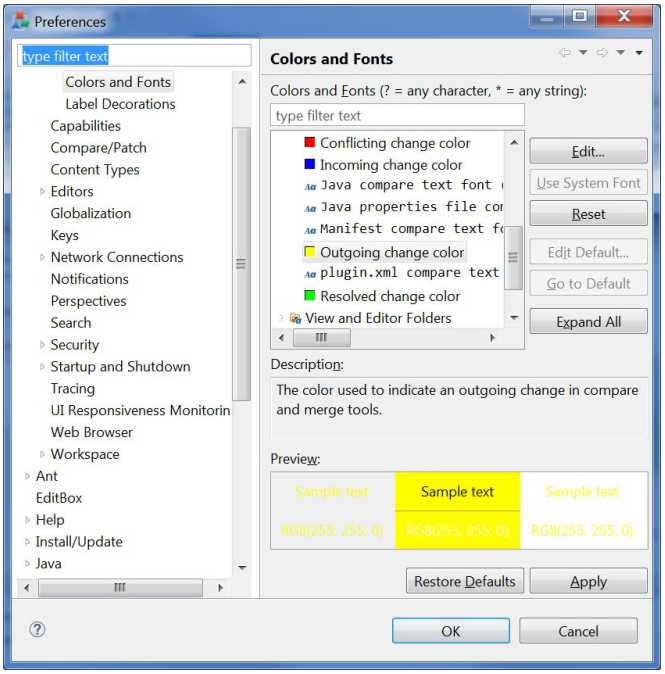


ð Click the Toolbar Visibility Tab, enable the Git toolbar, and click OK.



Set a contrasting background color for compare view

- Go to Preferences > General > Appearance > Colors and Fonts
- Select Outgoing Change Color
- Click Edit, and choose a color. RepreZen's default color scheme uses, red, blue and green for compare view, so it's best to choose a different color



Linking Projects to Git Repositories

RepreZen API Studio uses *qspkfdut* as the top-level containers to organize your work. In the Eclipse environment, a project corresponds to a physical folder on the filesystem. Files and subfolders within the RepreZen project usually mirror the physical layout of the corresponding filesystem folder.

Git also integrates with the filesystem in a similar fashion. In the filesystem, a folder is designated as a Git *sfqptjupsz*, and git adds some files within the hidden .git subfolder to manage manage the state of the repository, and its linkage to ashared remote repository, which is often hosted within a service like GitHub, BitBucket or VSTS.

In this section, we'll describe some options for organizing and linking RepreZen projects and Git repositories.

> Opuf;3EGit makes some specific recommendations for project organization, and certain UI elements reflect these recommendations. Not all of the scenarios described in this document align with those recommendations, and we leave it to you to decide what will work best in your environment.

Single-Project Repository

A Git repository may correspond to a single RepreZen project, one-to-one, in which case a single folder in the filesystem is both a local Git repository *boe* is the root folder for the RepreZen project.

The project folder may have been created as a git repository using the git init or git clone commands. In this case, RepreZen API Studio should automatically recognize that the project folder is within a Git project

(to be completed…)

Workspace Repository

The RepreZen Eclipse environment is managed under a single folder called a *xpslt qbdf*. Within the workspace folder, the .metadata subfolder stores the state of the workspace, including configuration settings and an index of included projects.

The project folders themselves do not have to be under the workspace folder. But they often are, as a matter of convenience and convention. The RepreZen and Eclipse dialogs will use the workspace folder as the default location for new projects.

A simple option for Git integration is to use the workspace folder as the Git repository, and maintain each project as a subfolder. The Git repository will contain all of the projects in the workspace.

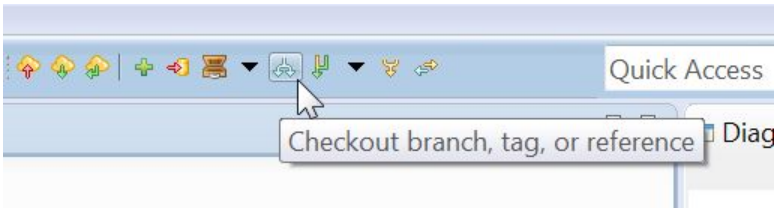
(to be completed…)

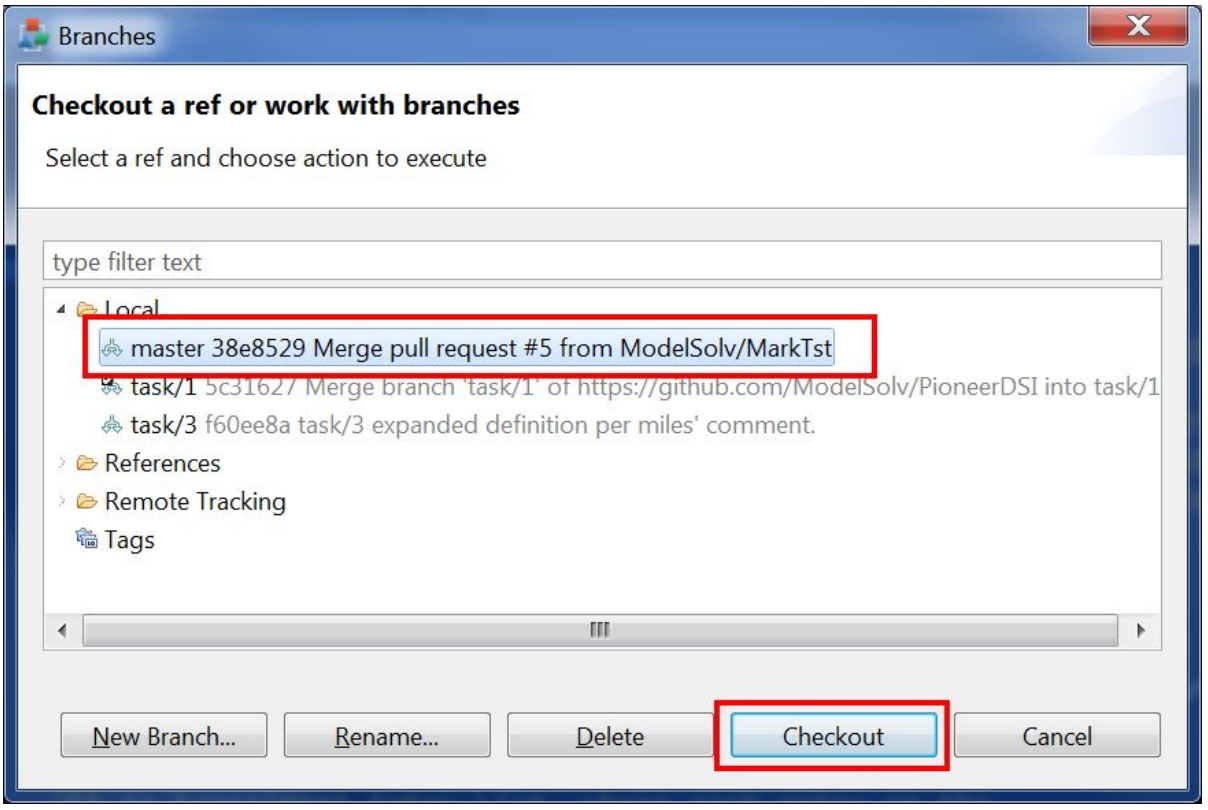
Parallel Repository

(to be completed…)

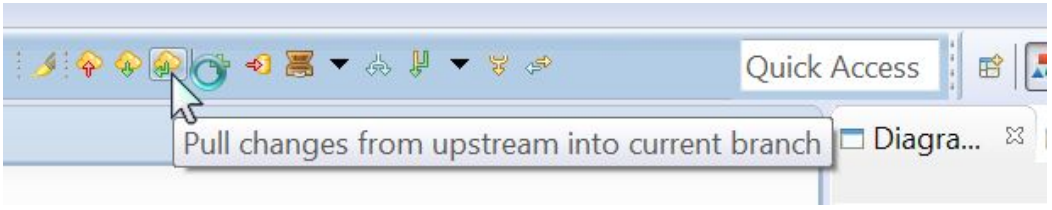
Pulling Changes from Master

- If you're on a branch other than master, use the Checkout Branch button to set your working directory to master:



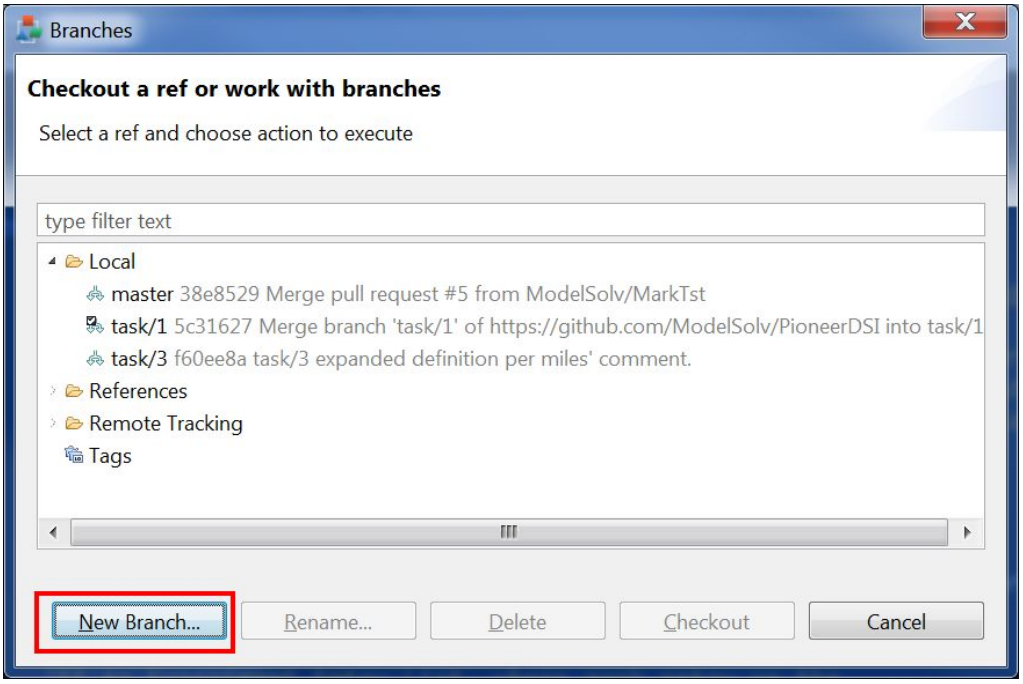


- Use the Pull button on the Git toolbar to synchronize from the origin repository to your current branch.



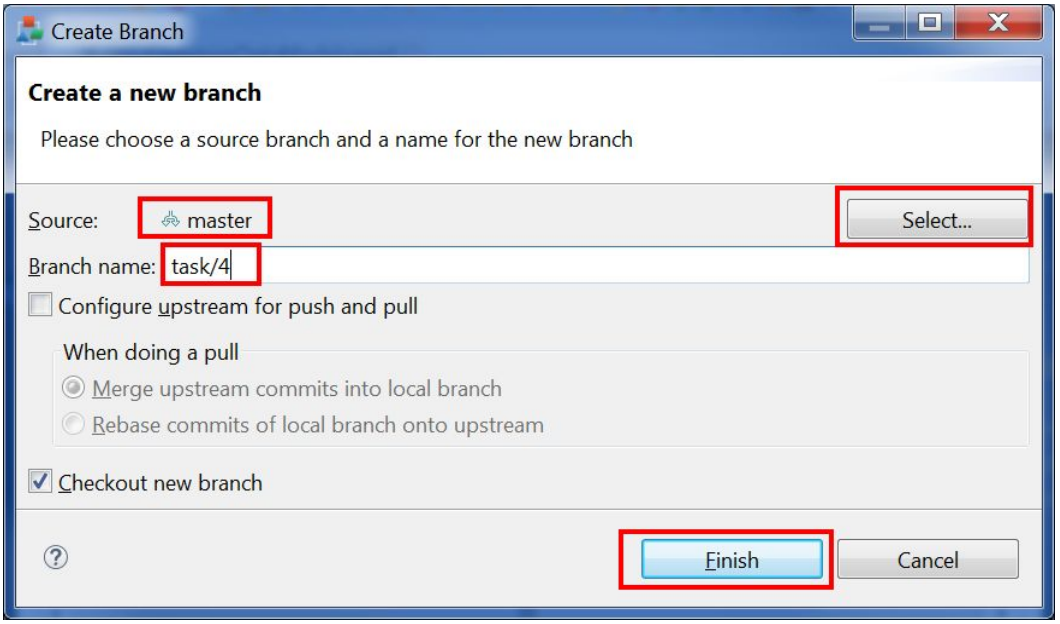
Creating a Task Branch

- Click the Checkout Branch button, and click New Branch…



- Click the Select button, and choose the local master branch as the source.

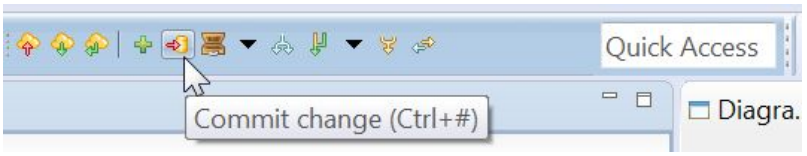
- Name the new branch, and click OK.



Committing and Pushing Changes

EGit provides a convenient Commit dialog that allows you to add changed files to the index, commit changes, and optionally push changes to the origin in a single step.

- ŏ Click the Commit button to show the Commit dialog:



- ŏ Select the files you want to include in the commit. By default, all changed files are selected.
- ŏ Provide a message for your commit.

- ŏ Click Commit to save the change locally, or Commit and Push (recommended) to push the commit to the remote origin repository after saving locally.