**Version Control Systems Basics, Git Basics Tutorial and Git Repositories @ NST**

## Version Control Systems Basics

A version control system is a software tool that allows to record changes brought in time to a file or a set of files in order to be able to recall specific versions later.

Using this system one can:

- revert files / sets of files to a previous state
- review the changes made over time
- working together and minimizing edit conflicts
- get little overhead by storing differences instead of full copies of files
- backup files automatically

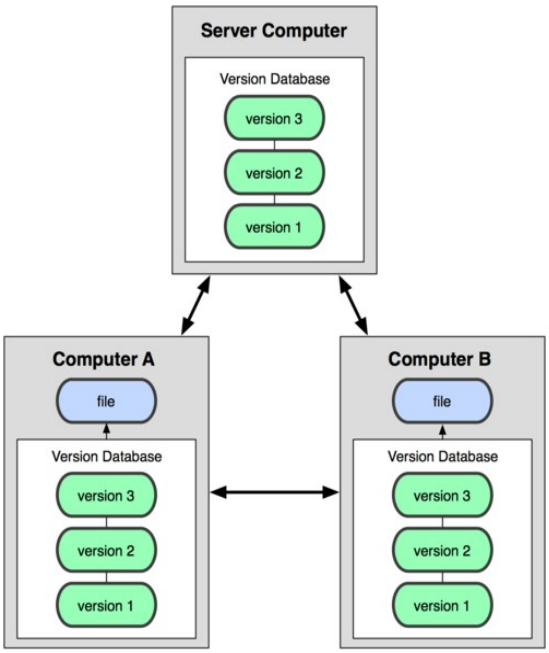
The core of the Version Control System is the repository. It refers to the data structure stored on the server which contains a set of files (grouped or not in directories), a history log of changes made in the files, a set of version control system files for data handling and management.

There are 3 types of version control systems: Local Version Control Systems, Centralized Version Control Systems and Distributed Version Control Systems. We will use and focus only on the latter type, as a tool for maintaining our projects at NST.

Distributed Version Control Systems allow:

- collaborative usage between developers
- clients get a full mirrored image of the files repository (every checkout is a full backup)
- supports complex paths of collaboration between clients with multiple repositories
- fault tolerant to server problems (easy to restore)

The generic architecture of a Distributed Version Control System:



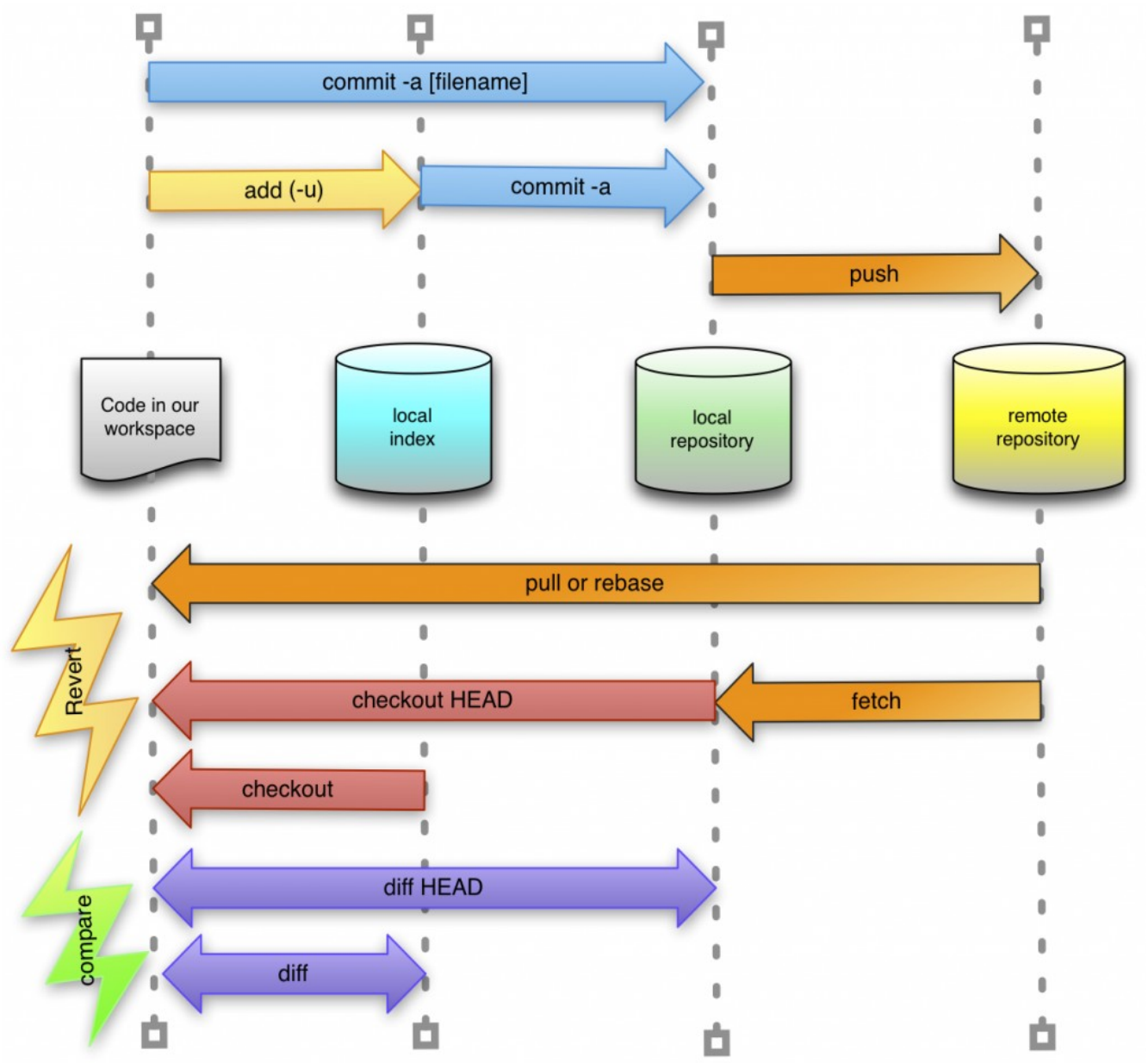
**Git Basics**

Git is a fully distributed Version Control System, widely used because of its speed, efficiency in handling large projects and the powerful branching mechanism which ensures a good management of many parallel branches.

Git is different from the other Version Control Systems in terms of:

- data handling (save snapshots, not differences)
- locality (no network latency in getting the entire change history)
- data integrity (check-summing based on the content of the file / files)
- handling errors (hard to loose changes, recovery is easy)
- clear workflow based on states

Git workflow diagram:



Understanding the workflow:

Assuming you have a repository (empty or not) on a remote server accessible via Internet or a local network. There are three possible scenarios.

If the remote repository is empty (not populated with files yet) you have to add the files you want in your project from your workspace to the local index (keeps a record of repository files, also called stage) and then commit them to the local repository. The local repository has the same structure as the remote repository. When everything is commited in the local repository you simply push the files into the remote repository.
Right now the remote repository has the same contents as you local repository.

If the remote repository is populated with files the first step is to get a snapshot of your remote repository on your local machine, so that you will get a working copy on your machine. This operation can be performed by cloning the remote repository. If you had already a local copy, in order to get the latest snapshot of the remote repository you just need to fetch it.

Assuming that you added some changes in the local repository files and you want to make these changes recorded in the remote repository, as to be available for others or as a major change to be recorded the following procedure must be performed.

First, add the modified files to the local index (stage), commit them to the local repository and then push them into the remote repository. If there is something that you are not sure that it should be pushed to the remote repository but it is commited in your local repository, you can checkout the specific commit (changeset) and rework it, by reverting the commit. The commits are stored in a stack like structure in the repository (local/remote) and you can access the latest change reference (HEAD) or any of the earlier entries. The entries are saved with a timestamp for sequencing them time and with a hash (SHA-1) for their internal storage in Git. After modifying the files the procedure repeats itself, by adding and commiting the changes to the local repository and then if needed to the remote repository.

In order to visualize the changes between the local repository version and the latest changes added to the files in the workspace we can use a difference viewer that can display differences to the local repository (commited changes) or to the local index (modified but not commited to the repository yet ).

**Small guide of useful Git commands**

The following commands are usually entered in the command line, but as there are visual tools that allow not only to visualize but also to operate on the repository, these commands are present also in the GUI clients. Sometimes, if you are coding in a IDE there are plug-ins or add-ons to integrate the use of Git in your development environment.

**git help** *command*

- Returns the information about *command.*

**git clone**

- Gets a snapshot of the remote repository on the local machine.

**git fetch**

- Gets the latest changes from the remote repository when a cloned snapshot is already on the local machine.

**git rebase**

- Merges you local commits with commits from the remote repository by first applying remote commits and then applying your commits on top.
- During a rebase you may encounter conflicts which have to be resolved manually.

**git log**

- Returns the stack-like formatted list of commits in the local repository. This list is synchronized with the remote repository when pushing / pulling / cloning. The information returned by this logging command contains the unique hash key (SHA-1) representing each commit, the author and date of the commit and also the commit message.
-

**git status**

- Returns the status of the files (the content) of the current working directory. Usually git status gives also hints on how to change the status of the files in the working directory to be able to add / commit / push the changes properly. Some examples of repository file statuses: not updated, updated in index, unmerged, untracked, ignored.

**git diff**

- Check the difference between two commits or the last commited version (HEAD) and the current version in the workspace.

**git add**

- Adds the modified file / files to the local index (stage) so that it / they can be commited to the local repository.

**git rm**

- Removes a file / files from the local index (stage) so that it / they won't be present anymore in the repository. If commit is called after removing files the local repository won't contain the removed file / files anymore. The change can be propagated to the remote repository by pushing the local repository.

**git commit**

- Copies the snapshot of the staged files from the local index into the repository and updates the existing references.

**git push**

- Pushes the changes from your local repository to the remote repository.

**git checkout**

- Copies files from the local index (stage) to the working directory, and can be used for switch branches in case that the repository has multiple branches.

**git reset**

- It is used to copy files from the local index without touching the working directory. It can also move the current branch to another position, and optionally updates the stage and the working directory.

## Git Tools

### GNU/Linux

Command line:
http://git-scm.com/download/linux
or from your package manager (e.g.: yum – RedHat based OS, apt – Debian based OS)

GUI Clients:
Gitk, git-gui

### MS Windows

Command line:
http://git-scm.com/download/win

Cygwin (http://www.cygwin.com/)

GUI Clients:
TortoiseGIT (http://code.google.com/p/tortoisegit/)
MsysGit (http://msysgit.github.com/)
Sourcetree (http://sourcetreeapp.com/)

### Mac OS

Command line:
git binary (http://git-scm.com/download/mac)

GUI clients:
GitX( http://gitx.laullon.com/)
Gitbox (http://gitboxapp.com/)
Gittiapp (http://www.gittiapp.com/)

## Git Repository Servers

The easiest way to get access to a git repository are public services like bitbucket or github.

Bitbucket https://bitbucket.org/
You can have unlimited private repositories. You are encouraged to use this site for non-public projects.

Github https://github.com/
Unlimited public repositories but no private repositories.

To use theses services you have to create an account on the respective websites.

**Git Repositories @ NST**

We encourage every running project in NST to be maintained using Git as the main tool for version control.

Guidelines:

- in order to avoid problems with accessing the repository from remote locations we decided to allow the usage of Bitbucket, https://bitbucket.org instead of using a local TUM network machine to store the repositories;
- this will allow you to easily create unlimited private repositories for your projects and **it's important to grant admin (read/write) access to your project supervisor because a snapshot of the repository will be mirrored on our local NST machines;**
- in order to keep the repositories clean, small and fast to access we recommend not to upload other file types than source code (e.g.: binaries, pictures, movies, large pdfs, archives);

In the following some short tutorial on basic git use cases will be given.

## Typical usage scenarios

### 1. Linux / MacOS / Cygwin on Windows command line Git tutorial

#### Getting the repository

You have created a repository with on of the repository providers (bitbucket, github, …) and what to start working with it.

# clone the remote repository using different protocols
# the following URL is an example for a bitbucket repository URL
# you get the URL from your repository provider
**git clone git@bitbucket.org:USERNAME/PROJECT.git**

See here for details:
http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository

#### Adding a new file to the repository

You want to create a new file and add it to the repository.

# create a file in the repository and put some text in it
# the following line is for linux command line – you can also use a normal text editor
**echo "git test string" > test.txt**

# tell git to add the file
**git add test.txt**

# commit the file (this is only locally – it does not change the remote repository)
# With -m a commit message can be given, in this case it is "a test commit"
**git commit -m "a test commit"**

# send the commit to the remote repository
# origin is the default name of the repository which you used when using clone
# master is the default name for the branch you are working on
**git push origin master**

See here for details:
http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository

#### Modifying a file and sending changes to the repository

You want to modify new file and add the changes to the repository.

# we modify the file we created previously
# the following line is for linux command line – you can also use a normal text editor
**echo "git test string with modifications" > test.txt**

# lets have a look on our changes
**git status**
# you should see a message “  modified: test.txt”

# lets have a more detailed look on our changes
**git diff test.txt**
# you should see changes made to the file. Leave by pressing q

# add the changes, commit them and push to the server
**git add test.txt**
# if you want to remove a file
**git rm unuseful_file.c**

# Note that add is used for adding files to the staging area. It does not matter if the file is new or modified. All files in the staging area will be added to a commit.
**git commit -m “modified the test file”**
**git push origin master**

See here for details:
http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository

**Getting commits from a remote repository**

In the mean time someone else may have added commits to the remote repository. You want to get changes from someone else to your local repository.

# the repository is on the local machine, let's see the history
**git log**

# first get the changes
# this does not change your working copy yet – it only fetches remote changes
**git fetch origin master**

# now we apply the changes to our local copy
# this command
**git rebase origin/master**

See here for details:
http://git-scm.com/book/en/Git-Branching-Rebasing

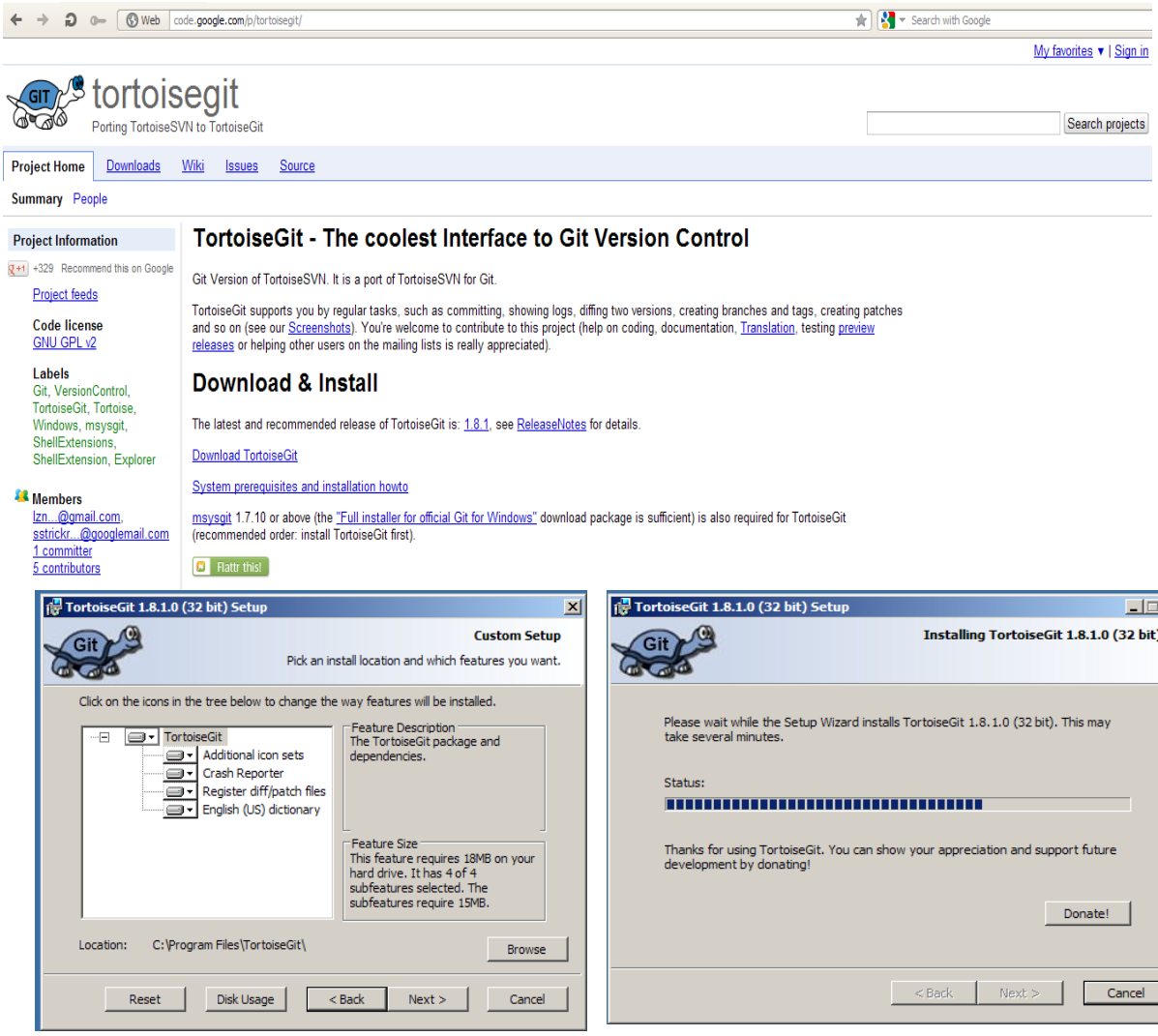
Most times rebase is able to handle conflicts automatically. But if you and someone else modified the same file, you may have to resolve changes.

## 2. Typical usage scenario on MS Windows using TortoiseGIT & msysGIT

### Install TortoiseGIT

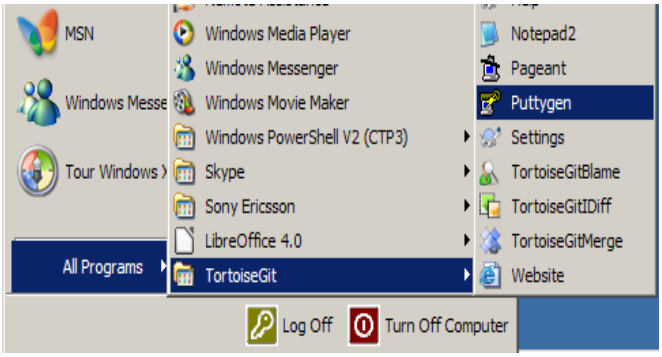
Go to: http://code.google.com/p/tortoisegit/ and download the latest version

Check the system prerequisites as for old version of MS Windows XP you might need a newer Windows Installer version; support is given to guide you on getting the correct version.

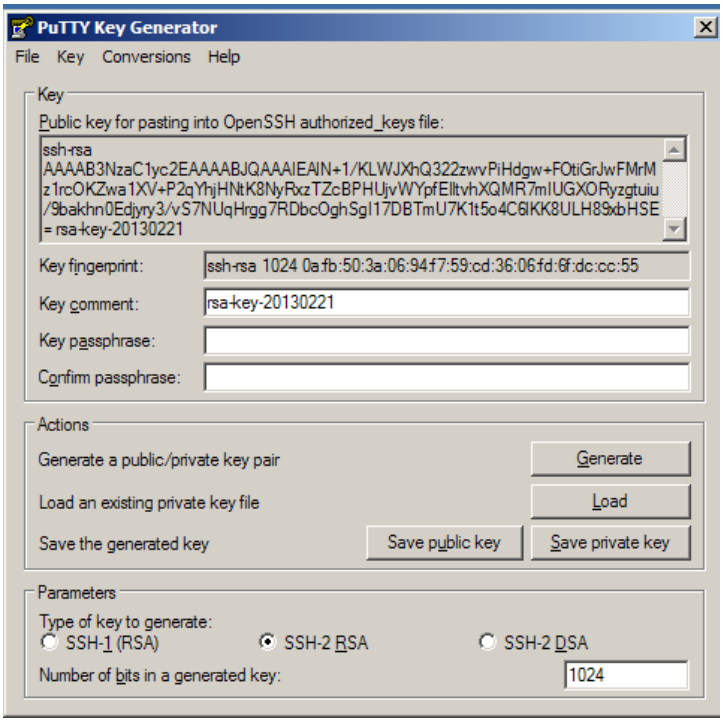



### Generate the SSH key to authorize the read and write access to the repository

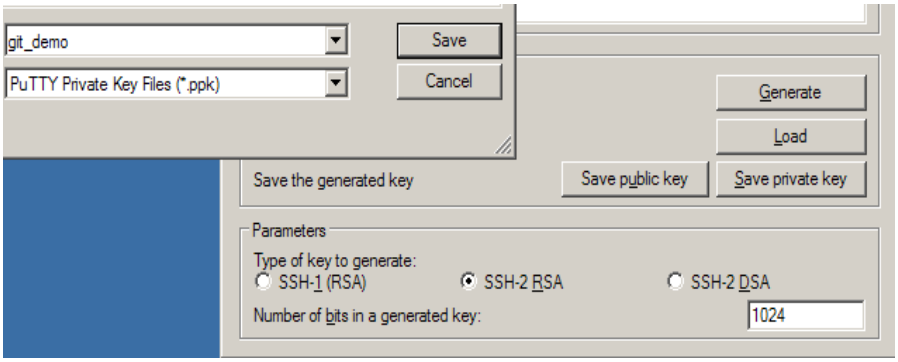
The SSH (SecureSHell) key is a hashed representation which encodes your machines identity to help authorizing the accesses to the repository (e.g. pull, push). In order to generate your machine's key you can use the PuttyGen tool provided by TortoiseGIT.



The first step is to generate the SSH key, by pressing generate in the PuttyGen window. As a default you can use the SSH-2 RSA, secure shell key secured with Rivest-Shamir-Adleman public-key cryptography algorithm.



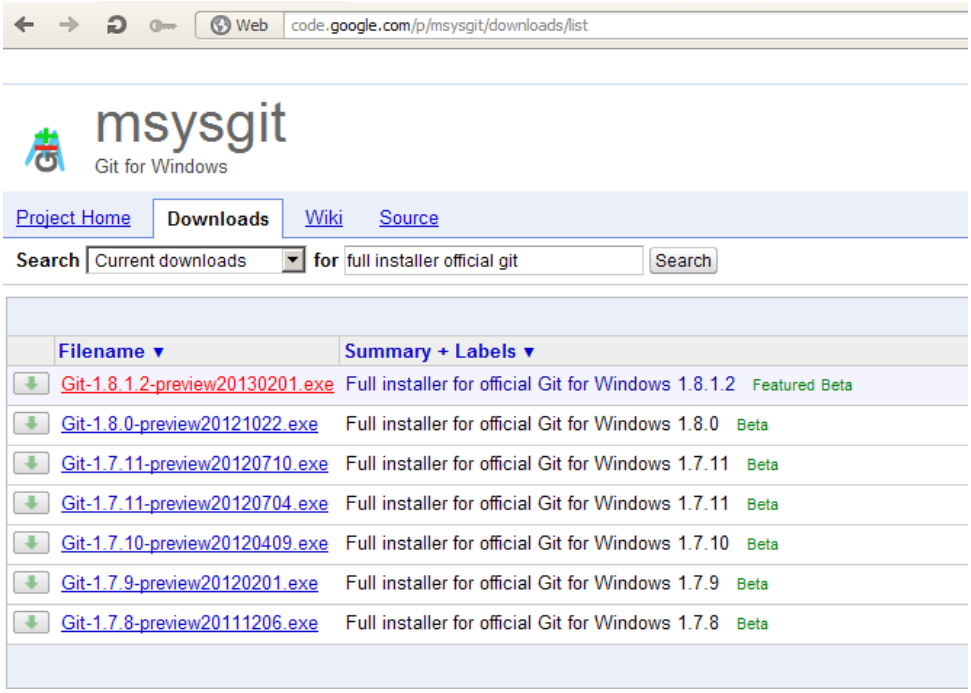

Afterwards you need to save the generated key on the disk, so that later you can use it to set up access to your repository. Note that in order to save the keys you need to define also a Key passphrase that will be also added to the generated key. Afterwards you can similarly save the public key, preferable in the same location as the private one, with a different name from the private one (i.e. adding .pub extension to the private key).



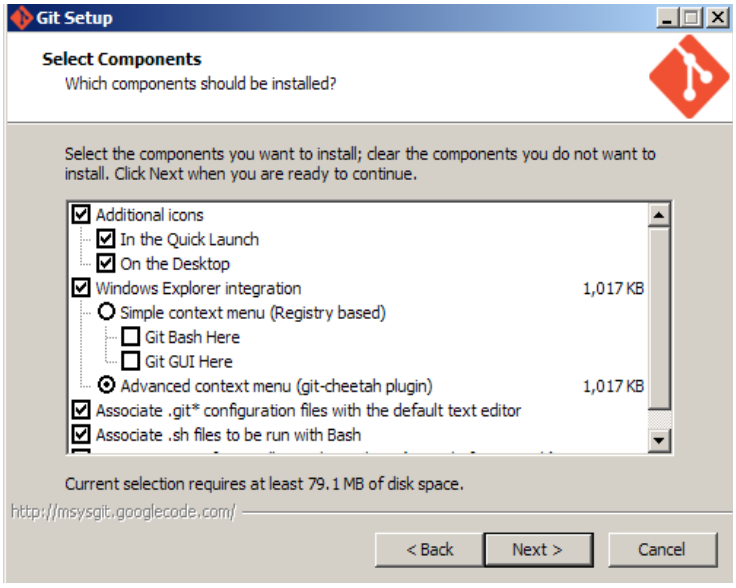

Next step is to install the git tools for Windows contained in the msysGIT.

**Install msysGIT**

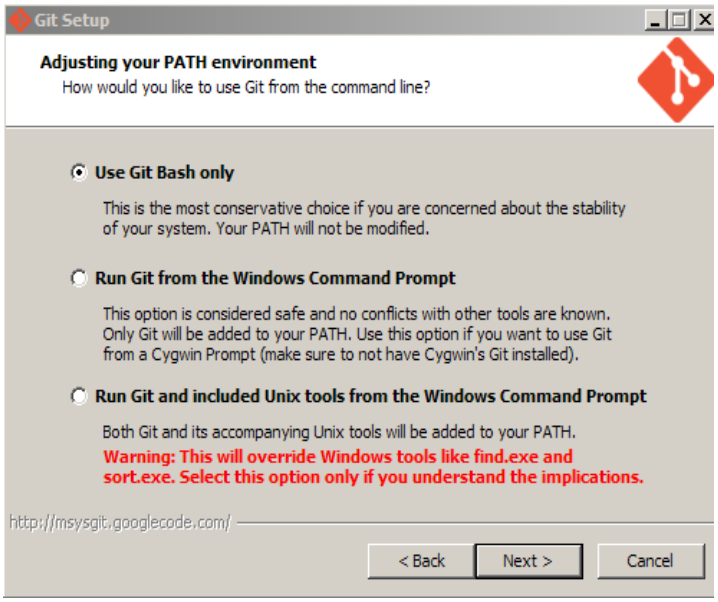
Go to [http://code.google.com/p/msysgit/downloads/list](http://code.google.com/p/msysgit/downloads/list) and download the latest version



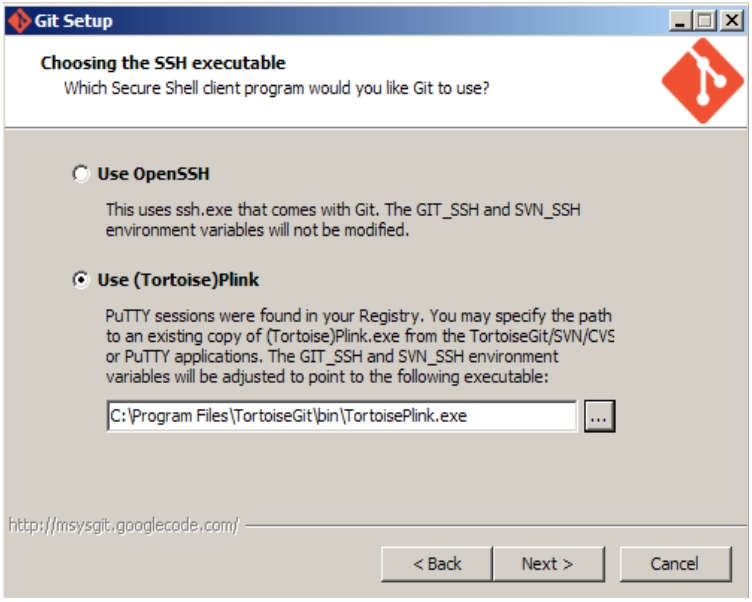

Activate both command line and GUI utilities in the installation process (e.g. done with simple context menu), but for beginners it's recommended to choose the Advanced context menu. The git commands will be integrated in the MS Windows right-click menus so that you will be able to issue the git commands directly from the Explorer without a command line.



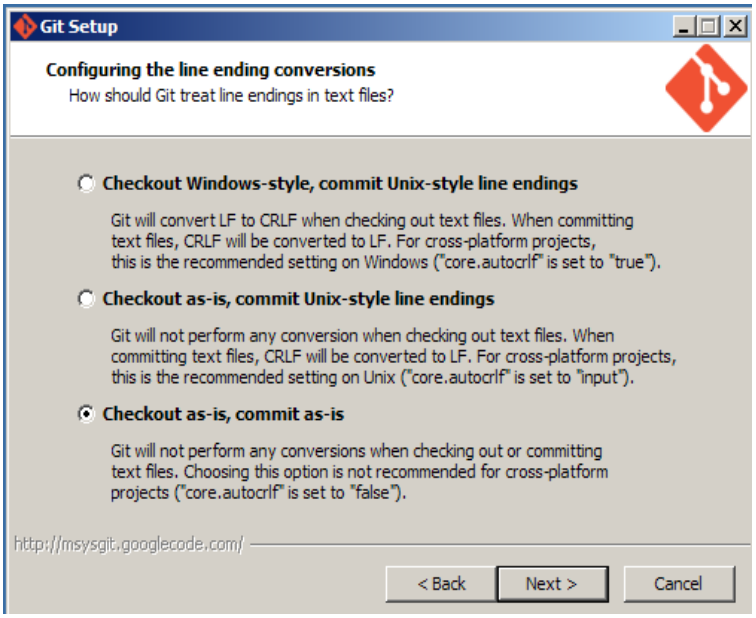

For normal users it is recommended to go with the default settings when choosing the background executables used with / in the git operation process.




The active authentication will be performed by a ssh application which will use the generated keys to perform the authentication to push changes in the remote repository. In this case you will use the ssh application provided by TortoiseGit, namely the Plink.



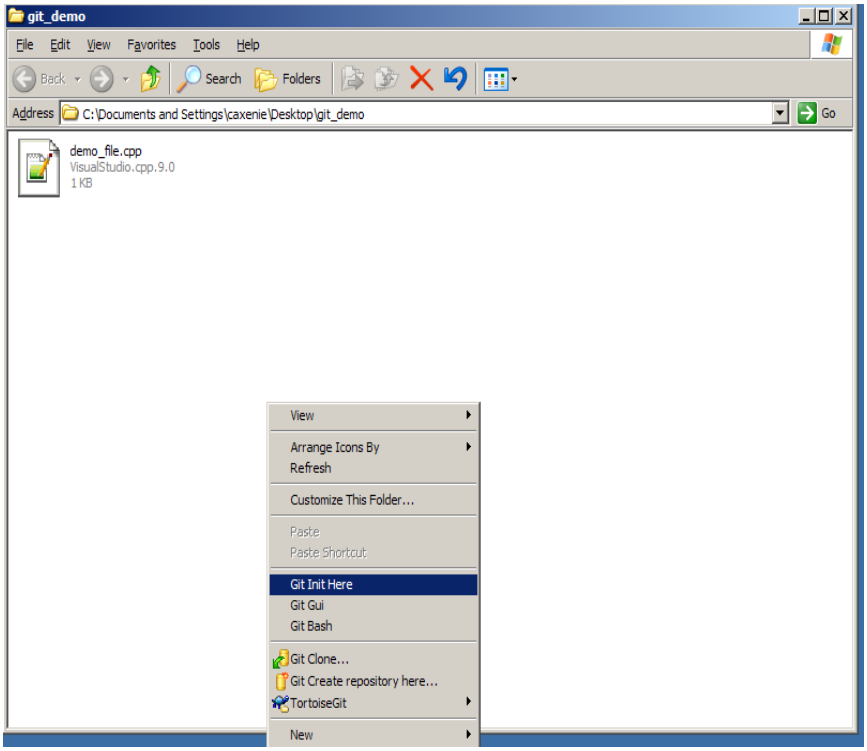

As each OS has it's different formatting scheme for ASCII encoded files, you will need to specify a preferred format to be respected by git when pushing / pulling files from the repository so that a consistent format is maintained across changesets.




## Sample usage scenario usage on MS Windows

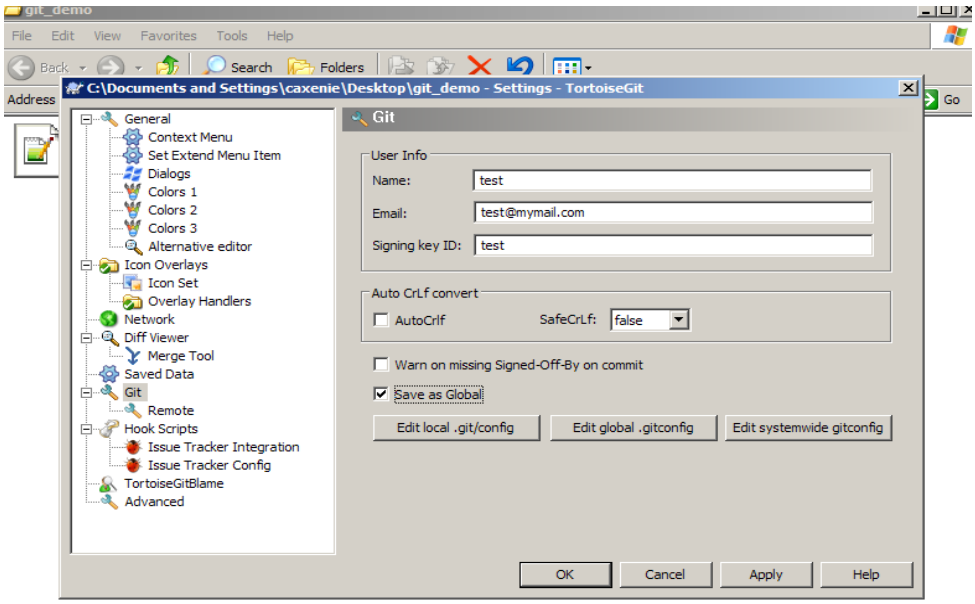
### Initialize the git repository

Assuming that you have a source code directory from which you would like to create a repository using git. The first is to initialize the git repository to that location on the disk, using git init command integrated in the menu.



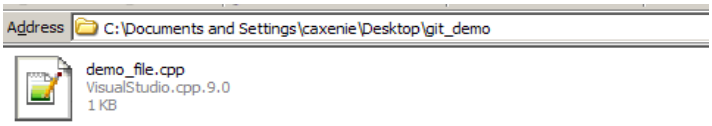
## Setting up the identity

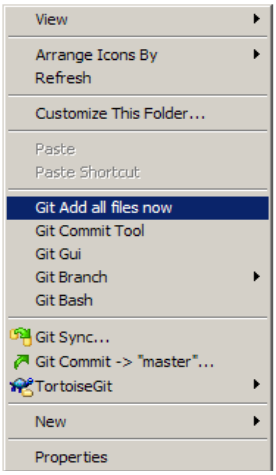
This step sets up the name (e.g. identifier for the user that adds changes to the repository) and the email in a global configuration file to be used in upcoming git operations.




## Adding files to the repository

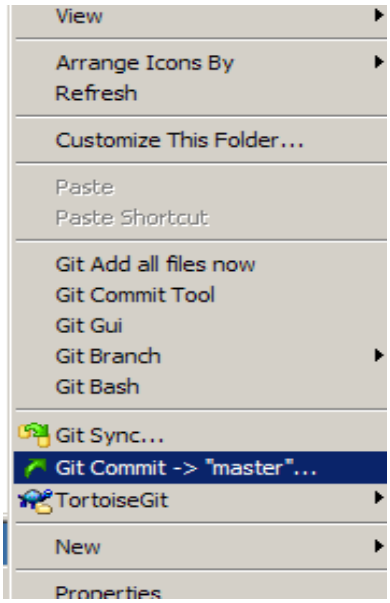
After initializing the repository the files must be added to the index, so that git will be able to keep track of all the changes you will add to them during the development process.



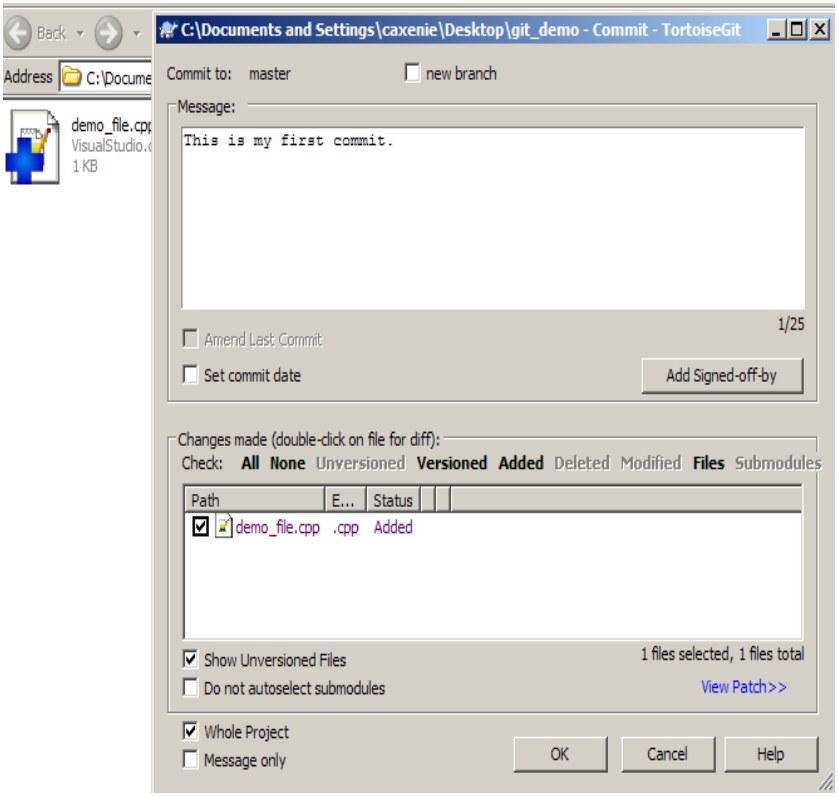



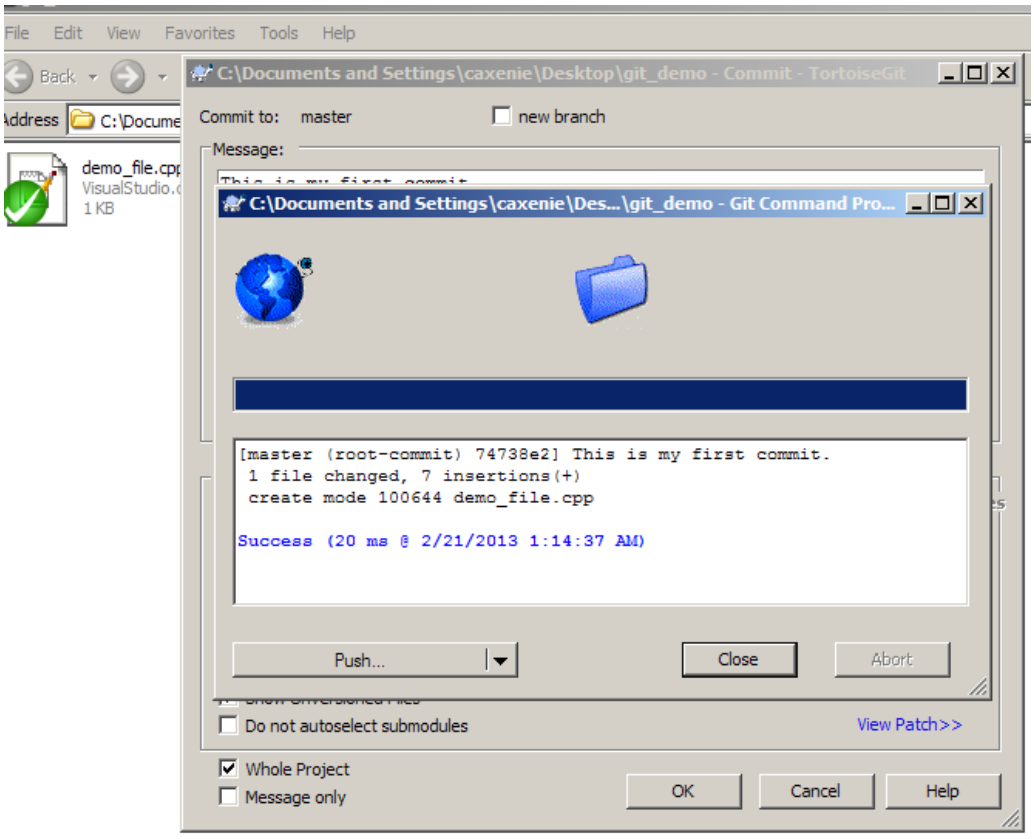
## Commiting changes

In order to commit the changes added to the repository file you should use the git commit in the right click menu.



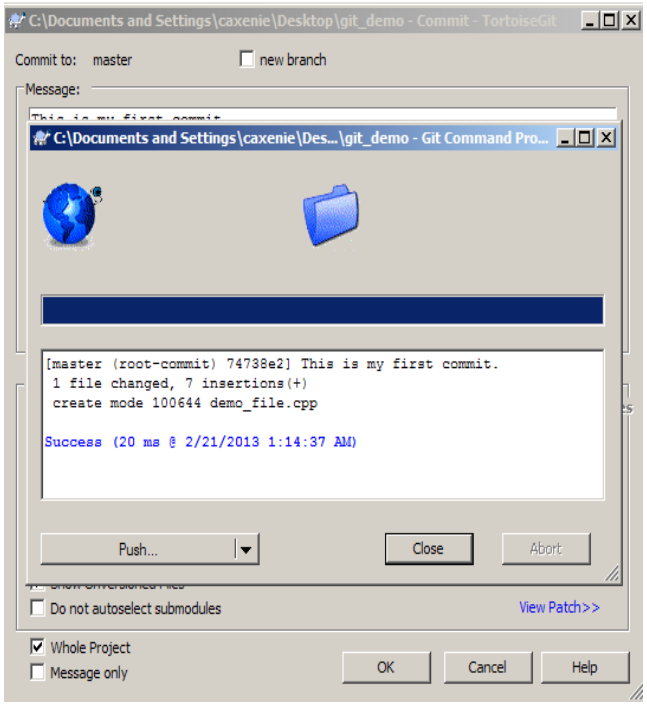

When commiting a new version of the file you should add a message, presenting in a short sentence, the change that was added from the previous version.

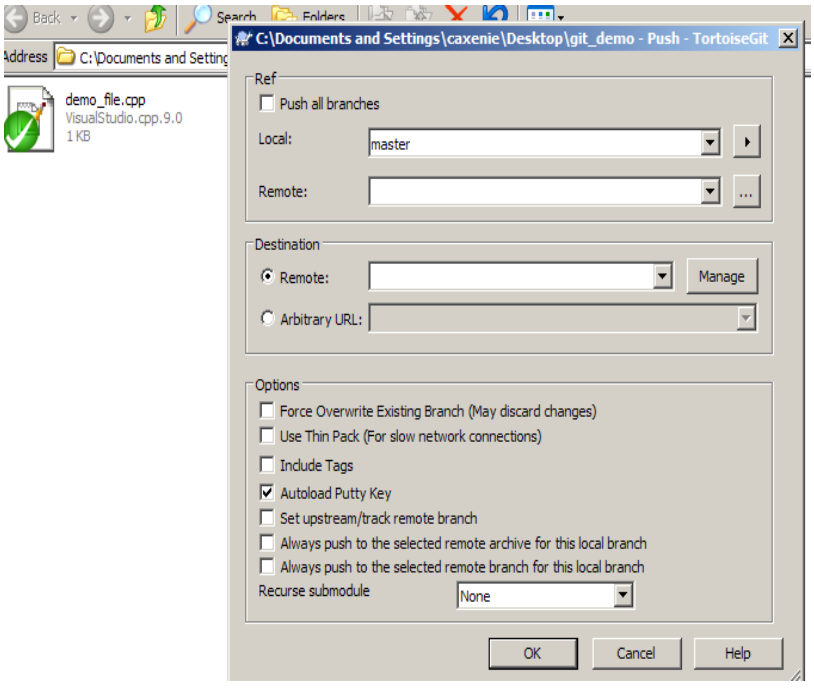




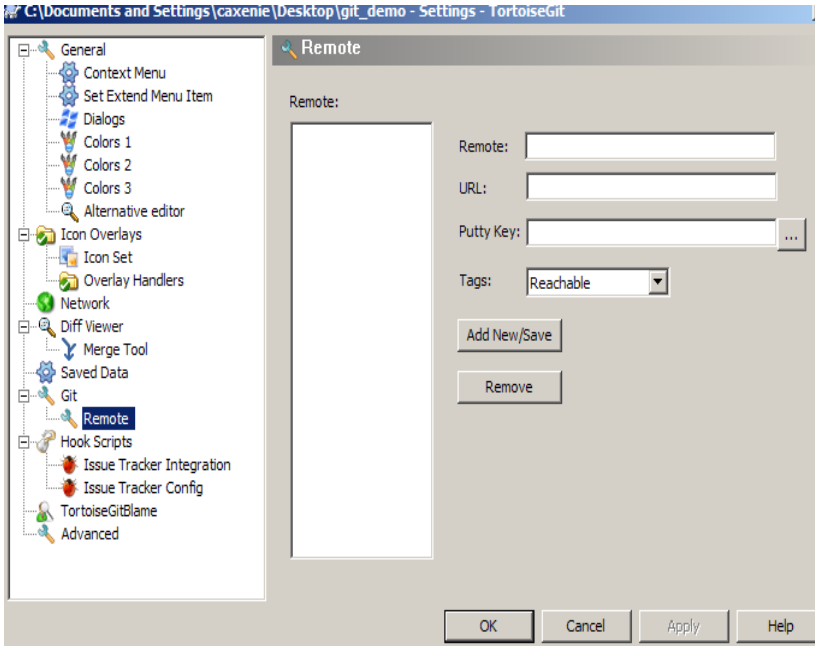

## Pushing changes in the remote repository

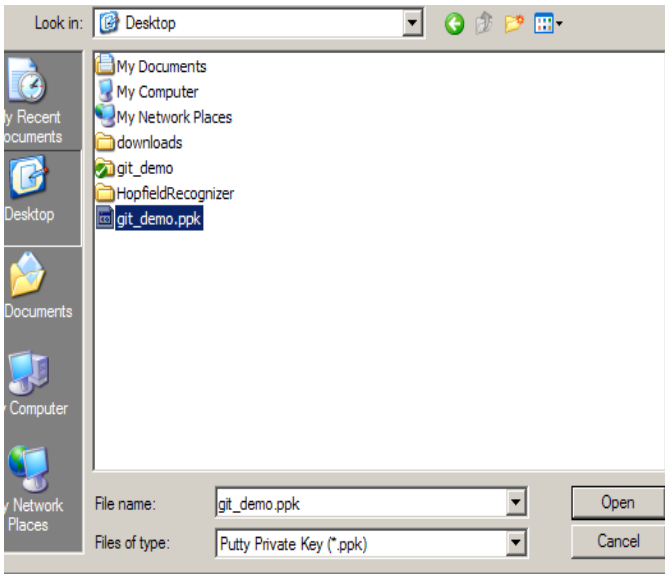
After commiting changes in the local copy of the repository it is time to push the changes on the remote repository selecting a remote Destination.



As mentioned earlier in order to execute operations on the remote repository you need to be authorized. In one of the previous steps we saved the SSH keys used to identify your machine when accessing the repositories. Now it's time to use this key to push changes to the repository. Remind that we've generated a key with the PuttyGen tool so we can import that key right now.

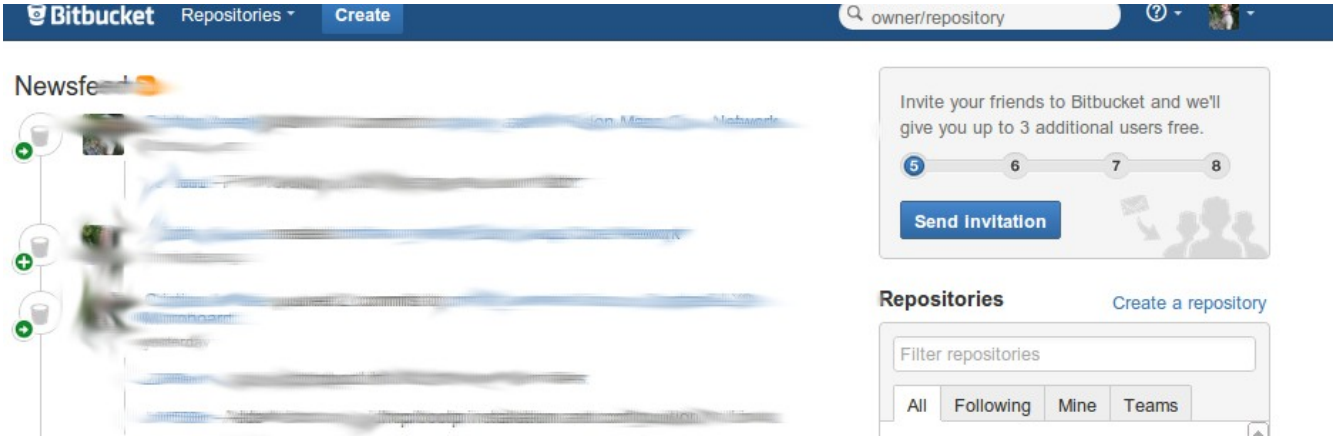


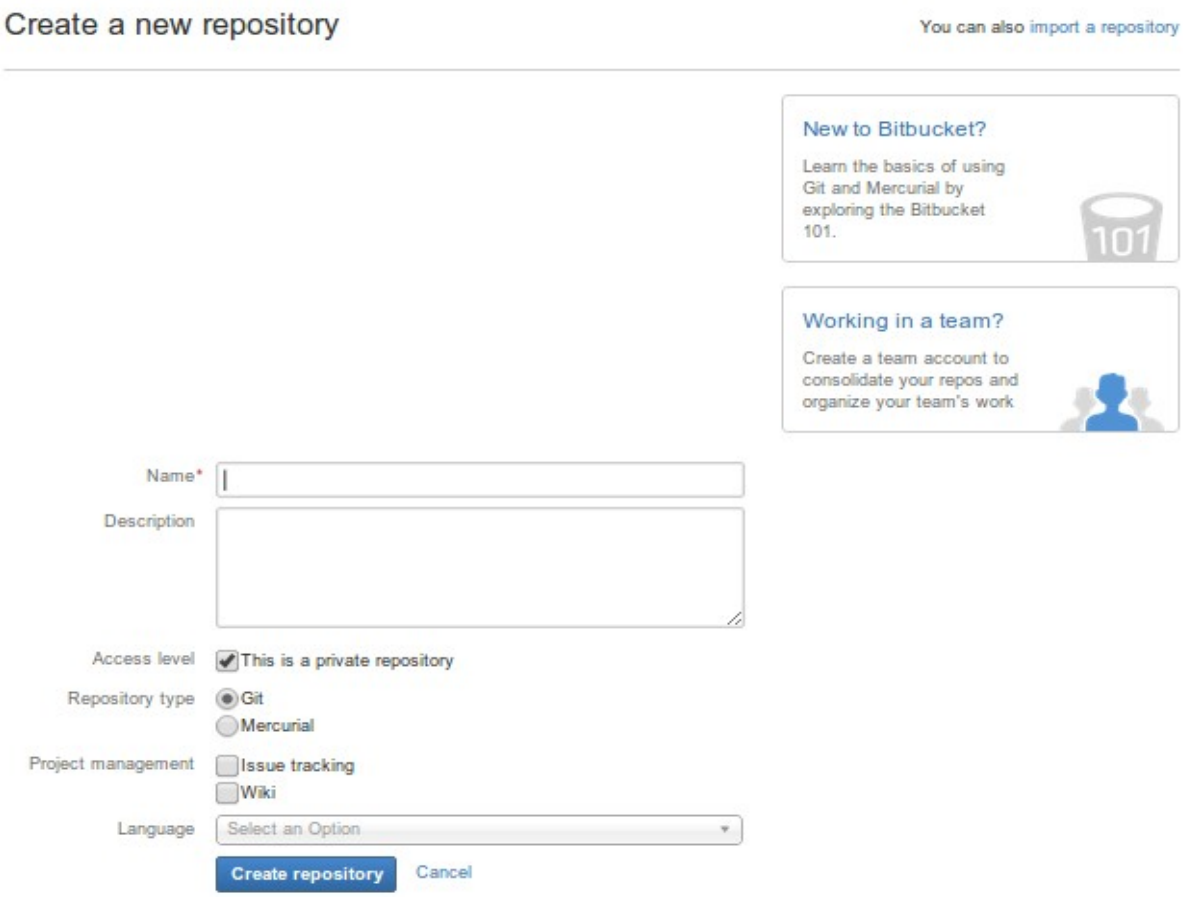
**Using Bitbucket for remote repository storage**

Bitbucket is an online free repository storage and management system that allows users to create and manage multiple repositories on the Internet. The advantage of this system is that can be accessed from every machine connected to the Internet, using authenticated access.

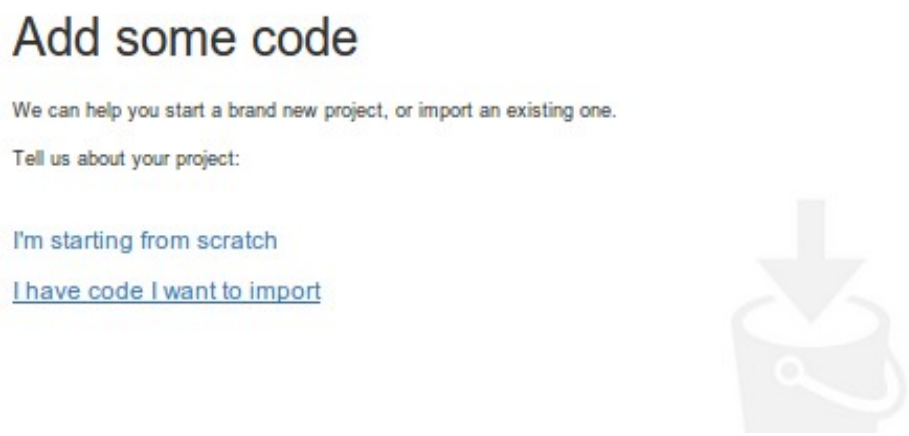
After creating an online account repositories can be added.



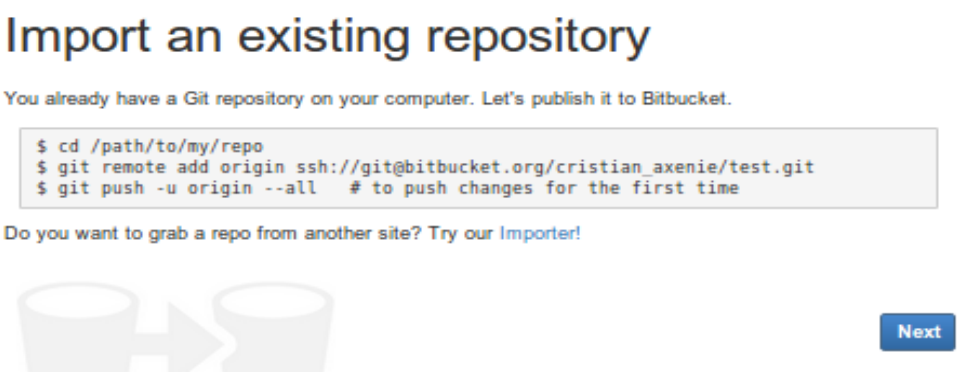

The next step is to add some information about your repository and choose access ritghts.



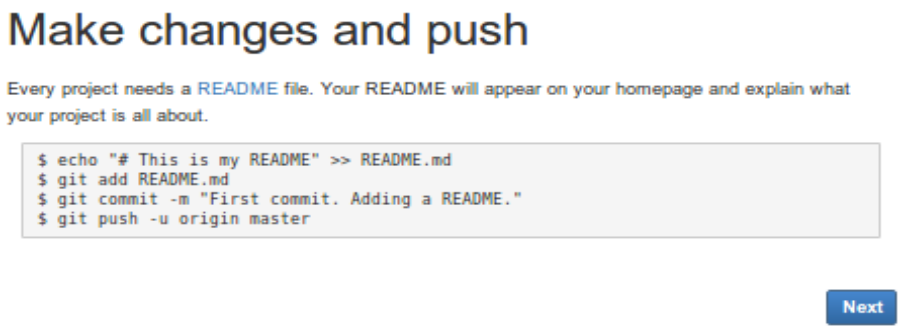
After creating the repository is time to add some code. You can either add new code or import existing code on your machine. The second option is the one commonly used and will be also used in our case.




In order to add existing code the user must link the local machine code with the newly created online repository. This is done by executing a remote add origin command from the command line or the Tortoise GIT menus.



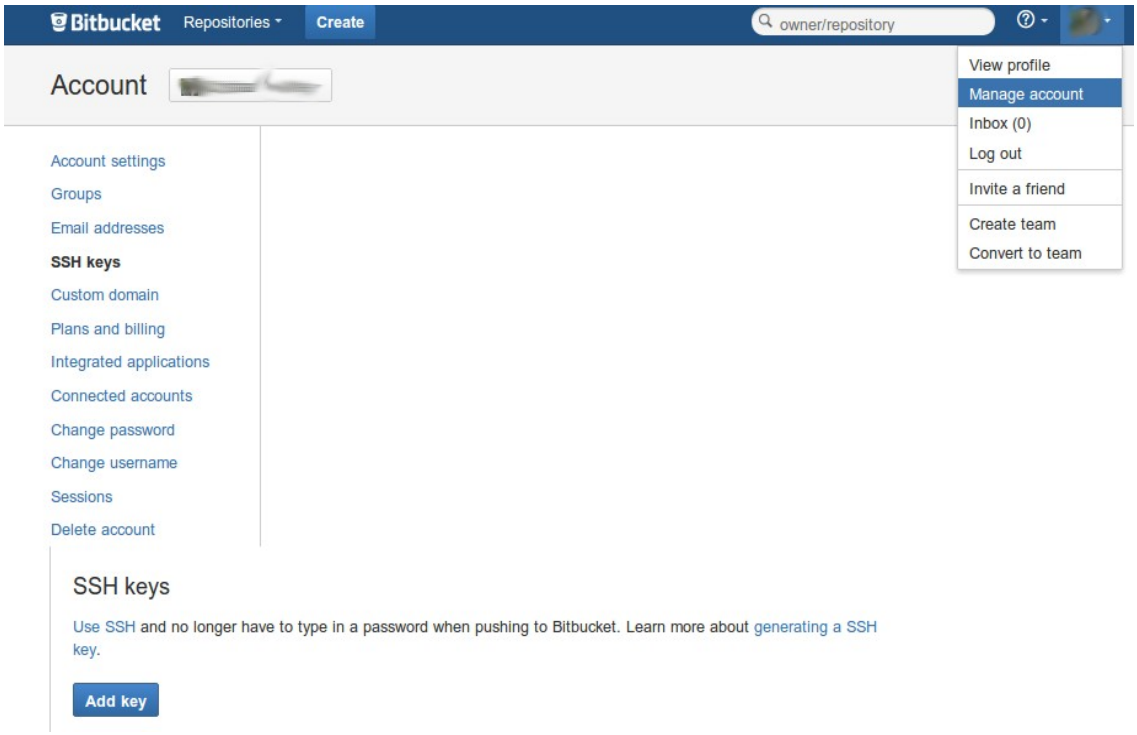

From now on everything is set and you can start developing and pushing changes to the remote repository.




**Important hint**

In order not to introduce a password all the time when pushing changes to the remote Bitbucket repository an user can use SSH keys to ensure automatic authenticated access.
The online Bitbucket interface offers support for SSH Keys saving for the development

computers the user uses to push changes to the remote repository.




On the website the users can find a small tutorial on how to use SSH protocol to access the remote repositories on Bitbucket.

https://confluence.atlassian.com/display/BITBUCKET/Using+the+SSH+protocol+with+bitbucket

Furthermore, the website also contains a small guide on how to generate and add SSH keys for authentication on  Linux, MacOS or MS Windows.

https://confluence.atlassian.com/display/BITBUCKET/How+to+install+a+public+key+on+your+bitbucket+account

**Available Internet tutorials**

You can find a very good overview, use cases and details on this site:
http://git-scm.com/book

Another resourceful site for problems or questions about git is stackoverflow
http://stackoverflow.com/

**Please send suggestions and comments to:**
**cristian.axenie@tum.de**