Git Source Control Provider for Stat

# Summary

Stat provides source control integration for Git, a popular distributed version control provider.

Git is free software, distributed under the GNU general public license. Stat integrates with Git by providing standard check-in-in/check-out functionality to a locally stored Git repository.

This document includes a brief discussion of Distributed Version Control Systems, Git installation instructions, instructions on creating a Git Repository, and how to configure and use Git in Stat.

# What is a Distributed Version Control System?

Git is a highly popular distributed version control system. There are some significant differences in the work flow of a distributed version control system (DVCS), compared to a conventional centralized version control system, such as Subversion (SVN) or Team Foundation Server. If you are new to using a decentralized version control system, it is important to understand these differences.

Centralized Systems

In a centralized system, the repository is a centrally located master copy, usually hosted on a server. Clients check out the master copy of an object to a local working folder to make changes. The object is locked for edits by other users. Once the modifications are complete, the changes are checked in to the central repository, and the object is unlocked. The workflow looks something like this:

* Retrieve the object from repository to local working folder.
* Lock object.
* Modify local copy.
* Check in local changes to repository
* Unlock object.

Distributed Systems

In a distributed system, such as Git, the system does not impose a central repository copy. Each user has a local version of the repository. Changes are committed to the local copy. There is no concept of locking an object. Local changes are shared with other developers by ‘pushing’ changes to a hosted remote repository, typically a hosted service such as GitHub. Remote changes are synchronized with the local repository by ‘pulling’ remote changes from the centralized repository. It is important to remember that in Git, the centralized repository in is arbitrary. A developer can clone a copy of any repository, and push/pull changes to the local repository. The distributed work flow might look something like this:

* Clone/Init - create or get a local copy of repository.
* Pull - Fetch merges/changes from remote repository.
* Edit - local copy of objects
* Stage - Add local changes to staging area. In Git perform ‘Add’ command.
* Commit local changes.
* Push – Push local changes/merges to remote repository.

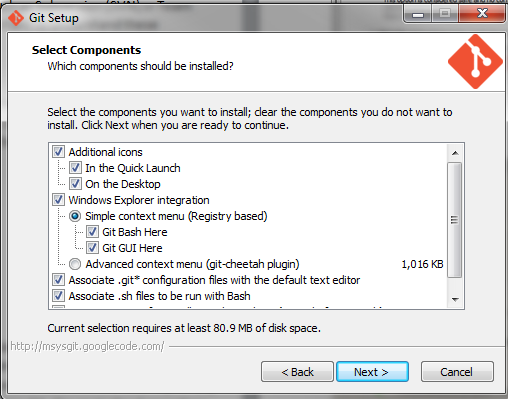
The work flow differences between these two models have significant implications for Toad. Because there is no system defined master copy, Toad checks in / out objects from the local repository, as specified by the user. There may be changes made by other users that Toad is not aware of. Updates and merges with a remote repository are handled outside of Toad. Toad also commits local changes immediately, which is similar to the work flow for other source control providers in Toad.

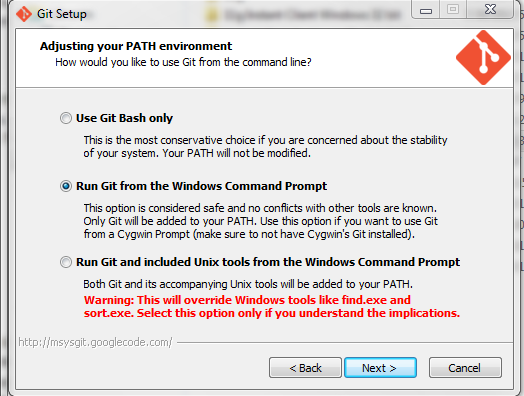
The most important thing to remember is that Toad checks in/out files from the local repository. Changes are posted back to the local repository immediately. Updates to any centralized repository are handled outside of Toad.

# Prerequisites

1. Install the Git Window Client

If it is not already installed, install Git Windows Client. You can download a windows install client from <http://msysgit.github.io/>. Run the installer. Please note the options in the following screens. Make sure to enable the ‘Run from Windows Command Prompt’ option.





1. Set Up an empty Git Repository

If you have an existing Git repository you may use it with Toad. In a typical environment, a client ‘clones’ and existing Git repository. For the purposes of this walkthrough, we will create a new empty Git repository to use with Toad. You can do this from the windows command line, or using the Git windows client. We will use the command line. Git is case sensitive. All commands are in lower case. Also note that Git converts the Windows back slash ‘\’ to forward slash ‘/’.

* 1. Open the windows command prompt from the Windows start menu.
  2. From the command prompt create a new directory by typing the following command:
     + 1. mkdir c:\Work\GitMaster
  3. Navigate to the directory you just created:
     + 1. cd \Work\GitMaster
  4. Create the new Git Repository by typing:

git init

Window displays ‘Initialized empty git repository in c:/Work/GitMaster/.git’ message

1. Populate Git Repository

Before using the repository we need to add some objects to control. Toad treats subdirectories under the Git main directory as modules or projects. Each project you use in Toad has a corresponding directory in the Git repository. For example, to have a TKD\_SCHOOL project, create a directory with that name in the Git master directory. In order to see the directory in Toad and Git, an initial file must be added to Git.

* 1. Create an empty TKD\_SCHOOL directory in the GitMaster directory

mkdir TKD\_SCHOOL

* 1. Edit or add a file to this directory, let’s say Initfile.txt. We created one using notepad.exe TKD\_SCHOOL\initfile.txt, but you can use any method you prefer.
  2. Add the file to git. This is a two-step process.
     1. ‘git add’ adds changes for staging. From the GitMaster directory type:

git add TKD\_SCHOOL/InitFile.txt.

* + 1. ‘git commit’ commits staged changes to the local repository. Type:

git commit –m “Initial commit to create directory” TKD\_SCHOOL/InitFile.txt

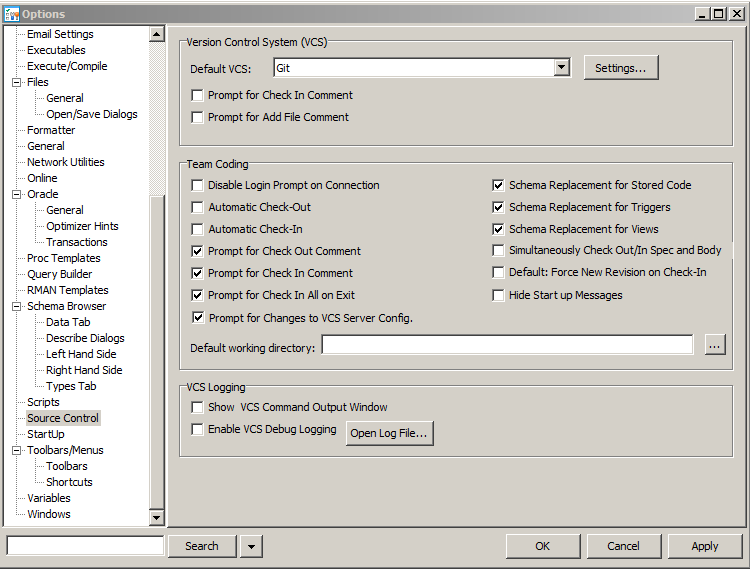
The Git repository is ready, with one sub-folder. You can add multiple sub-folders if you wish, by repeating the steps above.

# Using Toad Source Control with Git

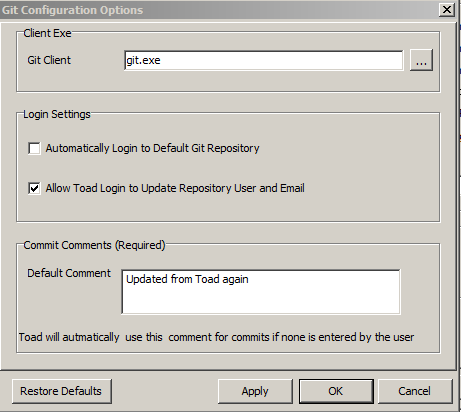
**Configure Client Settings**

To set up Toad Source Control to work with the Git client perform the following steps:

1. Open Toad Options – Source Control. Select ‘Git’ from the Default VCS dropdown.



1. Open ‘Settings…’ to configure the Git client. The following settings are available.

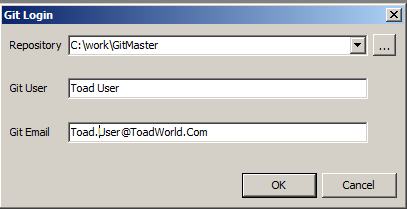


* 1. **Git Client:** Windows should point to the default Git client. Typically, no change is required from the default. If you wish to point to a different Git executable you may enter the full path to the client executable.
  2. **Automatically Login to Default Git Repository:** Check this option if you wish to login to the default repository without opening the login window.
  3. **Allow Toad Login to Update Repository User and Email:** If this option is checked, Toad will update the Git user and email configuration settings for the repository. These are used to track the author of changes and commits.
  4. **Commit Comments:**  Git requires comments for commit. The Default Comment value will be used for commits, if no comment is set by the user.

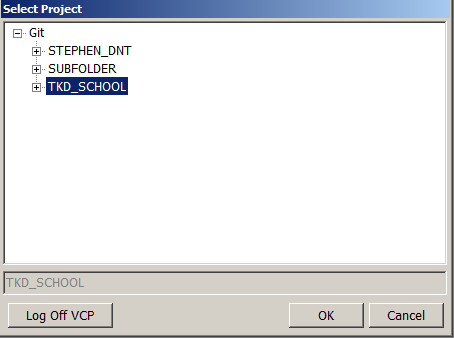
**Log in to Git from Toad Source Control**

To use Toad source control with Git, log in to source control.

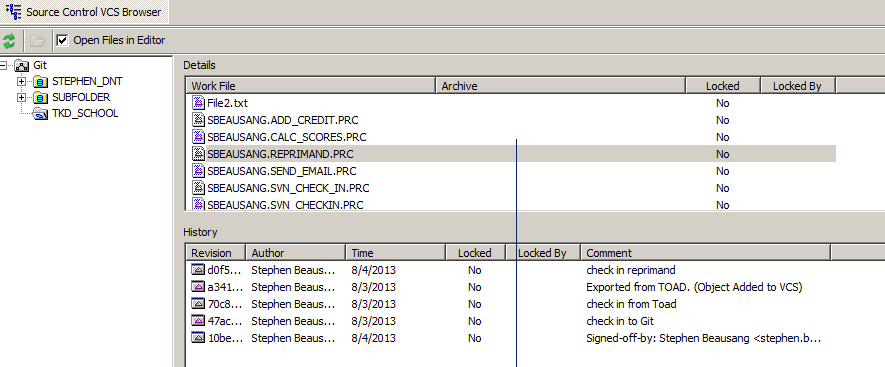
1. From the Toad main menu select Utilities – Team Coding – Select VCS Project. The Git login window will open. Enter the information for repository, Git User and Git email. Click Ok.



1. The source control select project dialog opens. You can now select the Git Sub Directory TKD\_SCHOOL from the list of modules.



1. Open the source control VCS Browser. You can browse Git controlled objects, open files in the editor, check in, check out or add files to the local Git repository.



WRH Additions/Changes:

Git for Windows is the Git Repository VIEWER/APPROVER – Not Source Code Add/remove.

To Add/Remove Files/Source to the GIT Repository - Add & Remove is used through the GIT Shell Client:

**O:\GIT\_HUB>git help**

usage: git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p|--paginate|--no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

The most commonly used git commands are:

add Add file contents to the index

bisect Find by binary search the change that introduced a bug

branch List, create, or delete branches

checkout Checkout a branch or paths to the working tree

clone Clone a repository into a new directory

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

fetch Download objects and refs from another repository

grep Print lines matching a pattern

init Create an empty Git repository or reinitialize an existing one

log Show commit logs

merge Join two or more development histories together

mv Move or rename a file, a directory, or a symlink

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

rebase Forward-port local commits to the updated upstream head

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

show Show various types of objects

status Show the working tree status

tag Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

Sample:

Add File to GitHub:

Using GitShell:

1. Move a copy of the file into the appropriate directory:
2. Issue a Git Add command: O:\GIT\_HUB\StatCode>**git add GeekGuy.jpg**
3. **Go to the Git Windows Client – Select the “Sync” button. This will query the local directories and pick up any new files recently added since the last synch.**