

# 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.

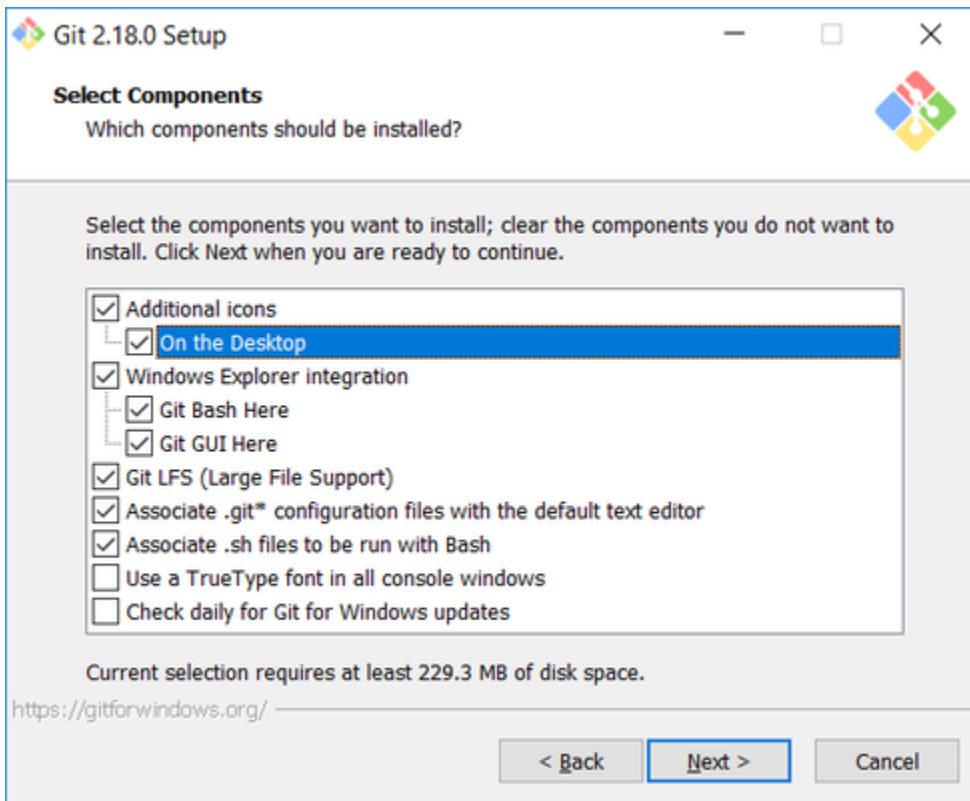
**Be sure to carefully follow all of the steps in all four sections.**

## 1. Git installation

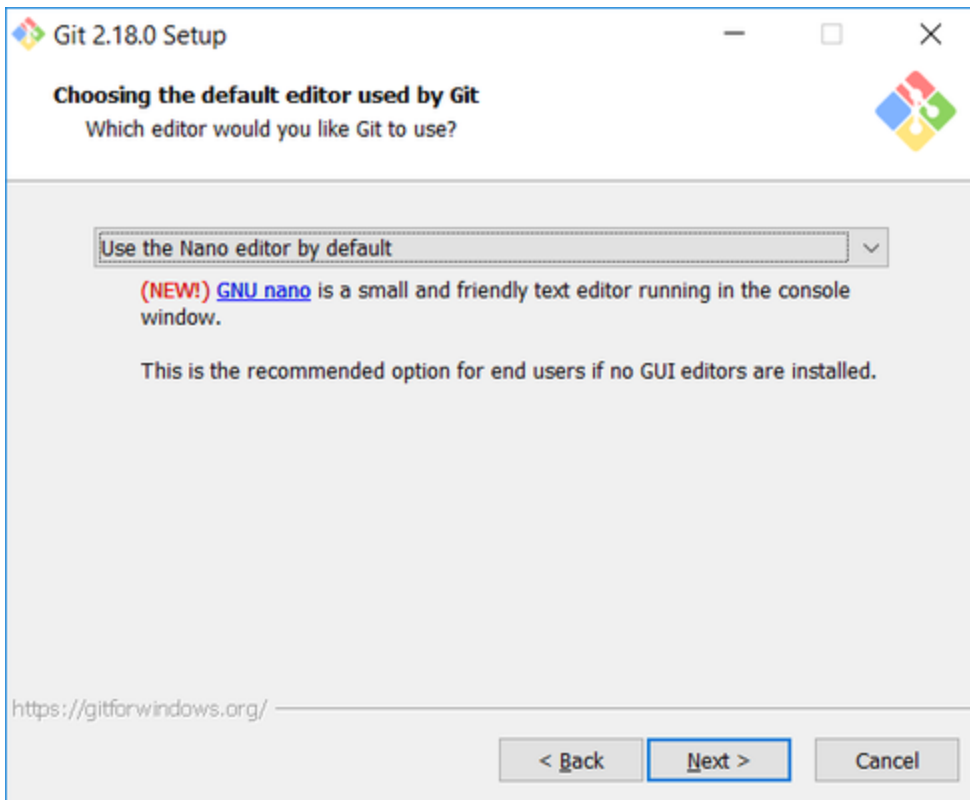
Download the Git installation program (Windows, Mac, or Linux) from <http://git-scm.com/downloads>.

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

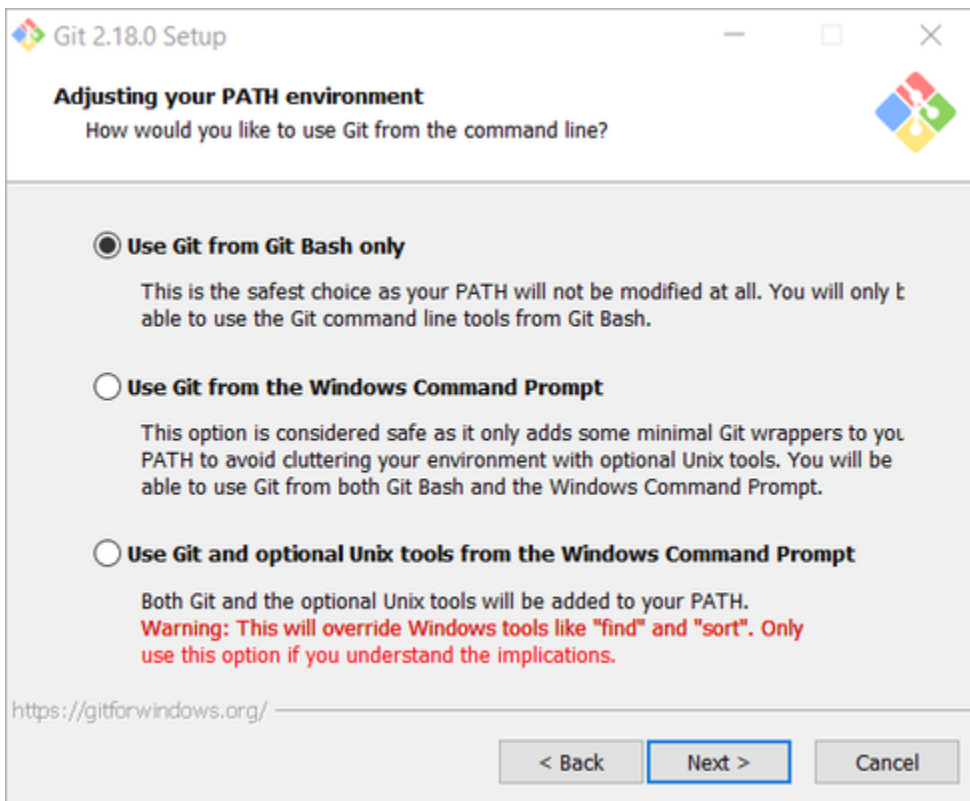
In the **Select Components** screen, make sure **Windows Explorer Integration** is selected as shown:



In the **Choosing the default editor used by Git** dialog, it is strongly recommended that you **DO NOT select the default VIM editor** - it is challenging to learn how to use it, and there are better modern editors available. Instead, choose **Notepad++ or Nano** - either of those is much easier to use. **It is strongly recommended that you select Notepad++, BUT YOU MUST INSTALL NOTEPAD++ first! Find the installation with Google.**

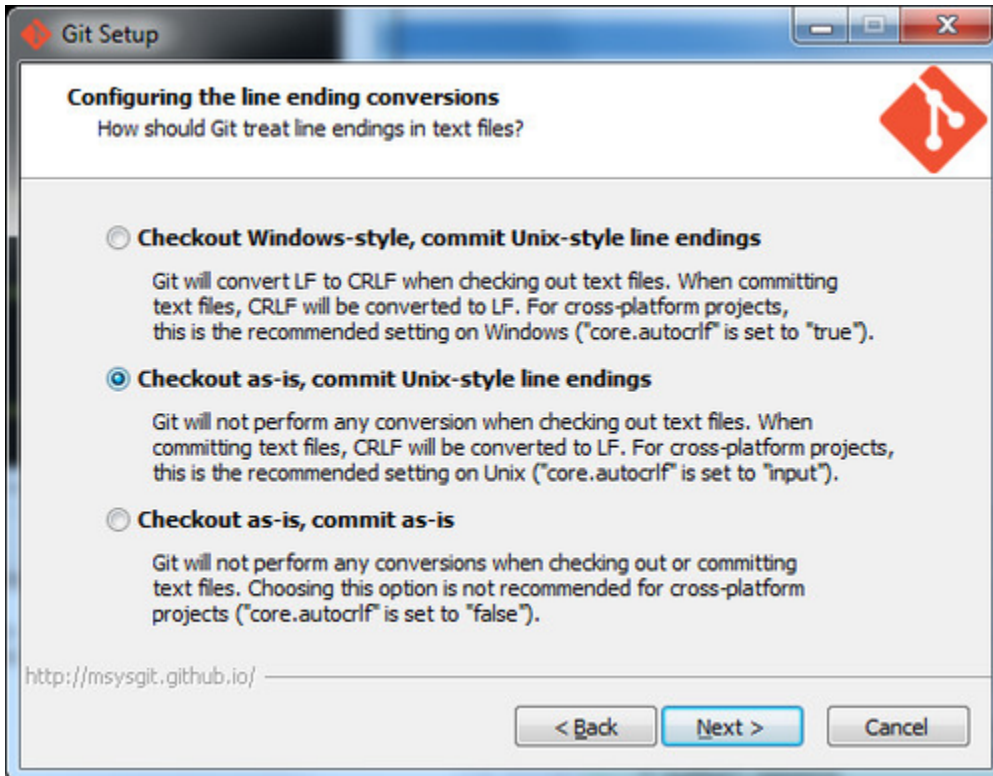


In the **Adjusting your PATH** screen, select the first option (**Use Git from Git Bash only**) as shown below. You could select the middle (default) option, but it adds unnecessary clutter to your path environment variable.



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, which most modern IDE's and editors use. By contrast, the (older DOS-based) Windows convention is CR-LF for line termination (so 1980's!). Most modern 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

**This part is extra important and required so that your repository does not get cluttered with garbage 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 file extension (.txt) on a file you create from File Explorer - and then it (by default) HIDES the file extension. 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>.)

```

#Lines (like this one) that begin with # are comments; all other lines are rules

#ignore all files in the bin/ and /out directories (takes care of .class files for some
IDEs)
bin/
out/
#ignore automatically generated backup files (created by some editors)
*.bak
#ignore Enterprise Architect and Microsoft temporary files
*.ldb
~*

#ignore all files that begin with a dot (this includes .classpath, .project, .iml, etc)
.*

# Any files you do want not to ignore must be specified starting with !
# For example, if you didn't want to ignore .classpath, you'd uncomment the following rule:
# !.classpath

```

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. 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.

### 4. Optional: Configure Git to use a custom application (WinMerge) for comparing file differences

It is recommended that you skip this step unless you really are attached to using WinMerge for file comparison tasks.

- a. 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 first. Get WinMerge at <http://winmerge.org/downloads/>.**
- b. 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 the default location (that is, 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"
```

iv. If you install WinMerge to an alternate location (for example, D:\WinMerge), enter  
`git config --global mergetool.winmerge.cmd "/d/WinMerge/WinMergeU.exe -u -e -dl \"Local\" -dr  
\"Remote\" \LOCAL \REMOTE \MERGED"`

c. Enter the command `git config --global diff.tool winmerge`

- This configures Git to use the application WinMerge to differences between versions of files.

d. 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 the default location (that is, C:\Program Files (x86)\WinMerge), enter  
`git config --global difftool.winmerge.cmd "\"C:\Program Files  
(x86)\WinMerge\WinMergeU.exe\" -u -e \LOCAL \REMOTE"`

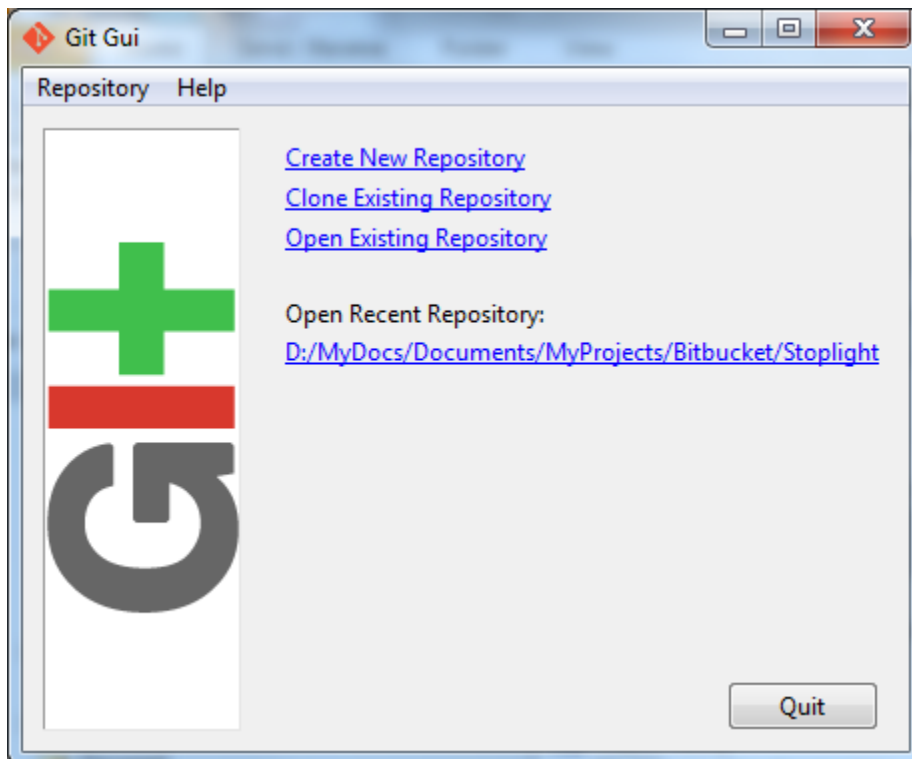
iv. If you install WinMerge to an alternate location (for example, D:\WinMerge), enter  
`git config --global difftool.winmerge.cmd "/d/WinMerge/WinMergeU.exe  
-u -e \LOCAL \REMOTE"`

## 5. Generating public/private key pairs for authentication

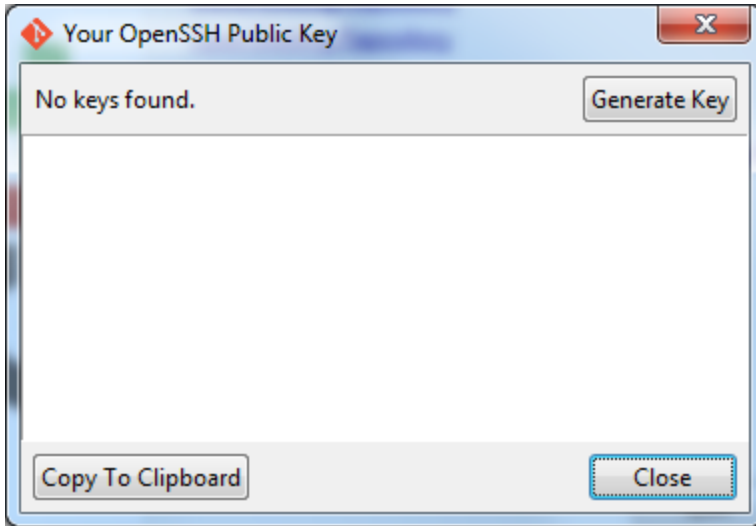
**This part is critical and used to authenticate your access to the repository.**

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 use 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).

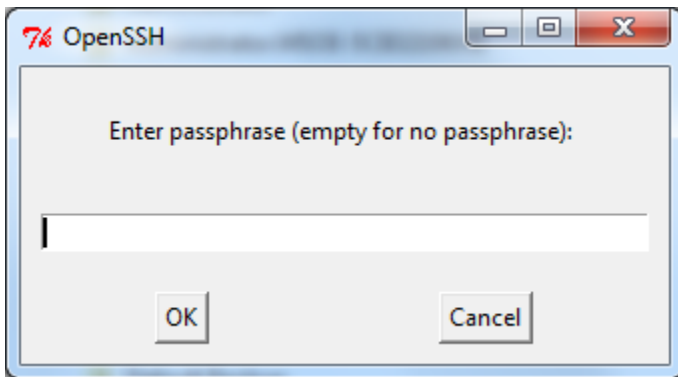
Generating the key pair is easy: From within File Explorer, right-click on any folder. From the context menu, select **Git GUI Here**. The following appears:



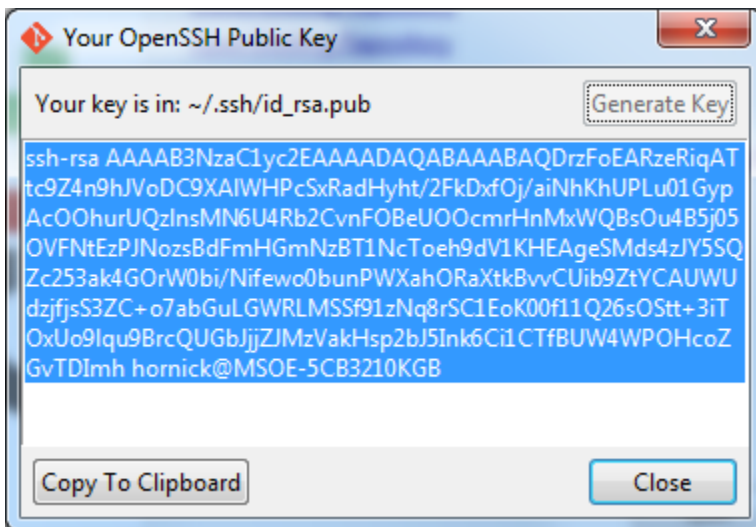
From the **Help** menu, select the **Show SSH Key** command. The following pup-up dialog appears:




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/username/.ssh** folder (where **username** is your MSOE user name). Don't ever delete these files! To configure Bitbucket to use this key:

1. Click on the **Copy to Clipboard** button in the Git GUI Public Key dialog.
2. Log in to Bitbucket
3. Click on your picture or the  icon in the left pane and select **Settings**.

4. Select **SSH keys** under **Security**.
5. Click on the **Add key** button.
6. Enter a name for your key in the **Label** box in the Bitbucket window. If your key is ever compromised (such as someone gets a copy off of your laptop), having a clear name will help you know which key to delete. A good pattern to follow is to name the computer used to generate the key followed by the date you generated it; for instance: "MSOE laptop key 2012-02-28".
7. Paste the key from the Clipboard into the **Key** text box in the Bitbucket window, and add it.

You should now be able to access your repository from your laptop using the ssh protocol without having to enter a password. Protect the key files - other people can use them to access your repository as well! If you have another computer you use, you can copy the id\_rsa.pub file to the .ssh folder on that computer or (better yet) you can generate another public/private key pair specific to that computer.