

# <Installing and Configuring 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 10/11 computer. Git is also available for Linux and Mac. The remaining instructions here, however, are specific to the Windows installation.

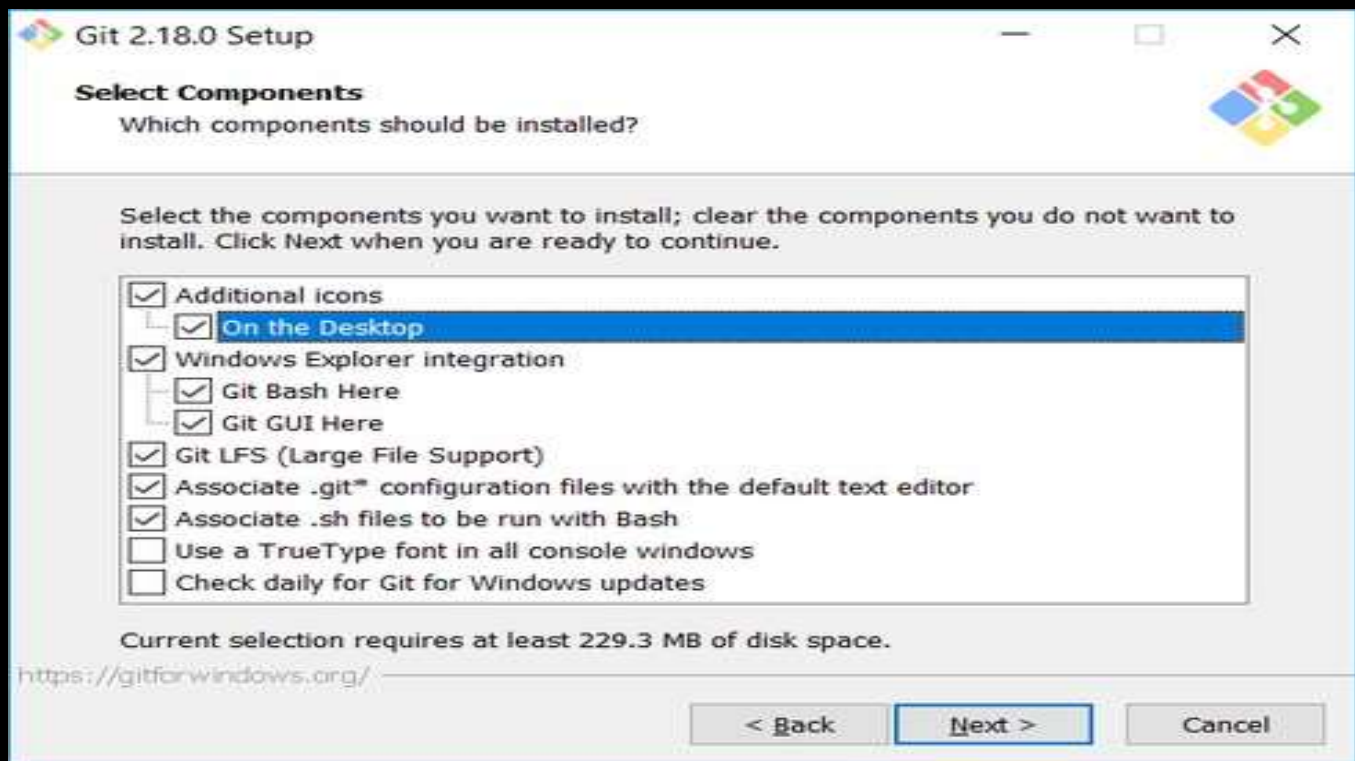
**Be sure to carefully follow all of the steps in the first five sections.** The last section, 6, is optional. There is also a section on common problems and possible fixes at the bottom of the document.

## 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.

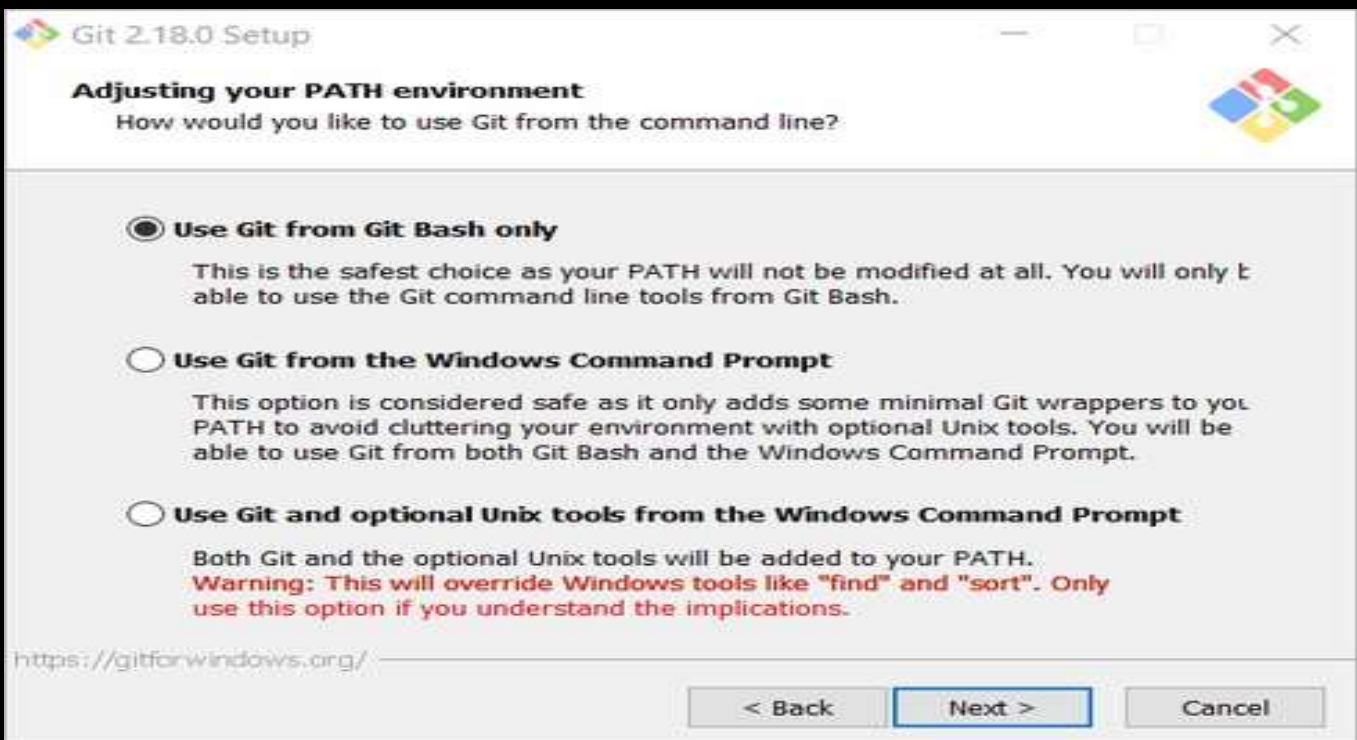


In the **Adjusting your PATH** screen, all three options are acceptable:

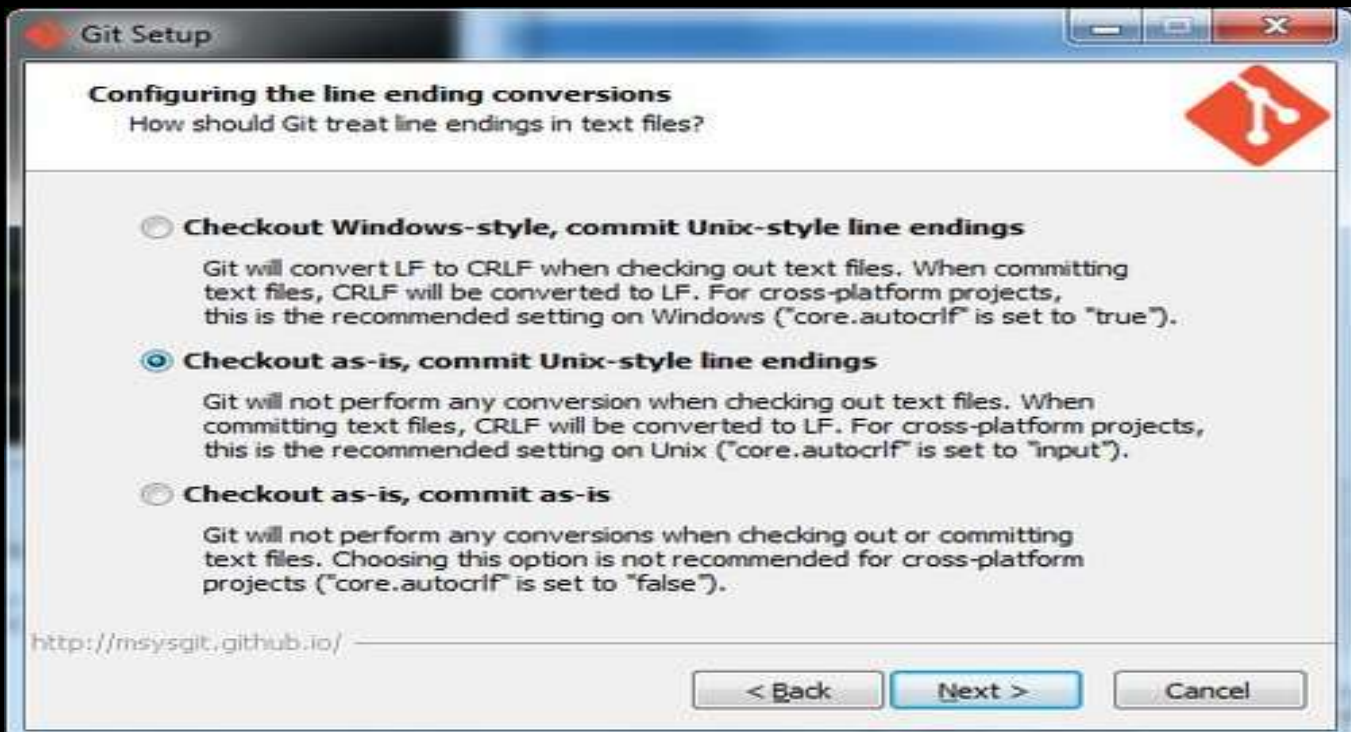
**Use Git from Git Bash only:** no integration, and no extra commands in your command path

**Use Git from the Windows Command Prompt:** adds flexibility - you can simply run git from a Windows command prompt, and is often the setting for people in industry - but this does add some extra commands.

**Use Git and optional Unix tools from the Windows Command Prompt:** this is also a robust choice and useful if you like to use Unix commands like grep.



In the **Configuring the line ending** screen, select the middle option (**Checkout as-is, commit Unix-style line endings**) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editors support. The Windows convention (CR-LF line termination) is only important for Notepad (as opposed to Notepad++), but if you are using Notepad to edit your code you may need to ask your instructor for help.



## 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 have Git automatically 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 a useful 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

# common build products to be ignored at MSOE
*.o
*.obj
*.class
*.exe

# common IDE-generated files and folders to ignoreworkspace.xml
bin/out/
.classpath
# uncomment following for courses in which Eclipse .project files are not checked in# .project

#ignore automatically generated files created by some common applications, operatingsystems
*.bak
*.log
*.ldb
~*
.DS_Store*
.*_Thumbs.db

# 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. Note that you can also have a

.gitignore file in any folder naming additional files to ignore. This is useful for project-specific build products.

### 3. Configuring Git default parameters

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

- 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**.

- Enter the command (replacing the name ) **git config --global core.excludesfile c:/users/name/. gitignore**

This tells Git to use the .gitignore file you created in step 2

NOTE: TO avoid typing errors, copy and paste the commands shown here into the Git Bash window, using the arrowkeys to edit the red text to match your information.

- Enter the command **git config --global user.email "Msoe email name"**

This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own MSOE email name.

- d. Enter the command `git config --global user.name "TekadeSukant"`

Git uses this to log your activity. Replace "TekadeSukant" by your actual first and last name.

- e. Enter the command `git config --global push.default simple`

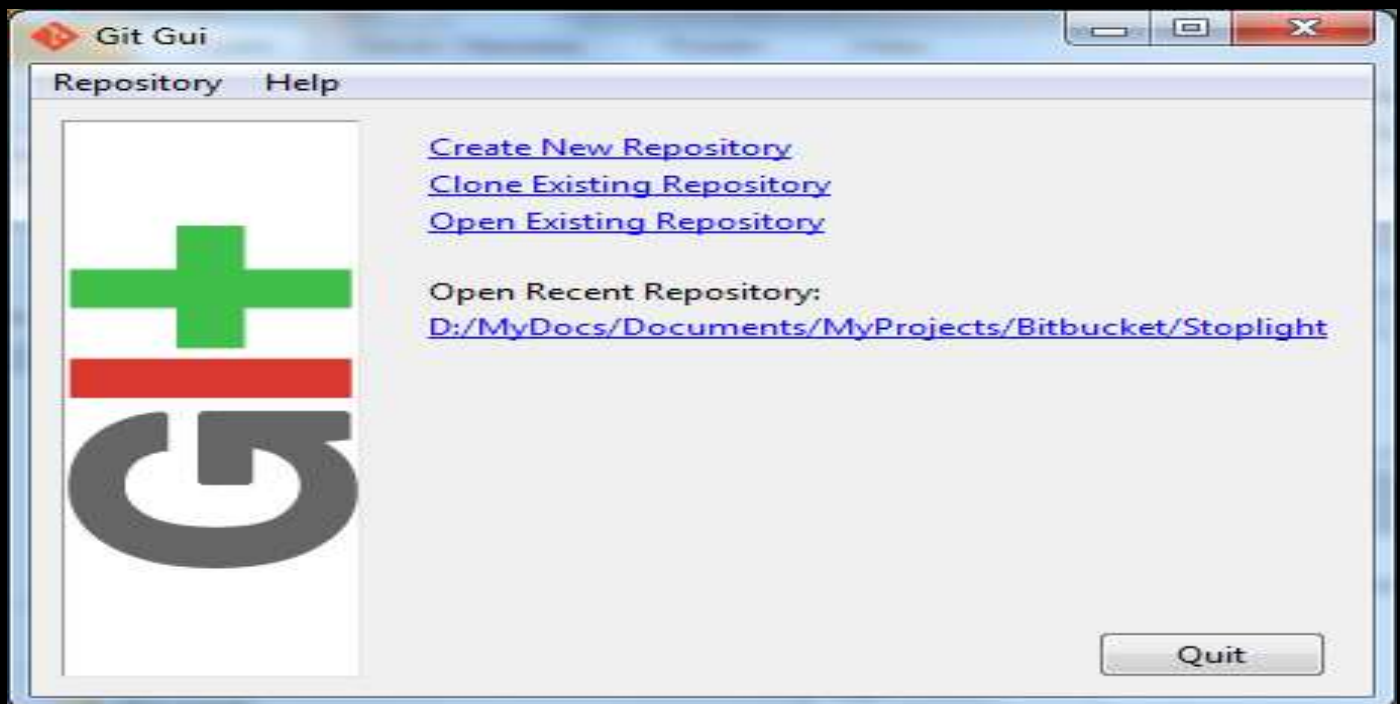
This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the masterbranch, forcing a merge.

#### 4. 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 or other 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:

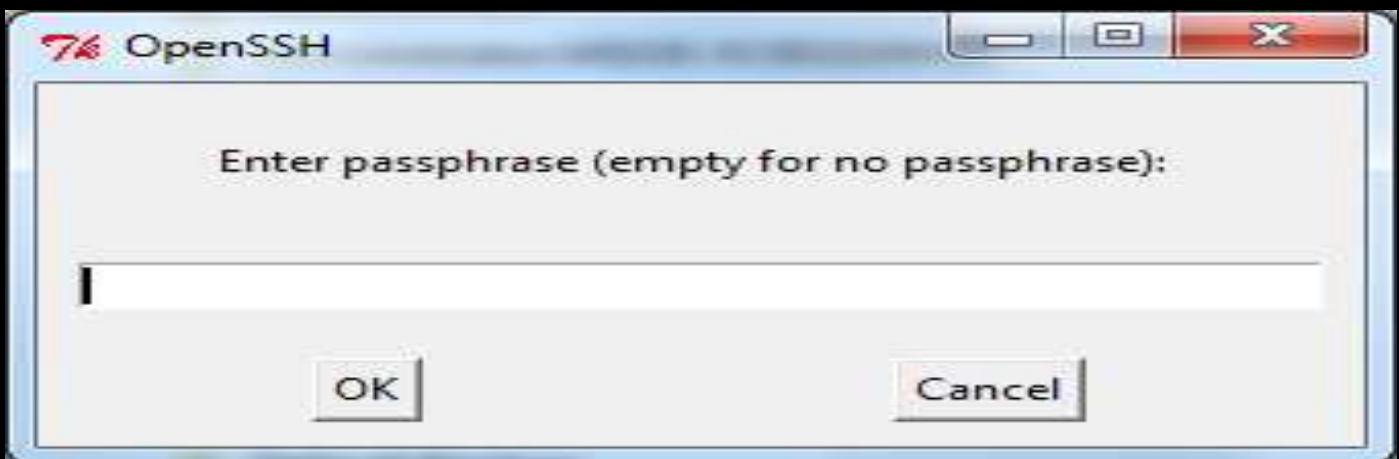


From the **Help** menu, select the **Show SSH Key** command. The following pop-up dialog appears:

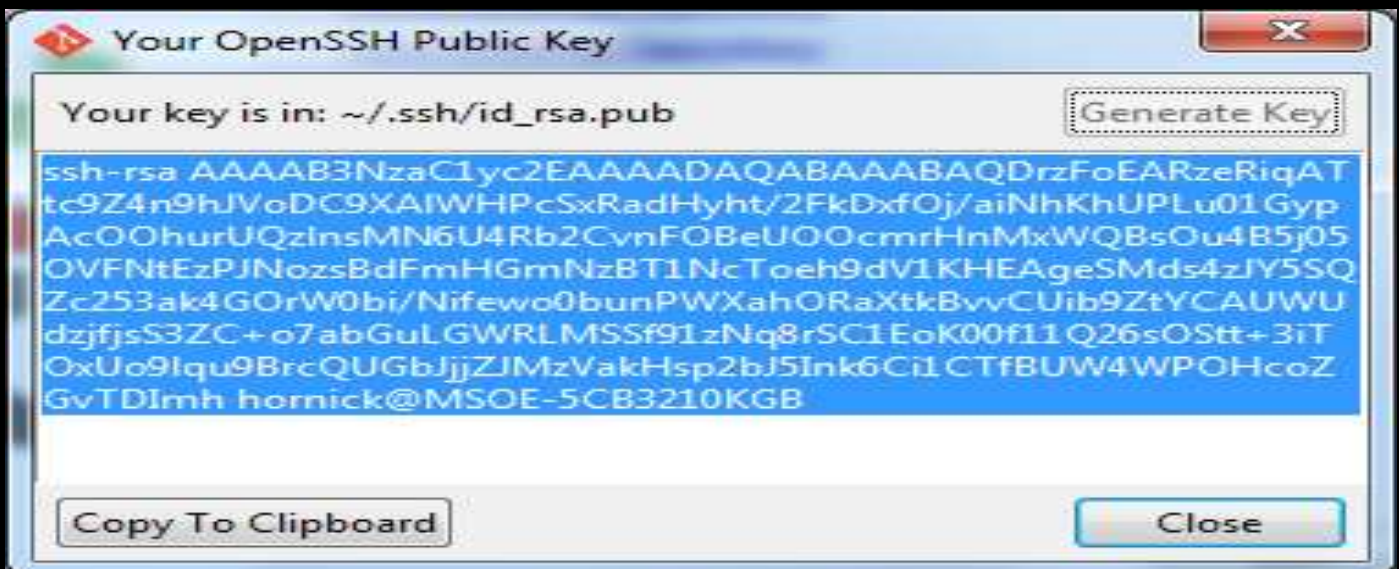





Initially, you have no public/private key pair; thus, the message "No keys found" appears within the dialog. Press the **Generate Key** button. The following dialog appears:



**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 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. Configuring other repositories (such as GitLab) is very similar.

## 5. Authenticating with private keys

Linux, Mac users:

1. Open a terminal prompt.
2. Type the commands

```
eval `ssh-agent` ssh-add
```

Note that backticks (`) are used, not forward ticks ('). The second command assumes your key is in the default location, `~/.ssh/id_dsa`. If it is somewhere else, type `ssh-add path-to-private-key-file`. Note that the directory containing the ssh key cannot be readable or writeable by other users; that is, it needs mode 700. You can add these commands to your `.bashrc` file so they are executed every time you log in.

Otherwise, the keys only remain active until you close the shell you are using or log out.

## Windows users:

1. Install Pageant if it is not installed. It is usually installed with PuTTY and PuTTYgen.
2. Start Pageant and select **Add Key**.
3. Browse to your .ppk file, open it, and enter the passphrase if prompted.

If git pull or get push cannot connect, you might need to add a system variable GIT\_SSH set to the path to the plink.exe executable. Go to Windows Settings, enter "system environment" in the search box, open the "Edit the system environment variables" item, click on Environment Variables..., then New... in the System Variables section (the bottom half), enter GIT\_SSH for the name, and browse to plink.exe for the value. Save the setting, then reboot your computer.

You will need to re-add the key to Pageant every time you log in to Windows (say, after a reboot). The command start/b pageantc:\path\to\file.ppk will open the file in pageant if you have pageant in your %PATH% variable.

## 6. 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
- iv. `git config --global difftool.winmerge.cmd "\"C:\Program Files (x86)\WinMerge\WinMergeU.exe\" -u -e \\\$LOCAL \\\$REMOTE"`
- v. 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"`

## Common problems and possible fixes

Problem	Fix
In Linux, pushes and pulls do not work with the error message " <b>permission denied</b> "	Check the ownership and permissions on the .git folder. Use "ls -ld .git" to check this. If the owner is root, use the chown command to fix that. If the permissions are not rwx for the owner, use the chmod command to set the permissions to 770.
Even though your keys are set up and available through ssh-agent or pageant, git push and pull commands prompt you for your username and password.	Confirm the repository address by entering "git remote -v show". If it shows the HTTPS protocol (starts with HTTPS), then you need to clone the repository using the SSH protocol specifier.