## Introduction

* Version control is the art of managing changes to information. It has long been a critical tool for programmers, who typically spend their time making small changes to software and then undoing or checking some of those changes the next day. Imagine a team of such developers working concurrently - and perhaps even simultaneously on the very same files! - and you can see why a good system is needed to manage the potential chaos.

## Git is an open source, distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Every Git clone is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Branching and merging are fast and easy to do.

## What is TortoiseGit?

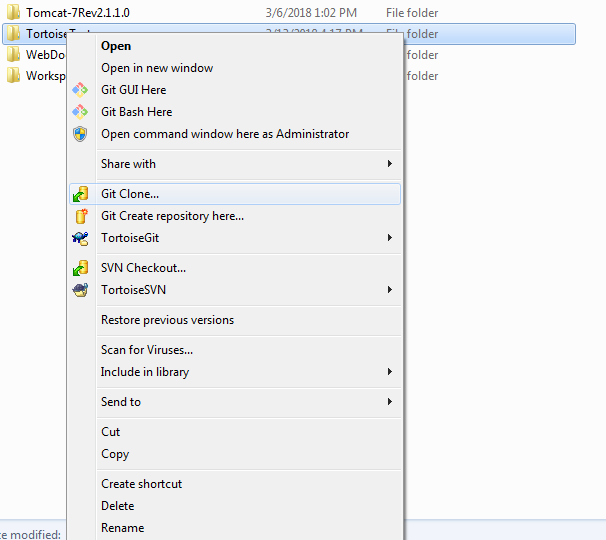
* TortoiseGit is a free open-source client for the Git version control system. That is, TortoiseGit manages files over time. Files are stored in a local repository. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your files and examine the history of how and when your data changed, and who changed it. This is why many people think of Git and version control systems in general as a sort of “time machine”.

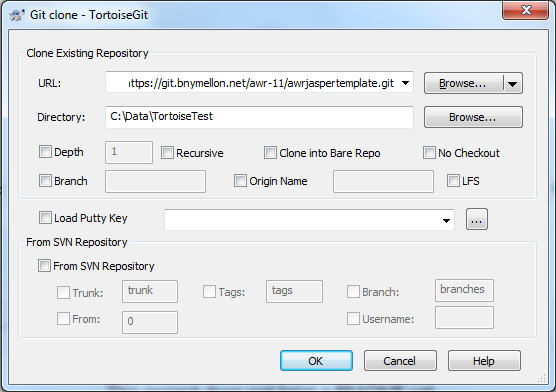
Clone Repository

* This section talks about how to clone a git repository from an existing repository. This operation is used to get a full copy of a remote repository. Cloning a git repository is very simple. At an empty directory, just use the explorer context menu and select **Git Clone...**.

The Clone Dialog will show.

**Figure  Git cloning**





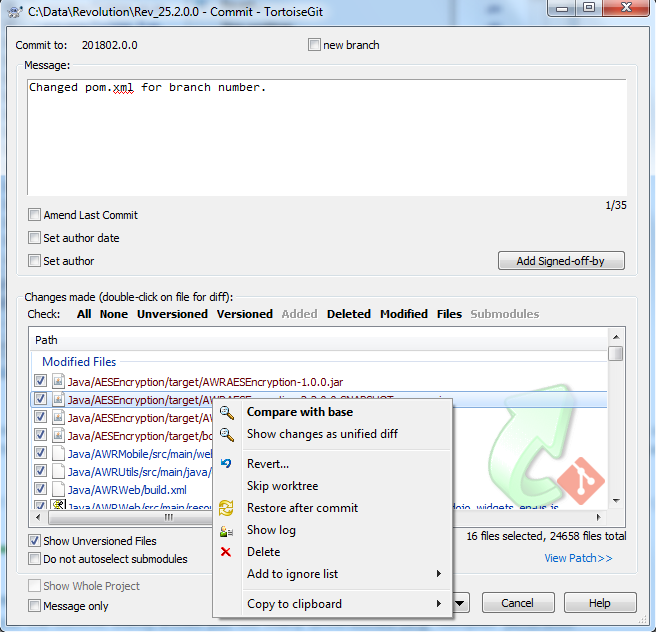
* URL: Input repository URL address, which you will clone from. You can click **Browse** to browse directory.
* Directory:Input your local directory, which you will clone to. You can click **Browse** to browse director.

## Committing Your Changes To The Repository

* Storing the changes you made to your working tree is known as committing the changes. you can use **TortoiseGit** → **Check for Modifications** first, to see which files have changed locally.

### **The Commit Dialog**

* If there are no conflicts, you are ready to commit your changes. Select any file and/or folders you want to commit, then **TortoiseGit** → **Commit...**.



* The commit dialog will show you every changed file, including added, deleted and unversioned files. If you don't want a changed file to be committed, just uncheck that file. If you want to include an unversioned file, just check that file to add it to the commit.

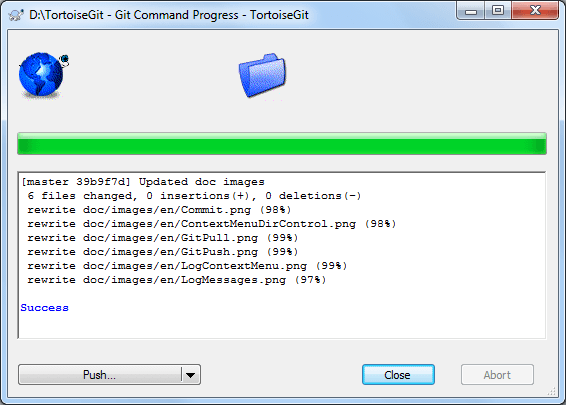
### **Commit Log Messages**

* Be sure to enter a log message which describes the changes you are committing. This will help you to see what happened and when, as you browse through the project log messages at a later date. The message can be as long or as brief as you like;

### **Commit Progress**

After pressing **Commit**, a dialog appears displaying the progress of the commit.

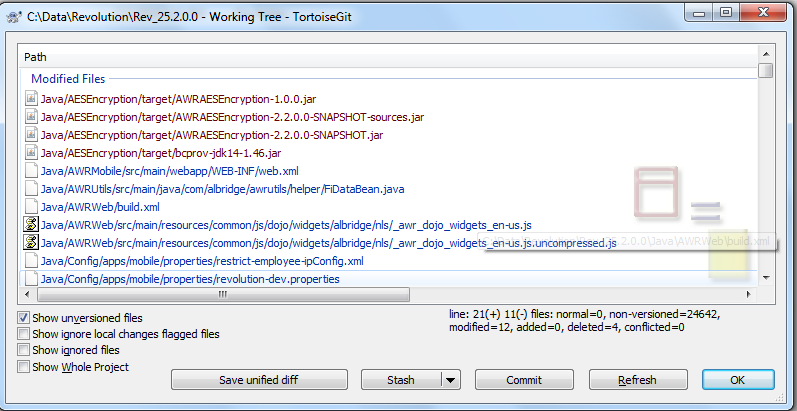
**Figure 2.11. The Progress dialog showing a commit in progress**



In the lower left, there is a menu button which provides shortcuts to further steps, such as **ReCommit** (resets the commit dialog and allows you to continue committing) or **Push** in order to push your commit to a remote repository

**Checking for modification**

* It's often very useful to know which files you have changed and also which files got changed and committed by others. That's where the command **TortoiseGit** → **Check For Modifications...** comes in handy. This dialog will show you every file that has changed in any way in your working tree, as well as any unversioned files you may have.



The dialog uses colour coding to highlight the status.

* Blue

Locally modified items.

* Purple

Added items. Items which have been added with history have a + sign in the Text status column, and a tooltip shows where the item was copied from.

* Dark red

Deleted or missing items.

* Green

Items modified locally and in the repository. The changes will be merged on update. These may produce conflicts on update.

* Bright red

Items modified locally and deleted in repository, or modified in repository and deleted locally. These will produce conflicts on update.

* Black

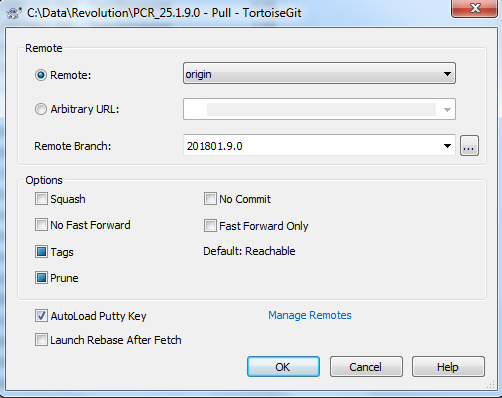
Unchanged and unversioned items.

## Pull and Fetch change

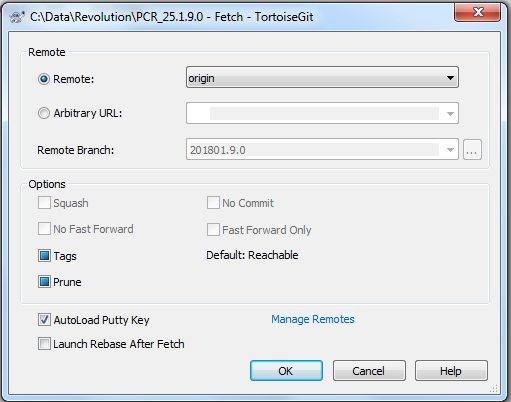
## Fetch just downloads the objects and refs from a remote repository and normally updates the remote tracking branches. Pull, however, will not only download the changes, but also merges them - it is the combination of fetch and merge (cf. [the section called “Merging”](https://tortoisegit.org/docs/tortoisegit/tgit-dug-merge.html)). The configured remote tracking branch is selected automatically.

* A pull/fetch can be initiated by using **TortoiseGit**→ **Pull...**or **TortoiseGit**→ **Fetch...**. Fetching and pulling changes is also possible using the Sync dialog (cf. [the section called “Sync”](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html)), however, there you have less options, but the sync dialog allows you to initiate other operations such as pushing and to see diffs and changes.
* A pull/fetch can be initiated by using **TortoiseGit**→ **Pull...**or **TortoiseGit**→ **Fetch...**

**Pull Dialog**

****

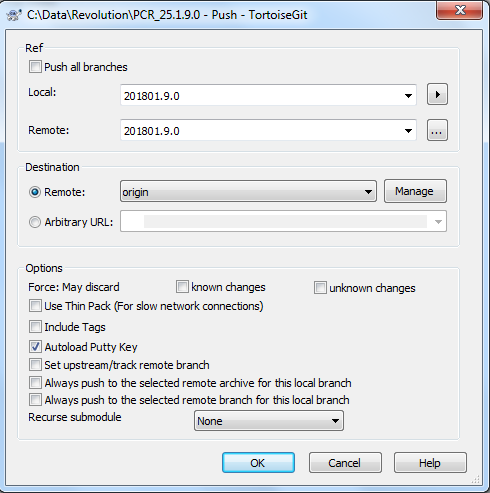
**Fetch Dialog**

****

* Remote Choose a configured remote repository (these can be changes using the Manage Remotes label). Instead of the configured repositories, you can also put the URL of another repository into the Arbitrary URL textbox. If the current active branch has a remote tracked branch set, the remote branch and remote repository are automatically selected
* Other: Input Other URL or local directory. You can click **...** to browse directory.

## Push

* In order to perform a push open the push dialog using **TortoiseGit**→ **Push...**

****

### **Branch**

* Local: The source branch which will be pushed to the other repository. If the current branch or the selected local branch has a remote tracked branch set, the remote branch and remote repository are automatically selected
* Remote: The remote branch of the other repository.

### **Destination**

* Remote: Choose an already configured remote repository.
* Arbitrary URL: The URL of a remote repository.

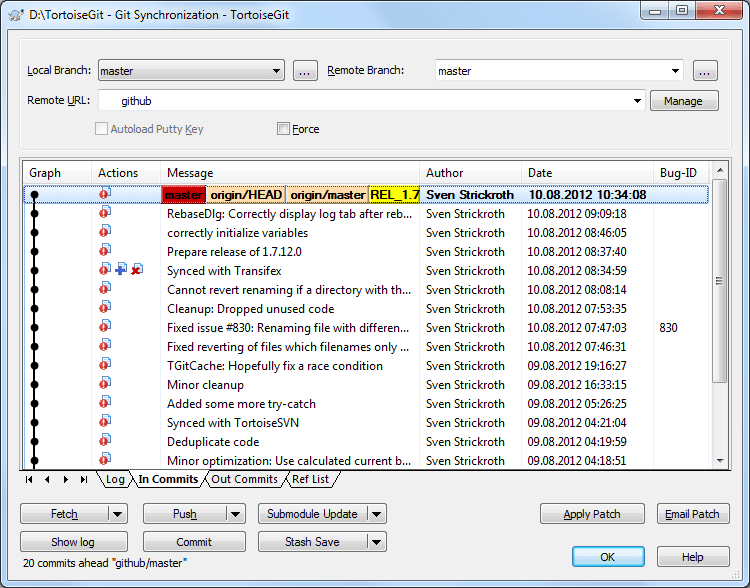
You must push change to a bare repository. Pushing changes to repository which has a working tree can lead to unexpected results.

* Force (May discard known changes) This allows remote repository to accept a safer non-fast-forward push. This can cause the remote repository to lose commits; use it with care. This can prevent from losing unknown changes from other people on the remote.
* Force (May discard unknown changes) This allows remote repository to accept an unsafe non-fast-forward push. This can cause the remote repository to lose commits; use it with care. This does not check any server commits, so it is possible to lose unknown changes on the remote.

## Sync

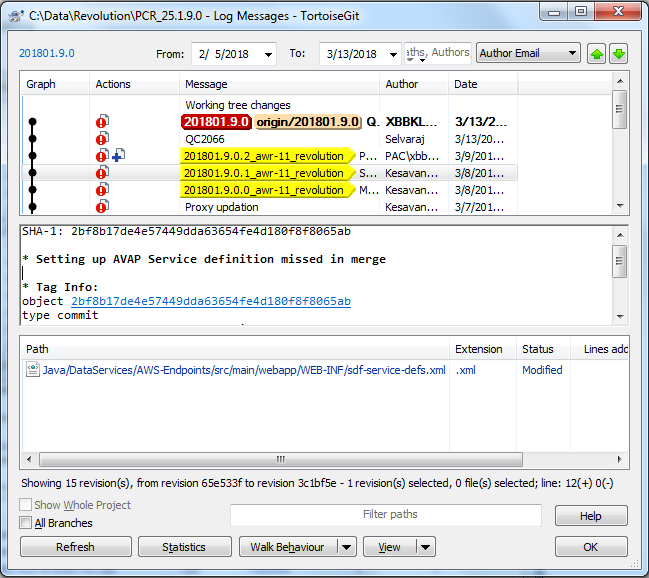
* The Sync Dialog provides an interface for all operations related with remote repositories in one dialog. This includes push, pull, fetch, remote update, submodule update, send patch... However, the sync dialog provides less options as the regarding dialogs (cf. [the section called “Pull and Fetch change”](https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html) and [the section called “Push”](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html)).
* The sync dialog can be opened using **Sync...**.

**Figure 2.17. Sync dialog**



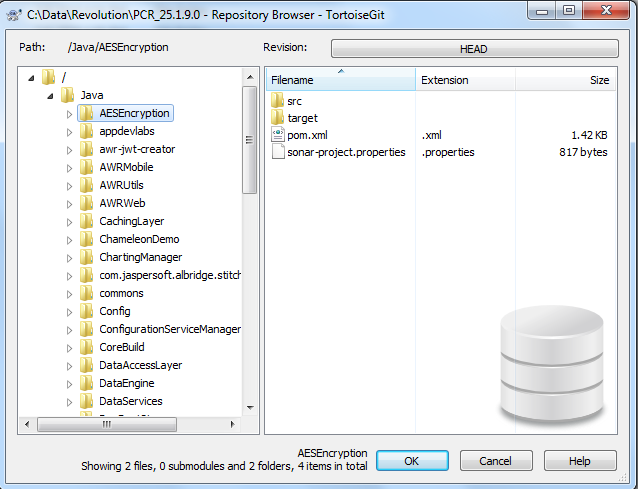
## Log Dialog

* For every change you make and commit, you should provide a log message for that change. That way you can later find out what changes you made and why, and you have a detailed log for your development process.
* The Log Dialog retrieves all those log messages and shows them to you. The display is divided into 3 panes.



## The Repository Browser

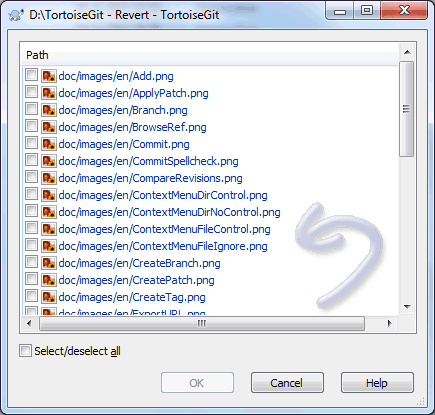
* Sometimes you need to see all contents/files of a repository, without having a working tree (e.g. a bare repository) or you want to see all files of a revision without switching to it. That's what the Repository Browser is for. You can open it using **TortoiseGit** → **Repo-browser**



## Undo Changes

* If you want to undo all changes you made in a file since your last commit you need to select the file, **right click** to pop up the context menu and then select the command **TortoiseGit** → **Revert** A dialog will pop up showing you the files that you've changed and can revert. Select those you want to revert and click on **OK**.

**Figure 2.44. Revert dialog**



* If you want to undo a deletion or a rename, you need to use Revert on the parent folder (or commit or repository status dialog) as the deleted item does not exist for you to right-click on.
* If you want to undo the addition of an item, this appears in the context menu as **TortoiseGit** → **Delete (keep local)**. This is really a revert as well, but the name has been changed to make it more obvious.