Installing the Git client

The following sections list the steps required to properly install and configure the Git clients - Git Bash and Git GUI - on a Windows 7 computer. Git is also available for Linux and Mac. The remaining instructions here, however, are specific to the Windows installation.

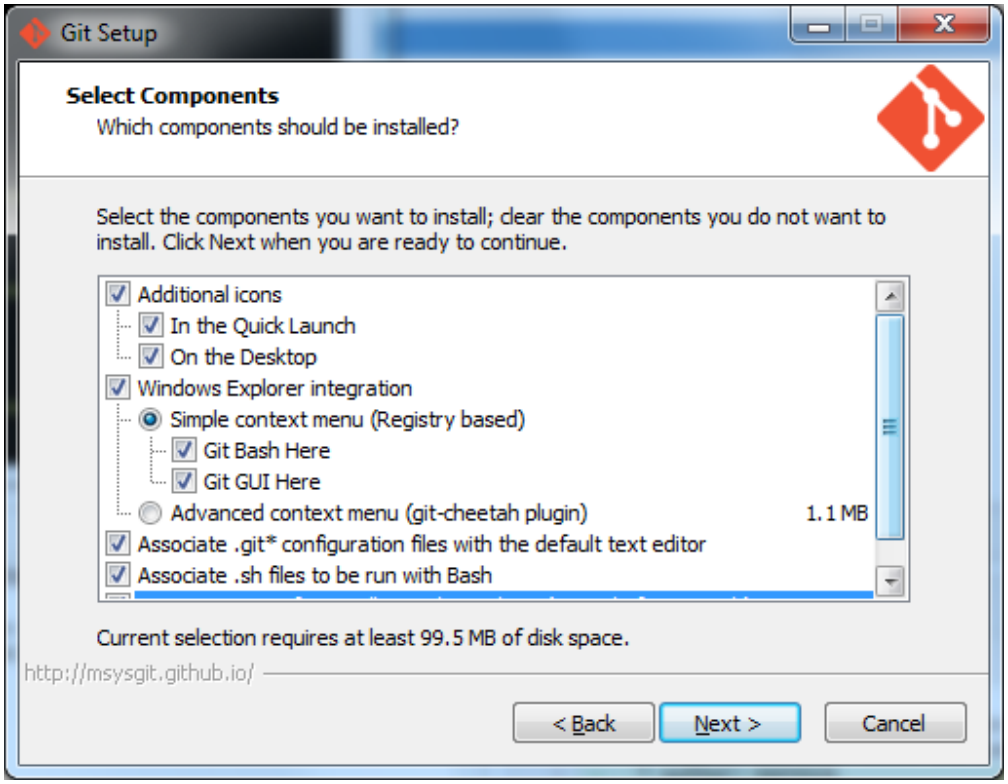
Be sure to exactly follow all of the steps in all four sections.

1. Git installation

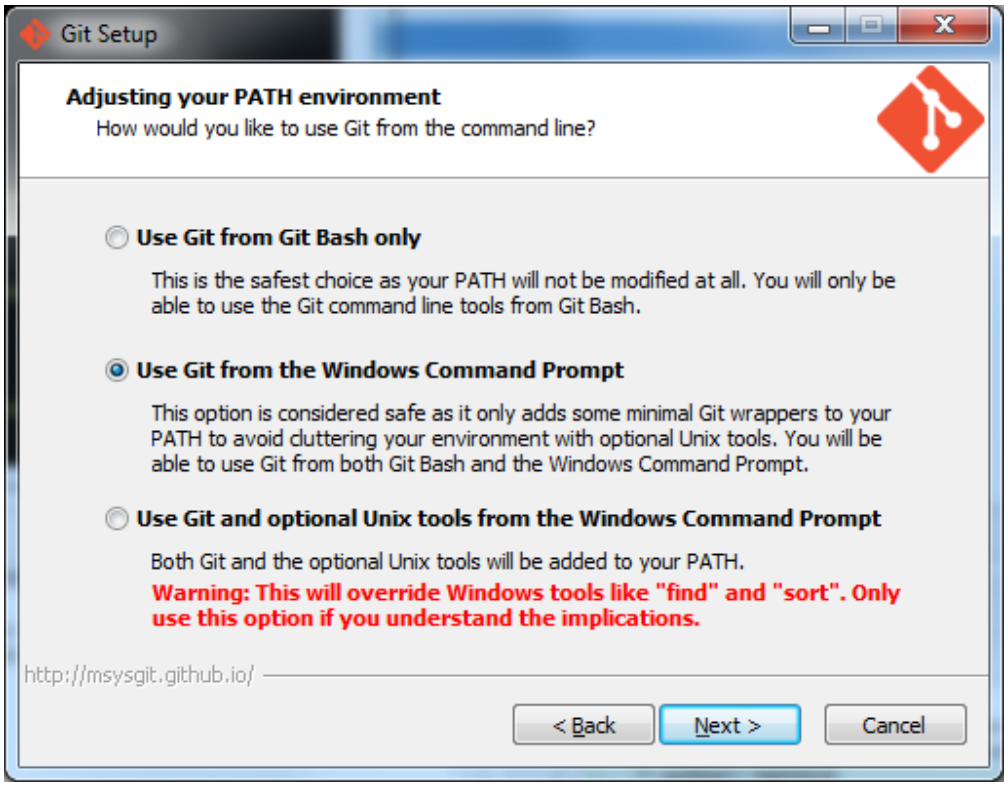
Download the Git installation program (Windows, Mac, or Linux) from http://git-scm.com/downloads.

When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, ***except in the three screens below where you do NOT want the default selections:***

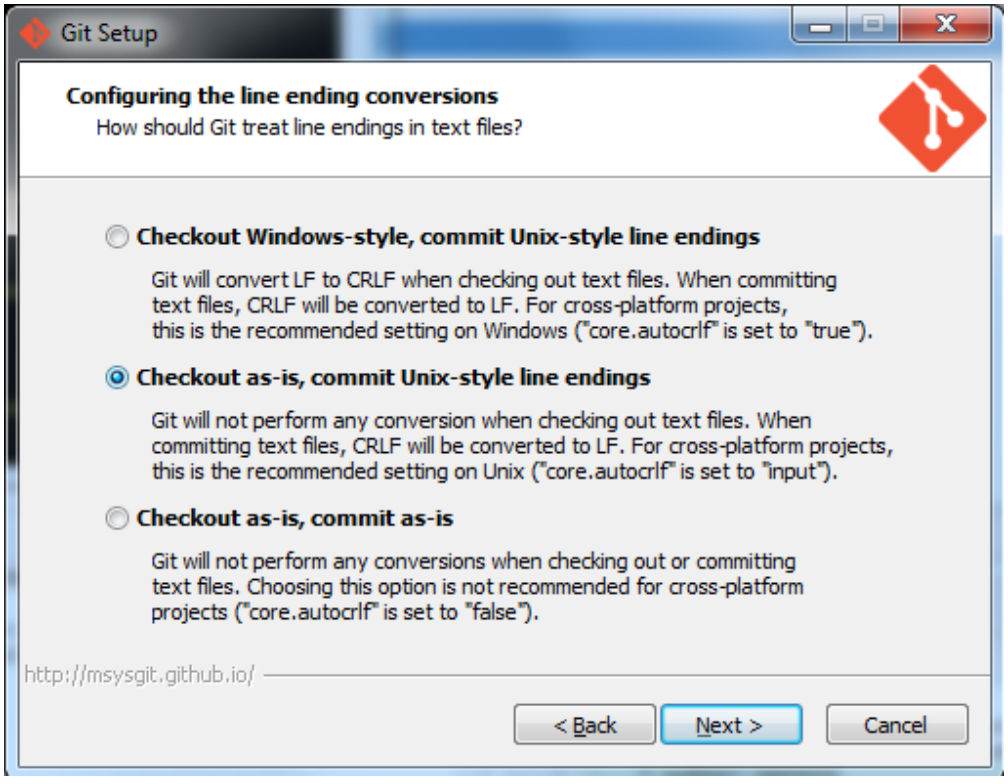
In the **Select Components** screen, select **Windows Explorer Integration** with **Simple Context Menu** selected as shown:



In the **Adjusting your PATH** screen, select the middle option (**Use Git from the Windows Command Prompt**) as shown:



In the **Configuring the line ending** screen, select the middle option (**Checkout as-is, commit Unix-style line endings**) as shown. This means that will eventually have unix-style (LF) terminators. By contrast, the Windows convention is CR-LF for line termination. Most Windows editors, however, have no problem with reading files containing only LF terminated lines; the notable exception is Notepad - so just don't use Notepad!



2. Configuring Git to ignore certain files

By default, Git tracks **all** files in a project. Typically, this is **NOT** what you want; rather, you want Git to ignore certain files (such as .**bak** files created by an editor, or .**class** files created by the Java compiler.

To automatically have Git ignore particular files, create a file named **.gitignore** (note that the filename begins with a dot) in the **C:\users\name** folder (where name is your MSOE login name).

NOTE: The .gitignore file must NOT have any file extension (e.g. .txt). Windows normally tries to place a files extension on a file you create from File Explorer. To avoid this, create the file from within an editor (e.g. Notepad++ or UltraEdit) and save the file without a file extension).

Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples provided at https://github.com/github/gitignore.)

```
#ignore all files in the bin/ directory (takes care of .class
files)
bin/
#ignore automatically generated backup files
*.bak
#ignore Enterprise Architect and Microsoft temporary files
*.ldb
~*
#ignore OS generated files
.DS_Store
.DS_Store?
Thumbs.db
#ignore all files that begin with a dot

.*
# do not ignore .classpath and .project
!.classpath
!.project
```

Note: You can always edit this file and add additional patterns for other types of files you might want to ignore.

3. Configuring Git default parameters

Once Git is installed, there is some remaining custom configuration you have to do. Follow the steps below:

a. From within File Explorer, right-click on any folder. A context menu appears containing the commands "**Git Bash here**" and "**Git GUI here**". These commands permit you to launch either Git client. For now, select **Git Bash here**.

b. Enter the command (replacing name as appropriate) `git config --global core.excludesfile c:/users/name/.gitignore`
   - This tells Git to use the .**gitignore** file you created in step 2
   - NOTE: You should simply copy/paste the commands shown here into the Git Bash window, in order to avoid typing errors.

c. At the $ prompt, enter the command `git config --global user.email "name@msoe.edu"`
   - This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name of course with your own MSOE email name.

d. Enter the command `git config --global user.name "Your Name"`
   - Git uses this to log your activity. Replace "Your Name" by your actual first and last name.

e. Enter the command `git config --global push.default simple`
   - This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.

f. Enter the command `git config --global merge.tool winmerge`
   - This configures Git to use the application WinMerge to resolve merging conflicts. You must have WinMerge installed on your computer. Get WinMerge here.

g. Enter the following commands to complete the WinMerge configuration:
   i. `git config --global mergetool.winmerge.name WinMerge`

ii. `git config --global mergetool.winmerge.trustExitCode true`
iii. If you install WinMerge to (for example) D:\WinMerge, enter
```
git config --global mergetool.winmerge.cmd "/d/WinMerge/WinMergeU.exe
-u -e -dl \"Local\" -dr \"Remote\" \$LOCAL \$REMOTE \$MERGED"
```
iv. If you install WinMerge to (for example) C:\Program Files (x86)\WinMerge, enter
```
git config --global mergetool.winmerge.cmd "\"C:\Program Files
(x86)\WinMerge\WinMergeU.exe\" -u -e -dl \"Local\" -dr \"Remote\"
\$LOCAL \$REMOTE \$MERGED"
```

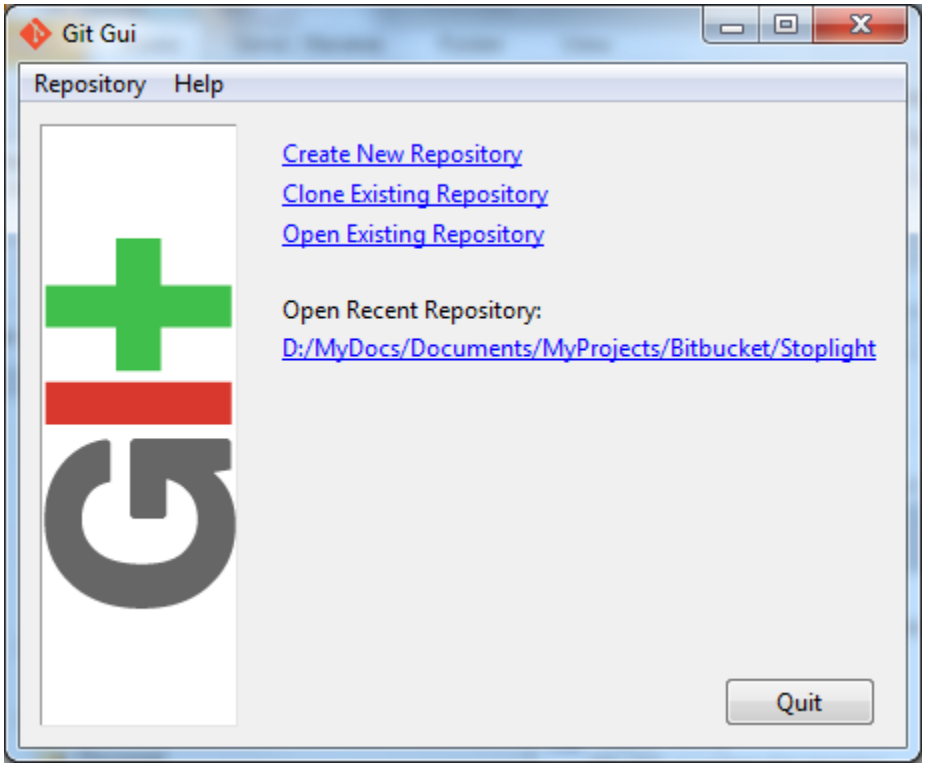
h. Enter the command `git config --global diff.tool winmerge`
   - This configures Git to use the application WinMerge to differences between versions of files.

i. Enter the commands to complete the WinMerge diff configuration:
   i. `git config --global difftool.winmerge.name WinMerge`
   ii. `git config --global difftool.winmerge.trustExitCode true`
   iii. If you install WinMerge to (for example) D:\WinMerge, enter
   ```
   git config --global difftool.winmerge.cmd "/d/WinMerge/WinMergeU.exe
   -u -e \$LOCAL \$REMOTE"
   ```
   iv. If you install WinMerge to (for example) C:\Program Files (x86)\WinMerge, enter
   ```
   git config --global difftool.winmerge.cmd "\"C:\Program Files
   (x86)\WinMerge\WinMergeU.exe\" -u -e \$LOCAL \$REMOTE"
   ```

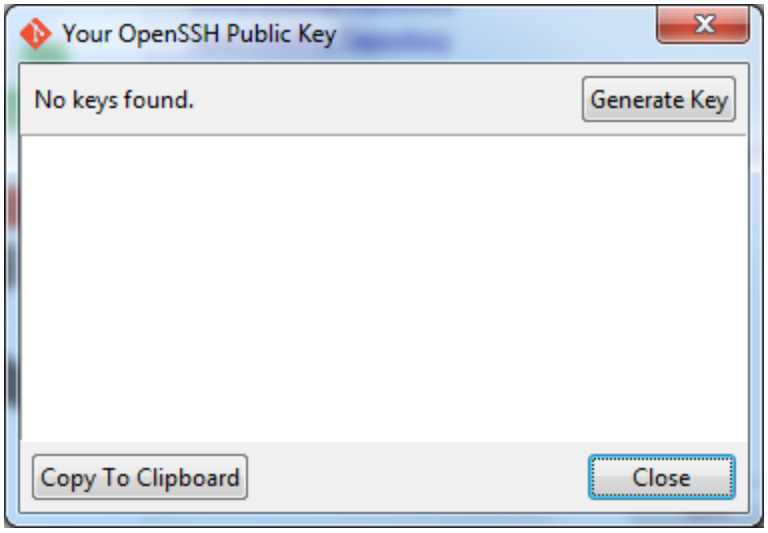
4. Generating public/private key pairs for authentication

You will eventually be storing your project files on a remote Bitbucket server using a secure network connection. The remote server requires you to authenticate yourself whenever you communicate with it so that it can be sure it is you, and not someone else trying to steal or corrupt your files. Bitbucket and Git together user public key authentication; thus you have to generate a pair of keys: a public key that you (or your instructor) put on Bitbucket, and a private key you keep to yourself (and guard with your life).

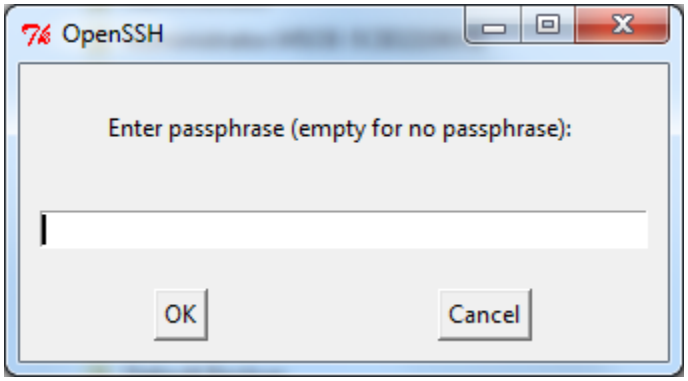
Generating the key pair is easy: From within File Explorer, right-click on any folder. From the context menu, select **Git GUI Here**. The following appears:



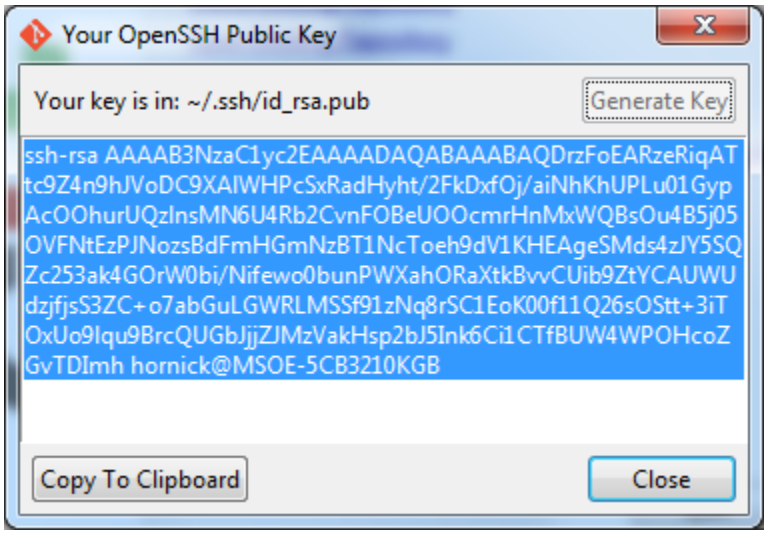
From the **Help** menu, select the **Show SSH Key** command. The following pup-up dialog appears:



Initially, you have no public/private key pair; thus the message "**No keys found**" appears withing the dialog. Press the **Generate Key** button. The following dialog appears:



Do **NOT** enter a passphrase - just press **OK** twice. When you do, the dialog disappears and you should see something like the following - but your generated key will be different:



The keys have been written into two files named **id_rsa** and **id_rsa.pub** in your **c:/users/name/.ssh** folder. Your instructor will show you how to install the public key (**id_rsa.pub**) into your Bitbucket account's **SSH keys** field.