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:

🚸 Git Setup
Select Components Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
 Additional icons In the Quick Launch On the Desktop Windows Explorer integration Simple context menu (Registry based) Git Bash Here Git GUI Here Advanced context menu (git-cheetah plugin) 1.1 MB Associate .git* configuration files with the default text editor Associate .sh files to be run with Bash
Current selection requires at least 99.5 MB of disk space. http://msysgit.github.io/
< <u>B</u> ack <u>N</u> ext > Cancel

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

🚯 Git Setup	
Adjusting your PATH environment How would you like to use Git from the command line?	
🔘 Use Git from Git Bash only	
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.	
Ise Git from the Windows Command Prompt	
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid duttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.	
\bigcirc Use Git and optional Unix tools from the Windows Command Prompt	
Both Git and the optional Unix tools will be added to your PATH. Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.	
http://msysgit.github.io/	
< <u>B</u> ack <u>N</u> ext > Cancel	

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!

🔶 Git Setup			
Configuring the line endin How should Git treat line end	g conversions ndings in text files?		
Checkout Windows	-style, commit Unix-style line endings		
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").			
Ocheckout as-is, commit Unix-style line endings			
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").			
Checkout as-is, con	nmit as-is		
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").			
http://msysgit.github.io/			
	< <u>B</u> ack <u>N</u> ext > Cancel		

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 fol der (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



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:

🚸 Your (OpenSSH Public Key	×
No keys	found.	Generate Key
Copy To	Clipboard	Close

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:

76 OpenSSH			
Enter passphrase (empty for no passphrase):			
1			
ОК	Cancel		

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:

Your OpenSSH Public Key	×
Your key is in: ~/.ssh/id_rsa.pub	Generate Key
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAA tc9Z4n9hJVoDC9XAIWHPcSxRadHyht/2FkDxf AcOOhurUQzInsMN6U4Rb2CvnFOBeUOOcm OVFNtEzPJNozsBdFmHGmNzBT1NcToeh9dV Zc253ak4GOrW0bi/Nifewo0bunPWXahORaXt dzjfjsS3ZC+o7abGuLGWRLMSSf91zNq8rSC1E OxUo9Iqu9BrcQUGbJjjZJMzVakHsp2bJ5Ink6Ci GvTDImh hornick@MSOE-5CB3210KGB	BAQDrzFoEARzeRiqAT Oj/aiNhKhUPLu01Gyp rHnMxWQBsOu4B5j05 1KHEAgeSMds4zJY5SQ kBvvCUib9ZtYCAUWU oK00f11Q26sOStt+3iT i1CTfBUW4WPOHcoZ
Copy To Clipboard	Close

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.