Guide to GitHub

Ethan Cheng, Yicheng Wang

Contents

1 Introduction

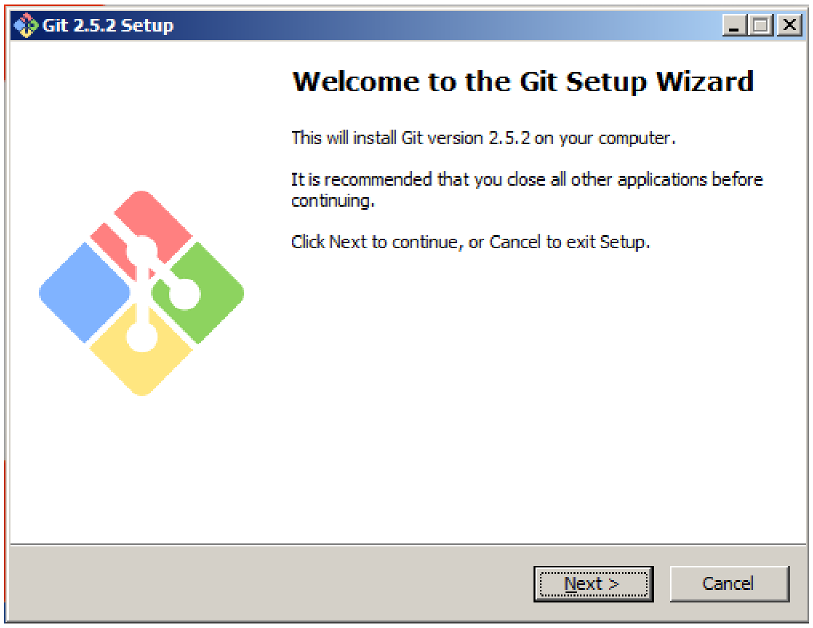
I made this guide because GitHub is an awesome tool, but using it incorrectly will cause a lot of problems. This guide is to serve as a tutorial and a reference for people using GitHub.

2 Installation

2.1 Windows

There are two ways of using GitHub on Windows. One way is via the graphical interface available for download here: `https://desktop.github.com`. However, most of the sections of this guide will be focused on command-line versions of `Git`, so we'll be using the second way of using GitHub on Windows: **Git Bash**, available for download here: `https://git-for-windows.github.io` and here: `http://git-scm.com/download/win`. (Yes, for those reading the original PDF version of this guide, these links are clickable.) Once you have downloaded it, run the installer. (The following pictures might look a bit different on your computer since I'm running Windows off Parallels, but most of the content is the same.)

Step 1. When you open the installer, you should see this:



Git 2.5.2 Setup

Welcome to the Git Setup Wizard

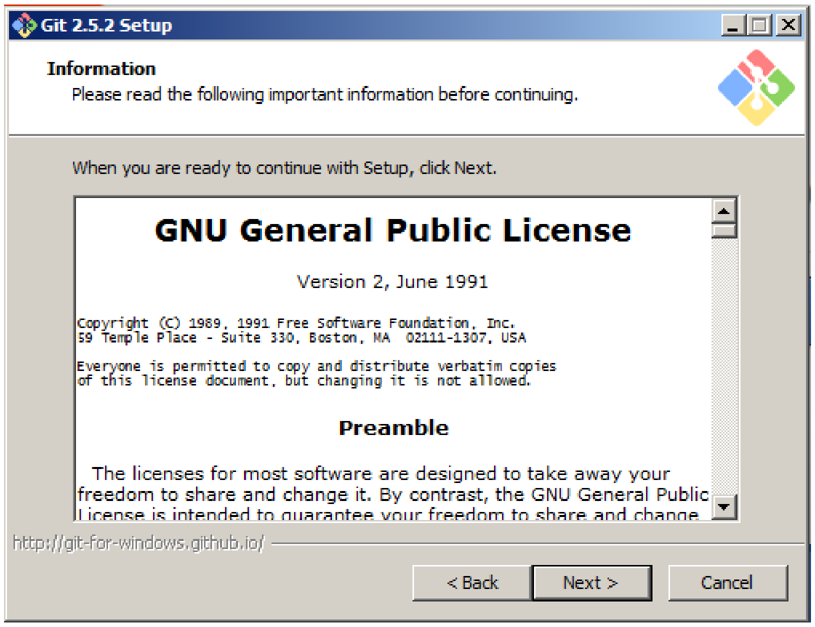
This will install Git version 2.5.2 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Click "Next."

Step 2. The next step is the GNU GPL for GitHub. You could take the time to read this... but in any case, just hit "Next."



Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

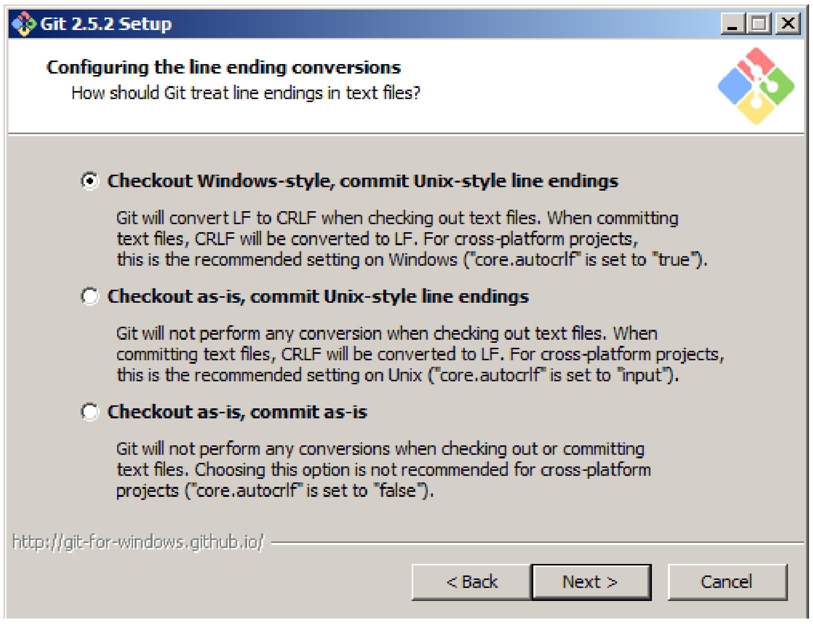
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

Step 3. Different operating systems represent 'newline' characters in different ways: it's why sometimes when you open code in Windows, everything is on a single line. Windows-style line endings are '\n\r' and Unix-style line endings are '\n'. For most cases, you want to select the 'Checkout Windows-style, commit Unix-style' option.



Configuring the line ending conversions

How should Git treat line endings 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").

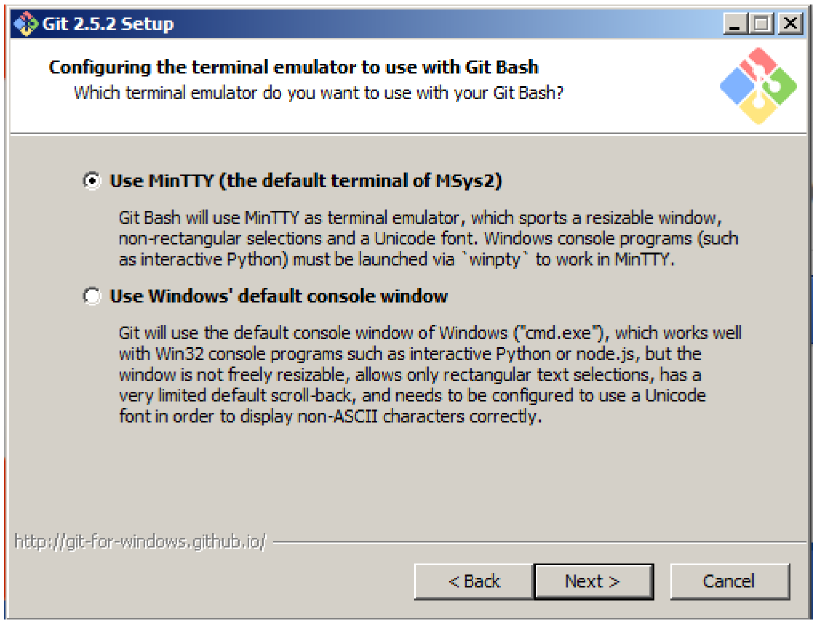
Checkout as-is, commit Unix-style line endings

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, commit 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").

Step 4. The Windows Command Prompt is known to be a pretty annoying interface. It's not freely resizable, it has very bad scrollback, and doesn't work well with a lot of things. Git Bash includes a version of TTY as its interface. Select 'MinTTY' if you don't want to deal with CMD when working with Git.



Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

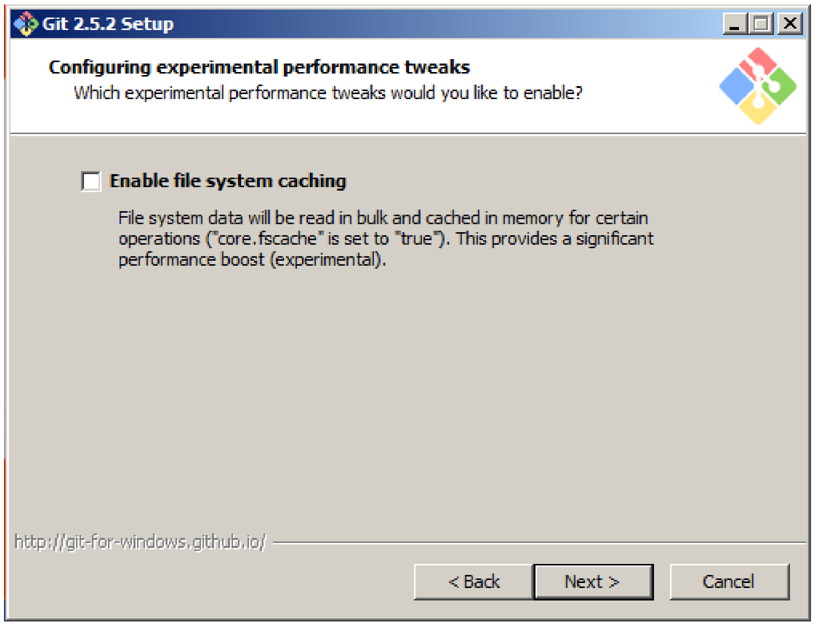
[✓ **Use MinTTY (the default terminal of MSys2)**]

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but the window is not freely resizable, allows only rectangular text selections, has a very limited default scroll-back, and needs to be configured to use a Unicode font in order to display non-ASCII characters correctly.

Step 5. The next step is experimental features. Use these at your own risk.



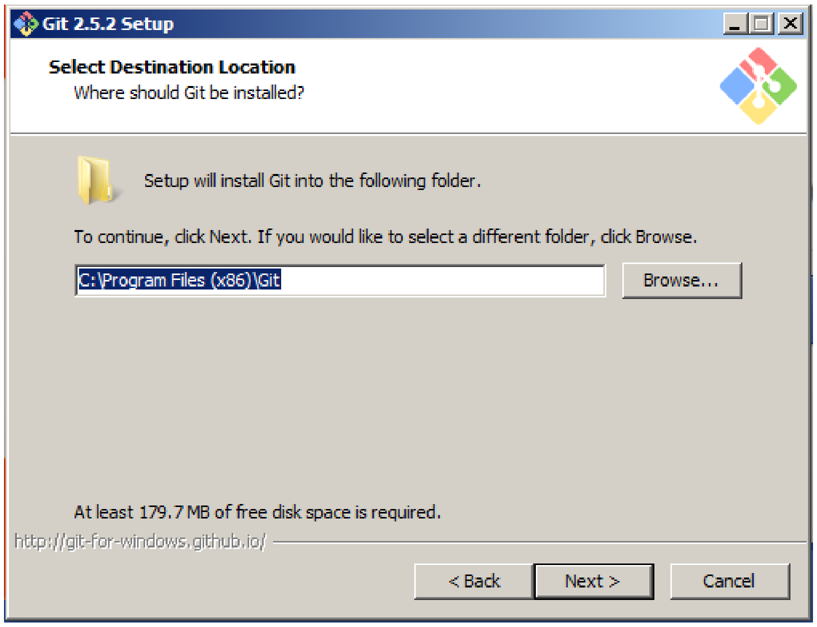
Configuring experimental performance tweaks

Which experimental performance tweaks would you like to enable?

☐ **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost (experimental).

Step 6. You can change the Git installation folder, but like with most Windows programs, it's best to leave the default value.


Git 2.5.2 Setup

Select Destination Location

Where should Git be installed?

Setup will install Git into the following folder.

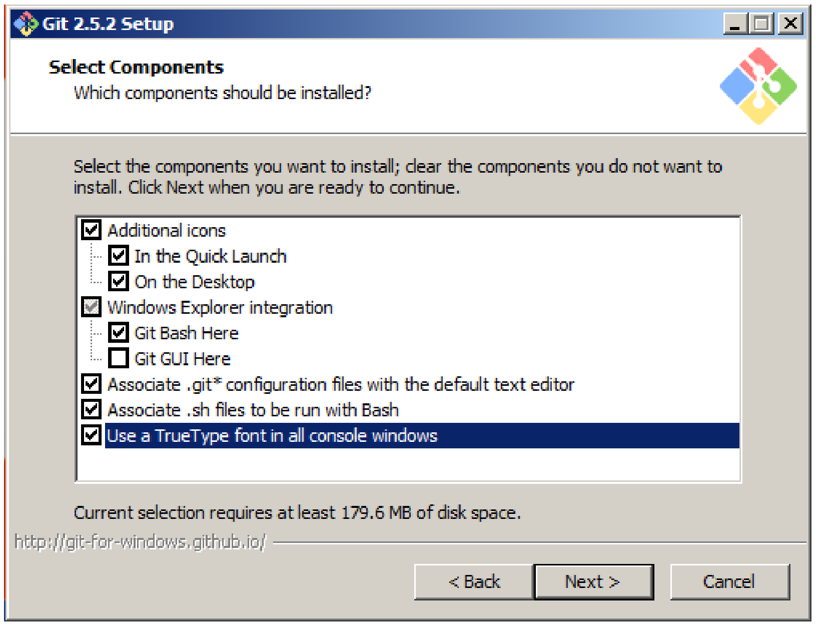
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files (x86)\Git

At least 179.7 MB of free disk space is required.

http://git-for-windows.github.io/

Step 7. Click/unclick the checkboxes to match the following:


Git 2.5.2 Setup

Select Components

Which components should be installed?

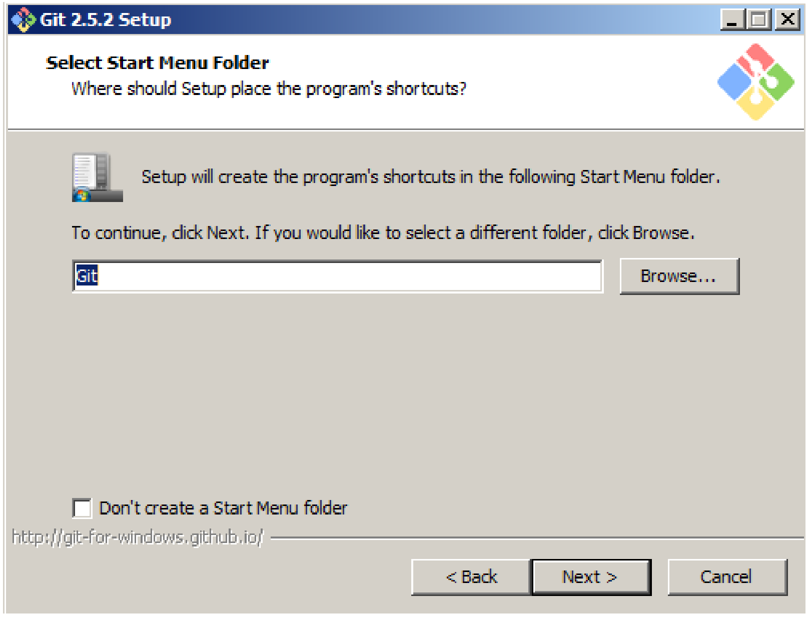
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] In the Quick Launch
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [ ] Git GUI Here
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [✓] Use a TrueType font in all console windows

Current selection requires at least 179.6 MB of disk space.

http://git-for-windows.github.io/

Step 8. Like in Step 6, just leave the default value.



Git 2.5.2 Setup

Select Start Menu Folder

Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

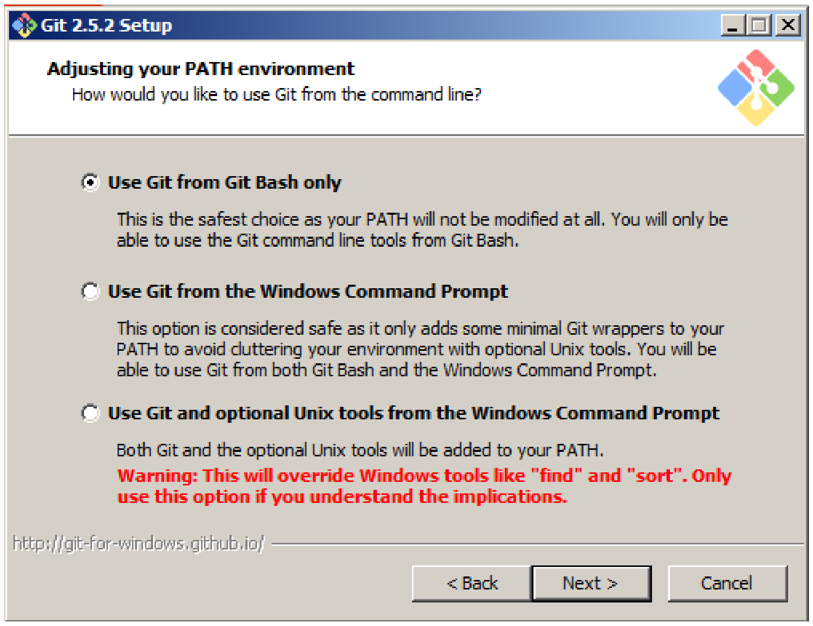
To continue, click Next. If you would like to select a different folder, click Browse.

Git

- [ ] Don't create a Start Menu folder

http://git-for-windows.github.io/

Step 9. You can leave this option on the 'Use Git from Git Bash only' option, or you can be risky and use one of the other two options. It is preferable that you only use Git from Git Bash to not alter the rest of the Windows system.



Git 2.5.2 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.

○ **Use Git from the Windows Command Prompt**

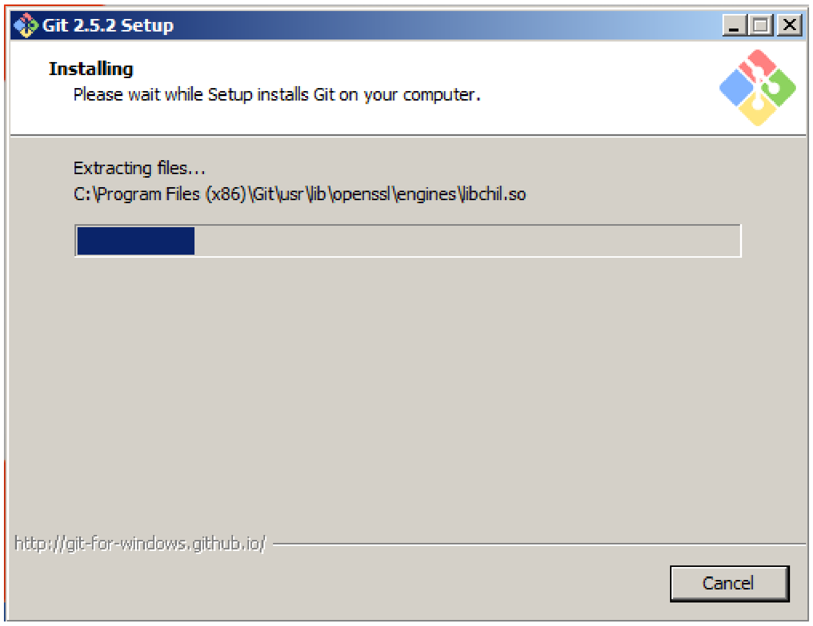
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

○ **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://git-for-windows.github.io/

Step 10. Wait a bit as the loading bar chugs on through. . .



Git 2.5.2 Setup

Installing

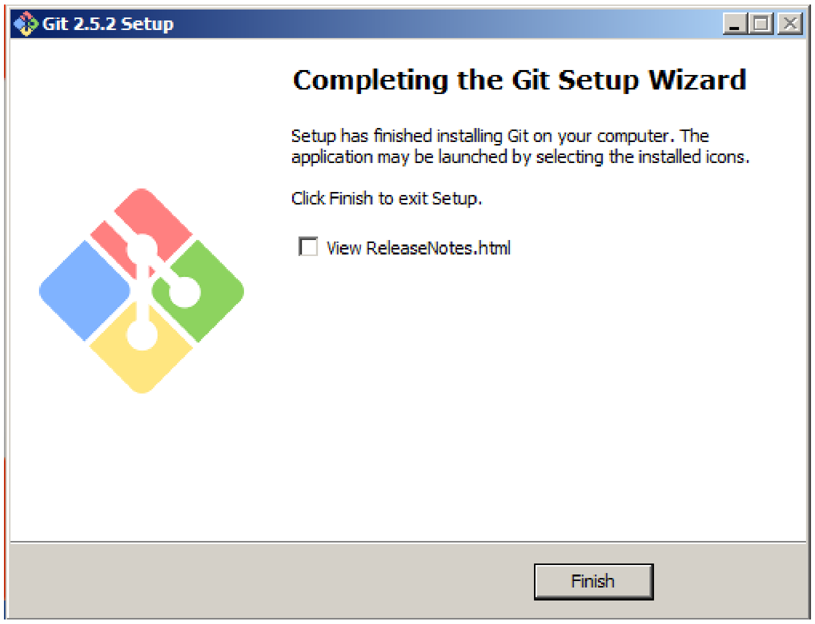
Please wait while Setup installs Git on your computer.

Extracting files...

C:\Program Files (x86)\Git\usr\lib\openssl\engines\libchil.so

http://git-for-windows.github.io/

Step 11. Uncheck the 'View ReleaseNotes.html' checkbox if you want. Release notes can be checked here: `https://github.com/git-for-windows/git/releases`



Git 2.5.2 Setup

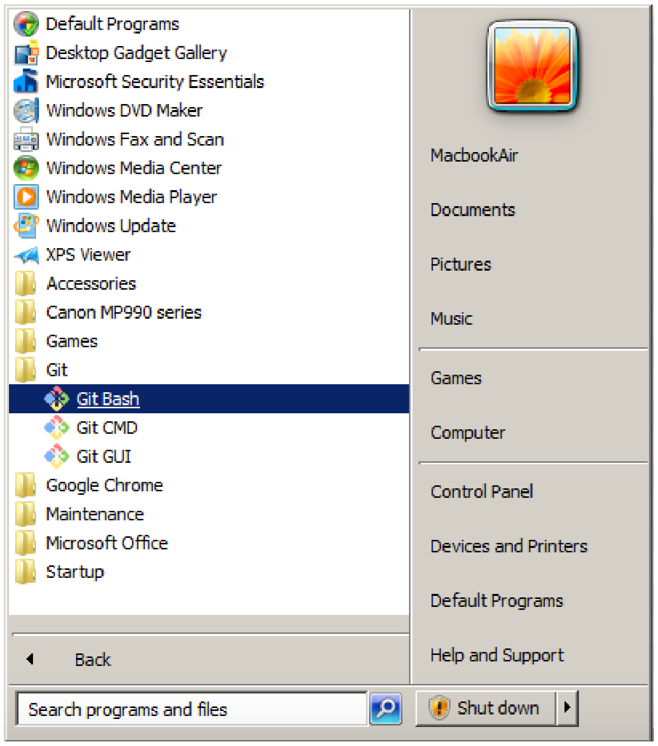
Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed icons.

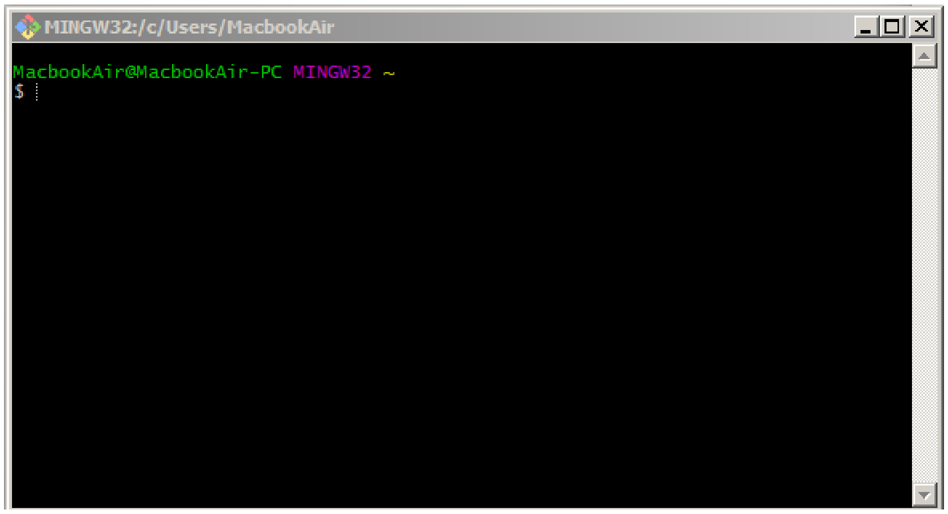
Click Finish to exit Setup.

☐ View ReleaseNotes.html

Step 12. You can open Git Bash through the Desktop shortcut, through the Start Menu, or through searching. Just open it.



Step 13. You should see Git Bash!



```
MacbookAir@MacbookAir-PC MINGW32 ~
$
```

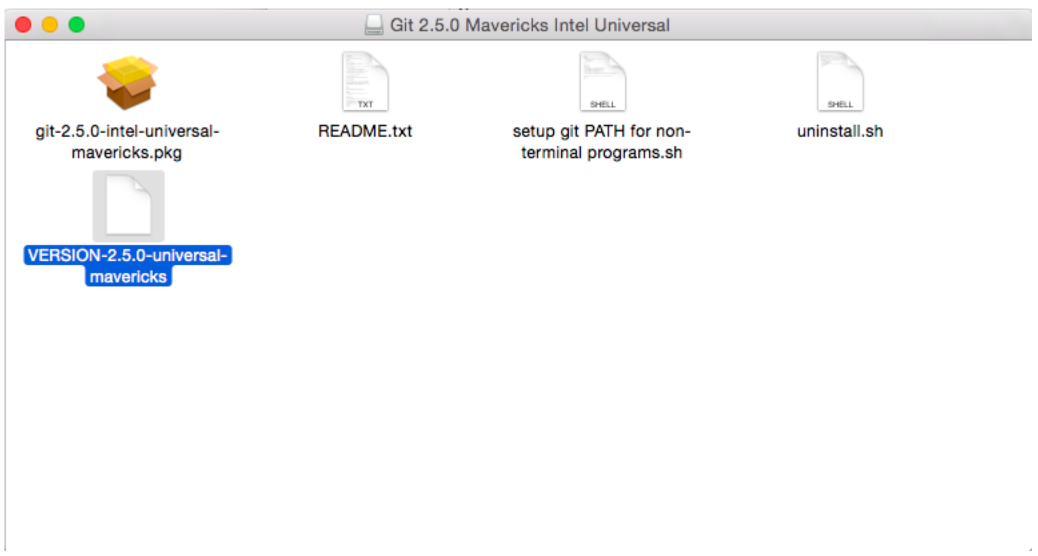
2.2 Mac OSX

There are a number of ways to get `git` on OSX. If you open Terminal (Terminal.app), you can type in `git`, and if it is not currently installed, it will prompt you to install it. However, XCode does not ship with the newest version of `git`, so you can do one of the following:

2.2.1 Method 1

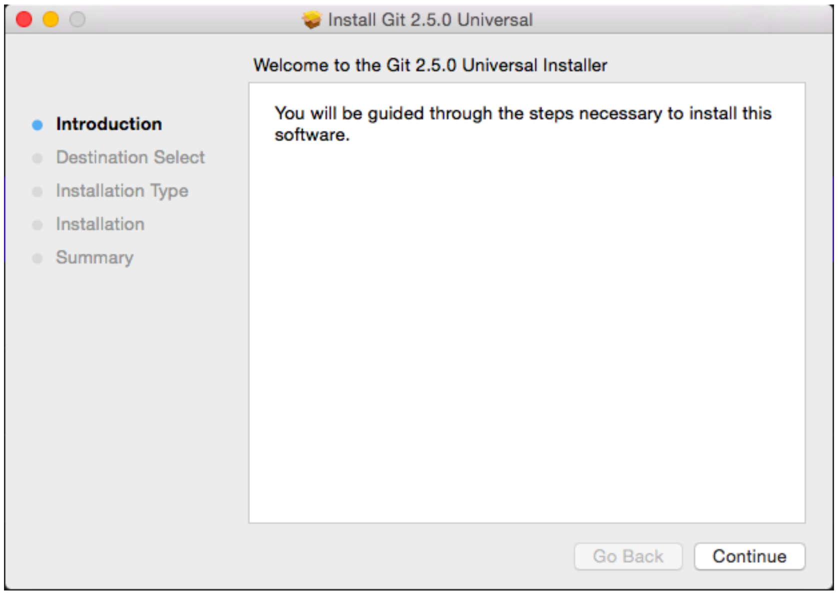
Using the GitHub provided installer here: `http://git-scm.com/download/mac`

Step 1. When you open the installer, you should see this:


Git 2.5.0 Mavericks Intel Universal
git-2.5.0-intel-universal-mavericks.pkg
README.txt
setup git PATH for non-terminal programs.sh
uninstall.sh
VERSION-2.5.0-universal-mavericks

Double click or open the `.pkg` file (the one that looks like an open box).

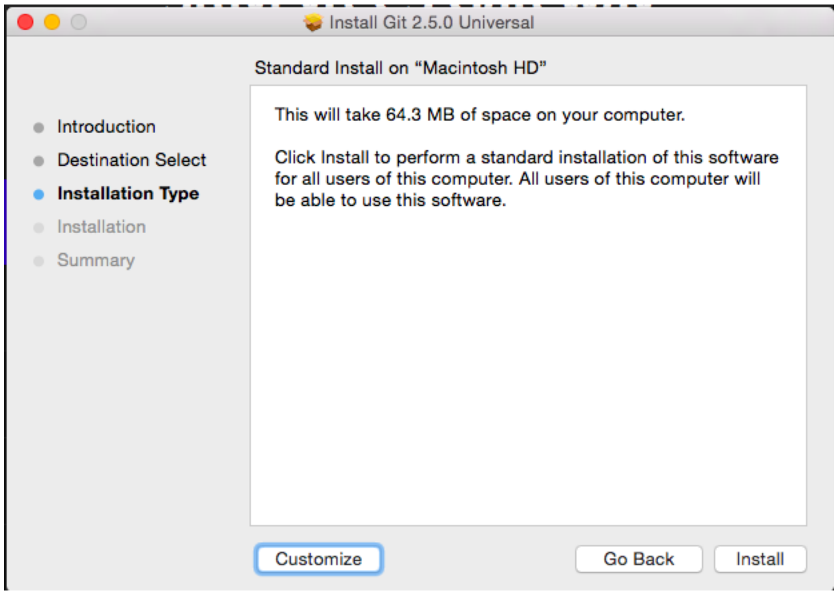
Step 2. The next step is the default OSX installer introduction. Just hit 'Continue.'



Welcome to the Git 2.5.0 Universal Installer

You will be guided through the steps necessary to install this software.

Step 3. You can hit 'Customize,' but just hit Install... You're done!



Standard Install on "Macintosh HD"

This will take 64.3 MB of space on your computer.

Click Install to perform a standard installation of this software for all users of this computer. All users of this computer will be able to use this software.

2.2.2 Method 2

This is the easy, command-line method using a neat command on Mac OSX: *Homebrew*. Just run the following commands to install Homebrew if you haven't already installed it:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then just run the command `brew install git` to install `git`!

```
$ brew install git
==> Downloading https://homebrew.bintray.com/bottles/git-2.5.1.yosemite.bottle.tar.gz
######################################################################## 100.0%
==> Pouring git-2.5.1.yosemite.bottle.tar.gz
==> Caveats
The OS X keychain credential helper has been installed to:
  /usr/local/bin/git-credential-osxkeychain

The "contrib" directory has been installed to:
  /usr/local/share/git-core/contrib

Bash completion has been installed to:
  /usr/local/etc/bash_completion.d

zsh completion has been installed to:
  /usr/local/share/zsh/site-functions
==> Summary
  /usr/local/Cellar/git/2.5.1: 1385 files, 32M
$ git --version
git version 2.3.2 (Apple Git-55)
```

2.3 Linux

Linux is perhaps the OS with the easiest installation process. Just open a terminal, and using your preferred package manager, install `git`.

```
$ yum install git
$ sudo apt-get install git
```

3 Local Usage

`Git` is a version tracking system for your coding projects. It was made so that a programmer can easily go through revision history, create multiple versions of their programs, and move between versions of their programs.

3.1 Creating a Git Repository

If you already have a project, you should `cd` into it. Otherwise, create a new folder for your project. For this tutorial, I will call my project `Git_Tutorial`. When you are inside your project folder, use the command `git init`.

```
$ mkdir Git_Tutorial
$ cd Git_Tutorial
$ git init
Initialized empty Git repository in Git_Tutorial/.git/
```

3.2 Basic Git Commands

Now that you have a brand new `git` project. Let's create some files and add them to the project. (The numbers at the right are for numbering the commands)

```
$ git status                                                       (1)
On branch master
nothing to commit, working directory clean
$ echo "A brand new git project" > README.txt                      (2)
$ ls                                                               (3)
README.txt
$ git status                                                       (4)
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

README.txt
```

```
nothing added to commit but untracked files present (use "git add" to track)
$ git add README.txt                                                    (5)
$ git status                                                            (6)
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

new file:   README.txt

$ git diff --cached                                                     (7)
diff --git a/README.txt b/README.txt
new file mode 100644
index 0000000..c8afd65
--- /dev/null
+++ b/README.txt
@@ -0,0 +1 @@
+A brand new git project
$ git config user.name "elc1798"                                        (8)
$ git config user.email "subliminalmau5@gmail.com"                      (9)
$ git commit -m "My first commit"                                       (10)
$ git status                                                            (11)
On branch master
nothing to commit, working directory clean
$ git log                                                               (12)
commit 58aace97366ac354d164ab5447c32bb42e0ca7bc
Author: elc1798 <subliminalmau5@gmail.com>
Date:   Sat Sep 12 17:35:06 2015 -0400

    My first commit
```

The above chunk of commands is a lot, so let's break it down.

Command 1) `git status` asks git about the current state of the project. The output of this command tells us the current branch of the project we are on (master) and the presence of edits to the current version of the project. If `git status` says ”Nothing to commit, working directory clean” then the current state of the project is the same as that of the latest version.

Command 2) `echo "A brand new git project" > README.txt` is not a git command. `echo` is supposed to print out whatever comes after it to the terminal, but the `>` redirects the output into the file `README.txt`. The command as a whole creates a new file called "README.txt" with the contents "A brand new git project".

Command 3) `ls` is a terminal command to (by default) list out the non-hidden contents of the current working directory.

Command 4) We run `git status` again after creating a new file, and our output is a bit different! It now shows the commit message of the current version: "Initial commit" and lists out the files that are currently not being tracked by `git`.a

Command 5) `git add <FILENAME>` will add a file to git's *staging area*. The staging area is a list of files with modifications that `git` will commit into the new version.

Command 6) Doing `git status` again after we have added "README.txt" to the staging area now gives us a "Changes to be committed" section, showing a "new file" named "README.txt".

Command 7) `git diff` will show us the differences between non-staged files and the state of files of the latest version of the project. `+` represents an added line, `-` represents a deleted line. Adding on the `--cached` flag (making the command `git diff --cached`) will show us the differences between staged files and the state of the files of the latest version of the project.

Command 8) `git config [flags] user.name "Your GitHub Username Goes in These Quotes"` will change the identity of the default committer of the current git project to the specified username. If the flag `--global` is used, this will change the name of the default committer of all git projects on the local computer.

Command 9) `git config [flags] user.email "Email you registered on GitHub with"` will change the default email address used for committing changes of the current project to the specified email. If the flag `--global` is used, this will change the default email of the default committer of all git projects on the local computer.

Command 10) `git commit -m "Commit Message"` will commit staged files and create a new version of the project. The `-m` or `--message` flag specifies the commit message. A good commit message should explain what was changed in the project.

Command 11) Since we just committed, `git status` will tell us that once again our working directory is clean.

Command 12) `git log` will show us a list of previous commits, as well as their commit messages.

3.3 Keeping Track of and Using Multiple Versions

3.3.1 Checking Out

Git can act like a store. The store contains all the files of your project that have been committed thus far. We can get a bunch of files from this ”store” and check them out at the cashier! Let’s look at an example from our Github_Tutorial project:

```
$ git status                                                 (1)
On branch master
nothing to commit, working directory clean
$ ls                                                         (2)
README.txt
$ cat README.txt                                             (3)
A brand new git project
$ echo "Modify file" > README.txt                            (4)
$ cat README.txt                                             (5)
Modify file
$ git status                                                 (6)
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   README.txt

no changes added to commit (use "git add" and/or "git commit -a")
$ git checkout README.txt                                    (7)
$ git status                                                 (8)
On branch master
nothing to commit, working directory clean
$ cat README.txt                                             (9)
A brand new git project
$ echo "Second commit" >> README.txt                         (10)
$ cat README.txt                                             (11)
A brand new git project
Second commit
$ git add README.txt                                         (12)
$ git commit -m "second commit"                              (13)
[master 57360d8] second commit
 1 file changed, 1 insertion(+)
```

```
$ git log --abbrev-commit                                    (14)
commit 57360d8
Author: elc1798 <subliminalmau5@gmail.com>
Date:   Sun Sep 20 14:55:37 2015 -0400

    second commit

commit 17e2103
Author: elc1798 <subliminalmau5@gmail.com>
Date:   Sun Sep 20 14:50:15 2015 -0400

    my first commit
$ git checkout 17e2103 README.txt                            (15)
$ cat README.txt                                             (16)
A brand new git project
$ git checkout HEAD README.txt                               (17)
$ cat README.txt                                             (18)
A brand new git project
Second commit
$ git status                                                 (19)
On branch master
nothing to commit, working directory clean
```

Let's explain this chunk of commands.

Command 1) Like always, we check the local status of our repository with `git status`. The output tells us there's nothing modified from the most recent commit.

Command 2) Just for demonstration purposes, `ls` tells us the only non-hidden file is `README.txt`

Command 3) We display the contents of `README.txt`, which says "A brand new git project" (from the previous section).

Command 4) We change the contents of `README.txt` to "Modify file"

Command 5) Verify the contents changed with `cat`

Command 6) `git status` tells us that `README.txt` is modified.

Command 7) `git checkout README.txt` is our 'checking out.' `git checkout <FILENAME>` will take a file called `<FILENAME>` from the most recent version of the project and put that into the current working directory.

Command 8) `git status` tells us that the local modifications have been overwritten and the working directory is clean.

Command 9) Verify the contents changed with `cat`

Command 10) We changed the contents of `README.txt` by adding the second line "Second commit."

Command 11) Verify the contents changed with `cat`

Command 12) We add the modified file to the trackers with `git add`.

Command 13) We commit the change with the message "second commit."

Command 14) `git log --abbrev-commit` shows us a list of previous commits, with each commit abbreviated.

Command 15) We once again 'check out' `README.txt` with `git checkout`, except this time we specify the commit, so now the local working directory's version of `README.txt` is the same as that in commit `17e2103`. By doing `git checkout <COMMIT> <FILENAME>`, the local file called `FILENAME` will be reverted to its version in `COMMIT`

Command 16) Verify the contents changed with `cat`

Command 17) We can also revert to the latest commit by using `git checkout HEAD README.txt`, we change the file to the most current commit.

Command 18) Verify the contents changed with `cat`

Command 19) We can now see that the working directory is up to date with `git status`.

3.3.2 Branches

A Git project is like a tree. There is a main trunk with the current public release, and there are different branches with other versions of the project. To see the options available for the `branch` feature of `git`, use the command: `git branch -h`
You can use the command `git branch -a` to list all the branches of a git project.

As usual, let's break down the chunk of commands above.