How to use Git in conjunction with Github

1. Create github account

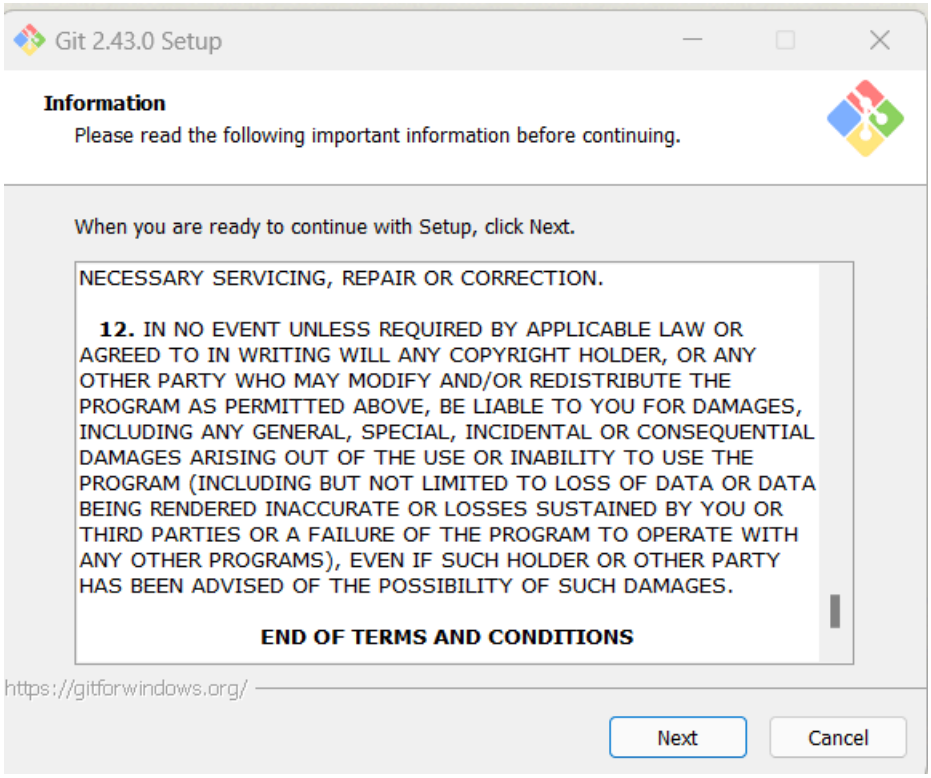
You will need a github account. After you create an account, it will be a good idea to eventually enable 2 factor authentication, but only do this from a computer you trust as you will be getting crypto keys that you will need to record / save somewhere and you don't want to do that on a squadron or public computer.

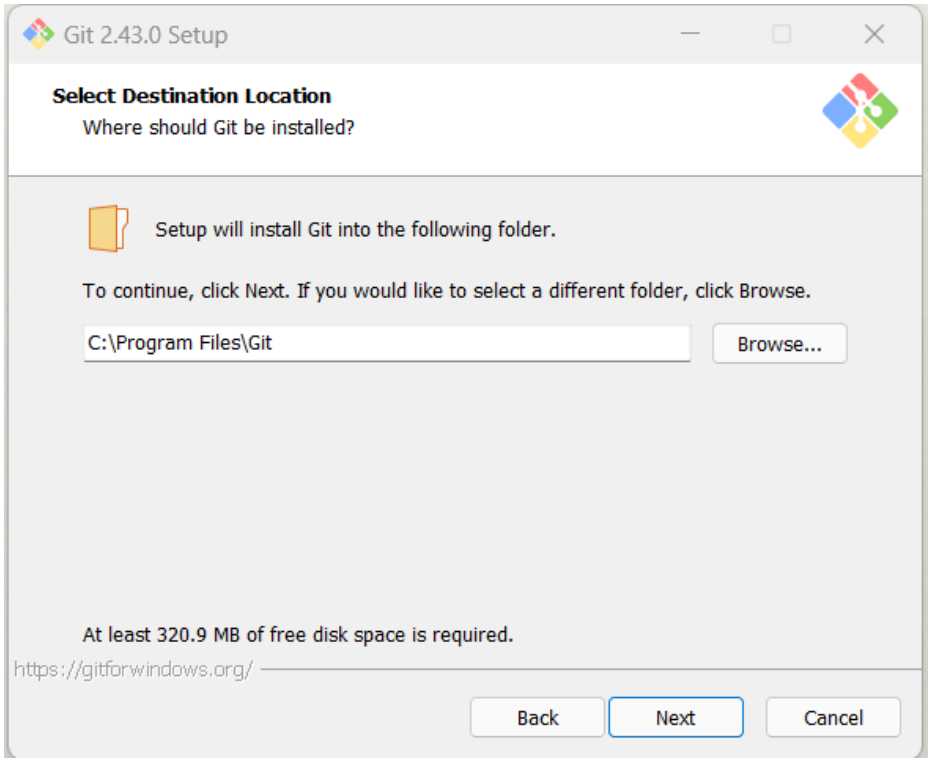
https://github.com/
https://github.com/signup

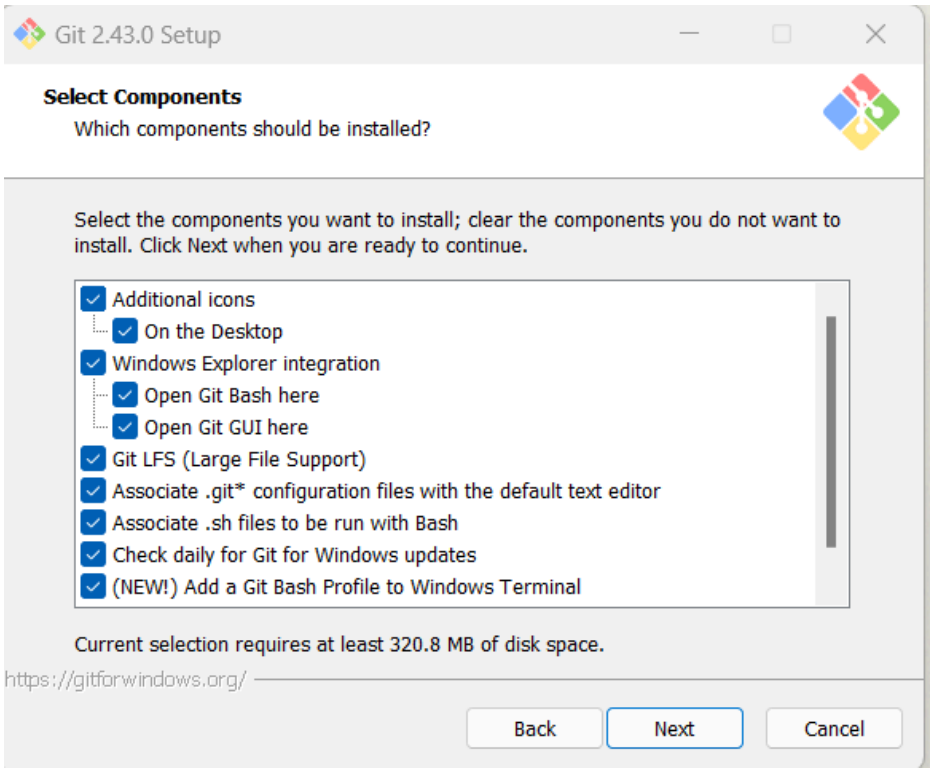
2. Install Git

On your local machine, you will need git installed. For Windows, Windows Git Bash works great.

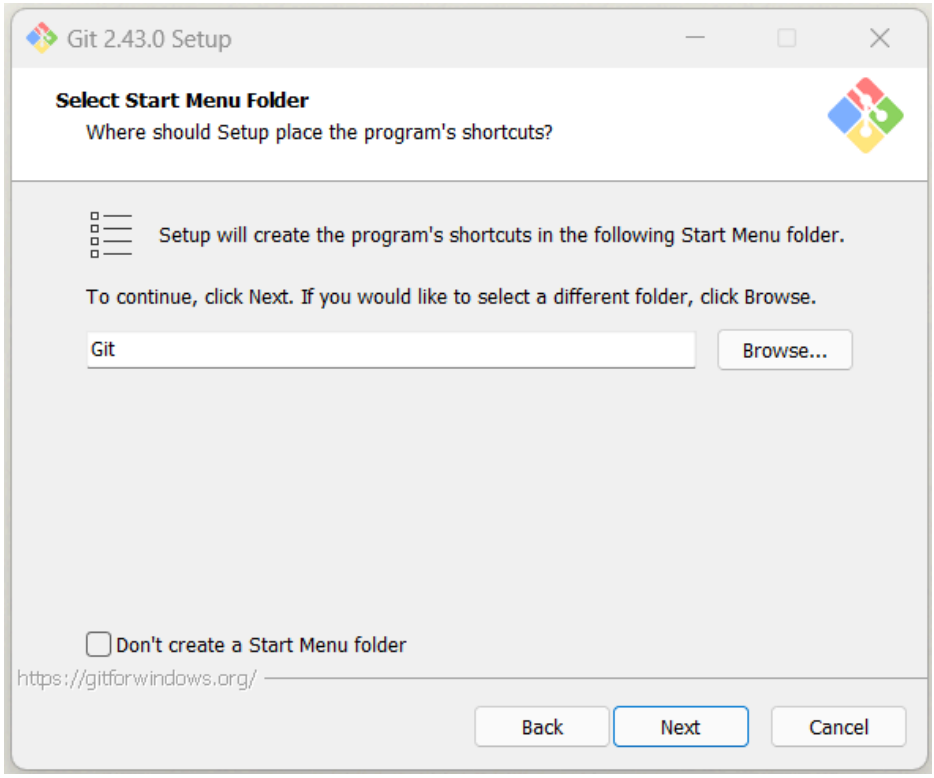
https://git-scm.com/downloads
https://git-scm.com/download/win

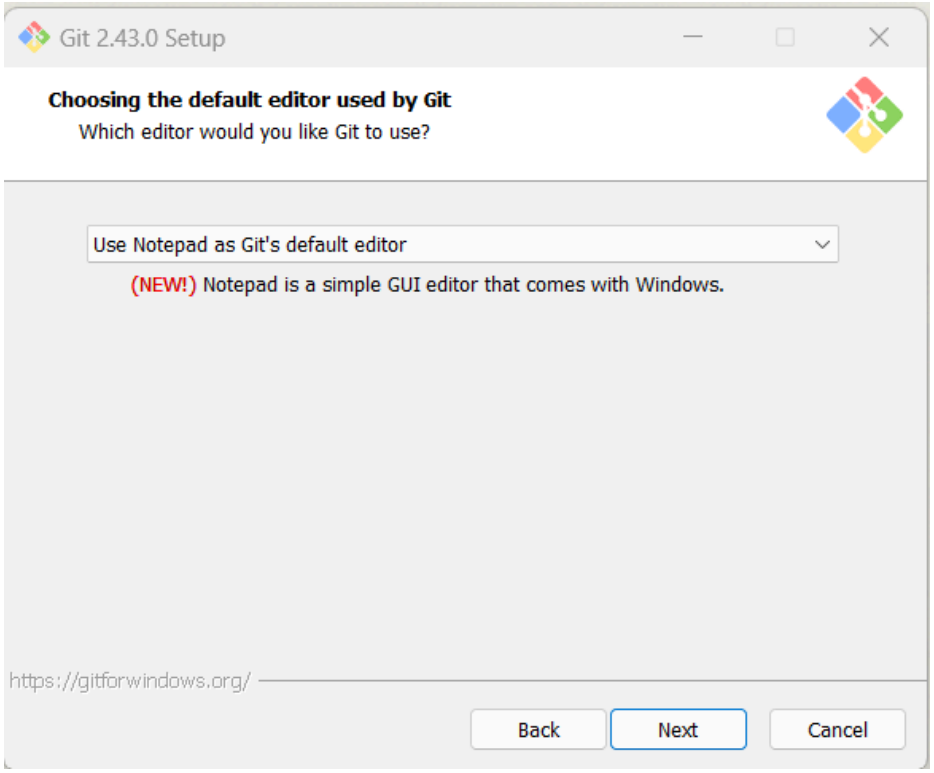


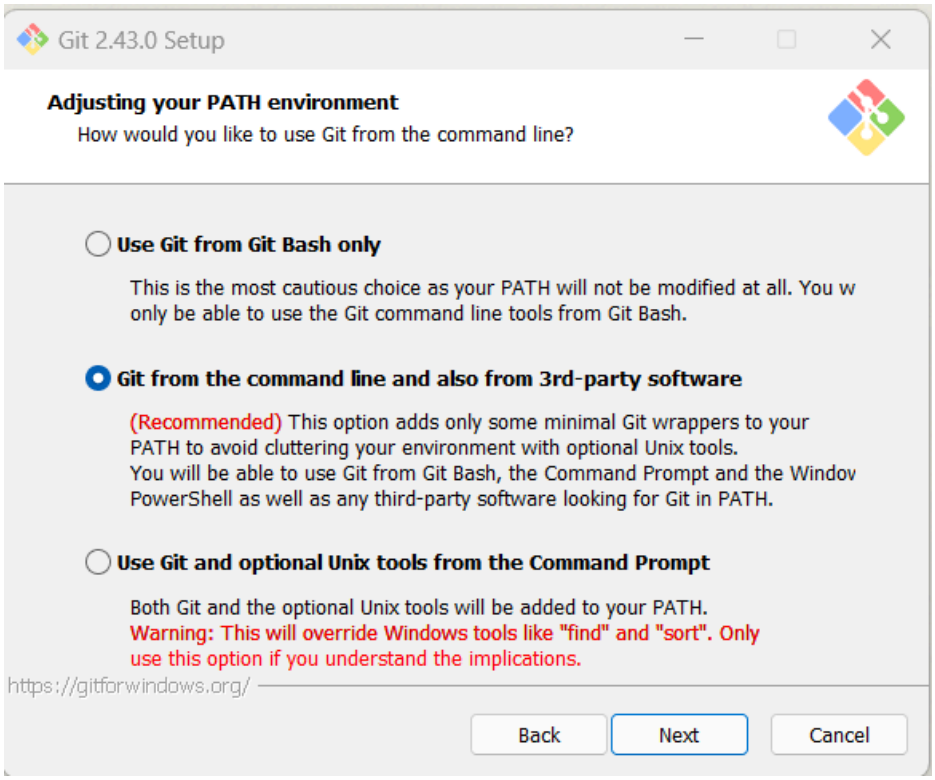


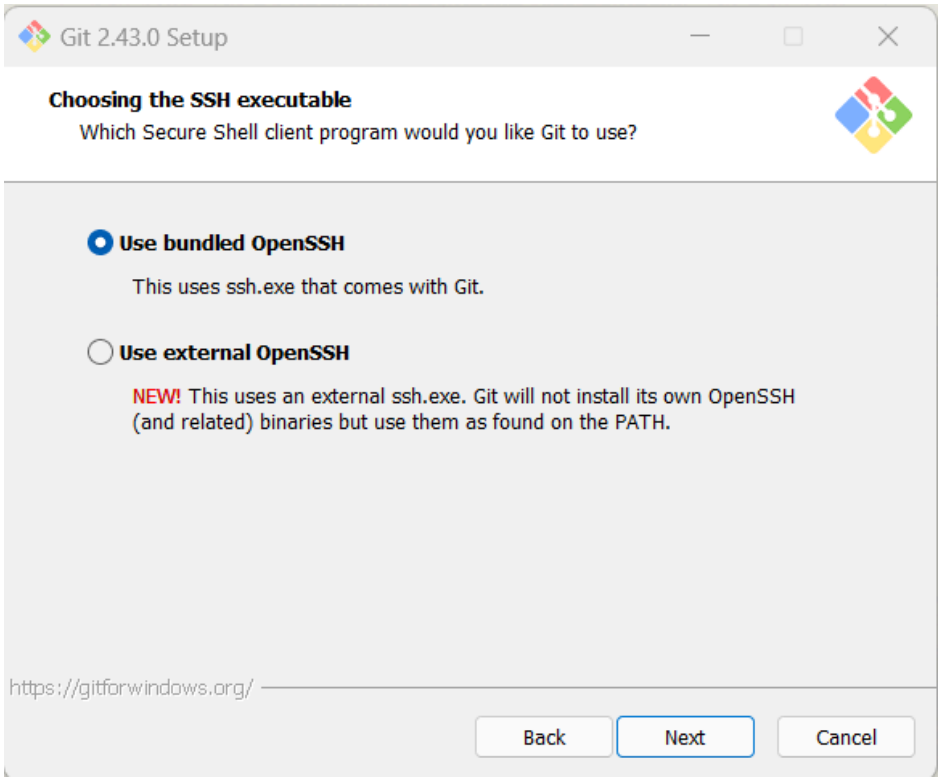


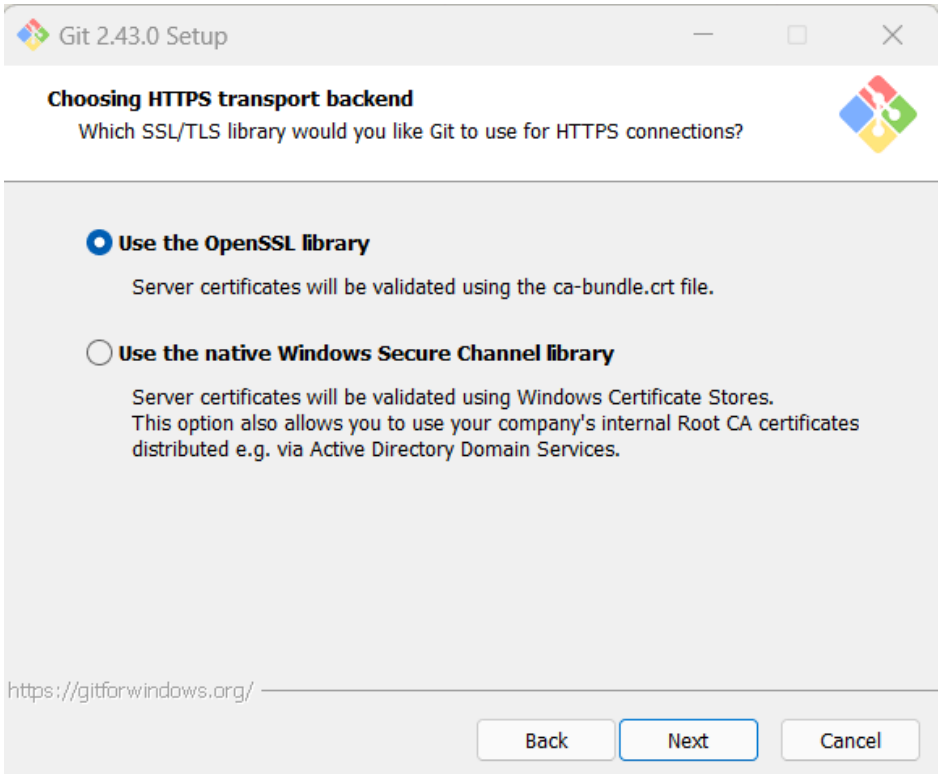
Don’t need the last one - Scalar

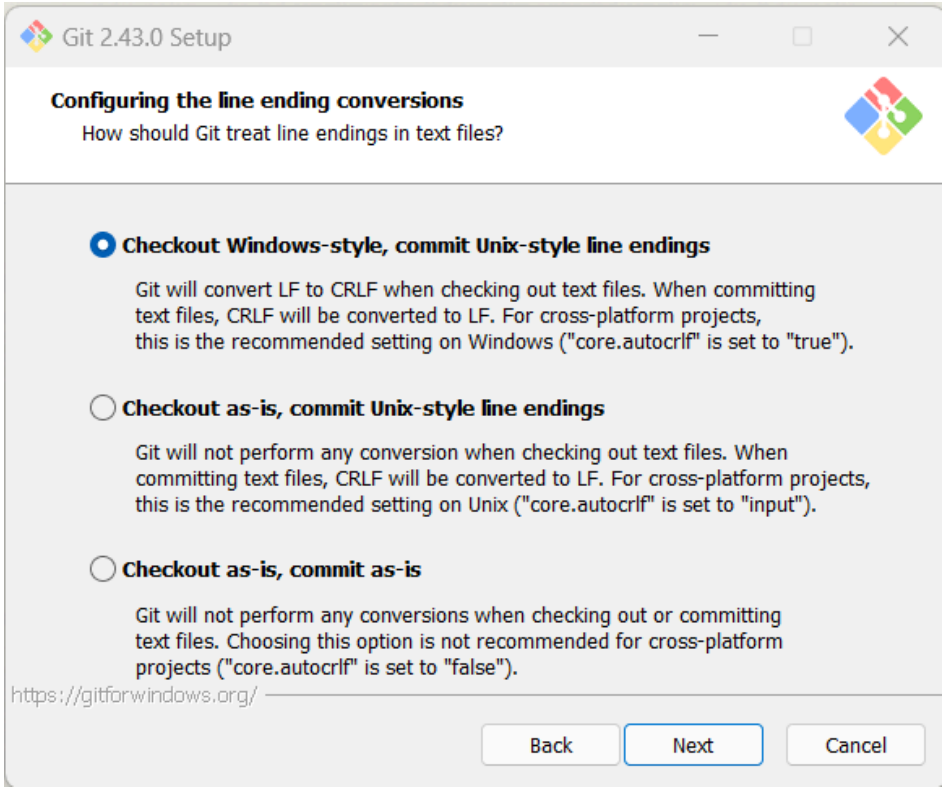

Git 2.43.0 Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Git
Browse...
Don't create a Start Menu folder
https://gitforwindows.org/
Back
Next
Cancel

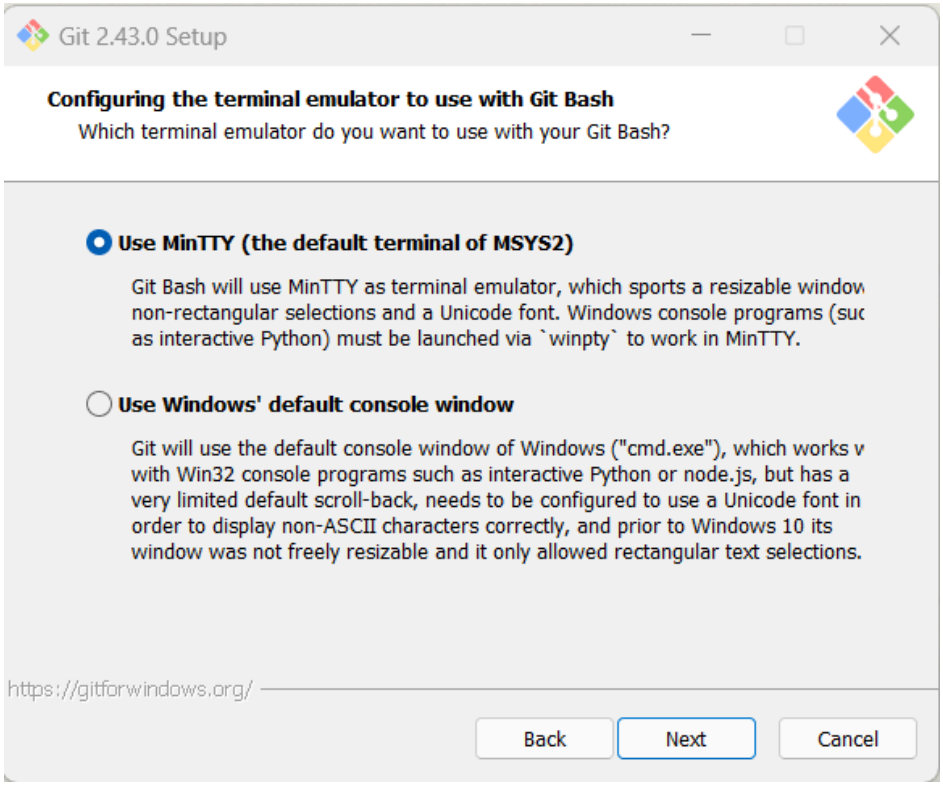

Git 2.43.0 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use Notepad as Git's default editor
(NEW!) Notepad is a simple GUI editor that comes with Windows.
https://gitforwindows.org/
Back
Next
Cancel

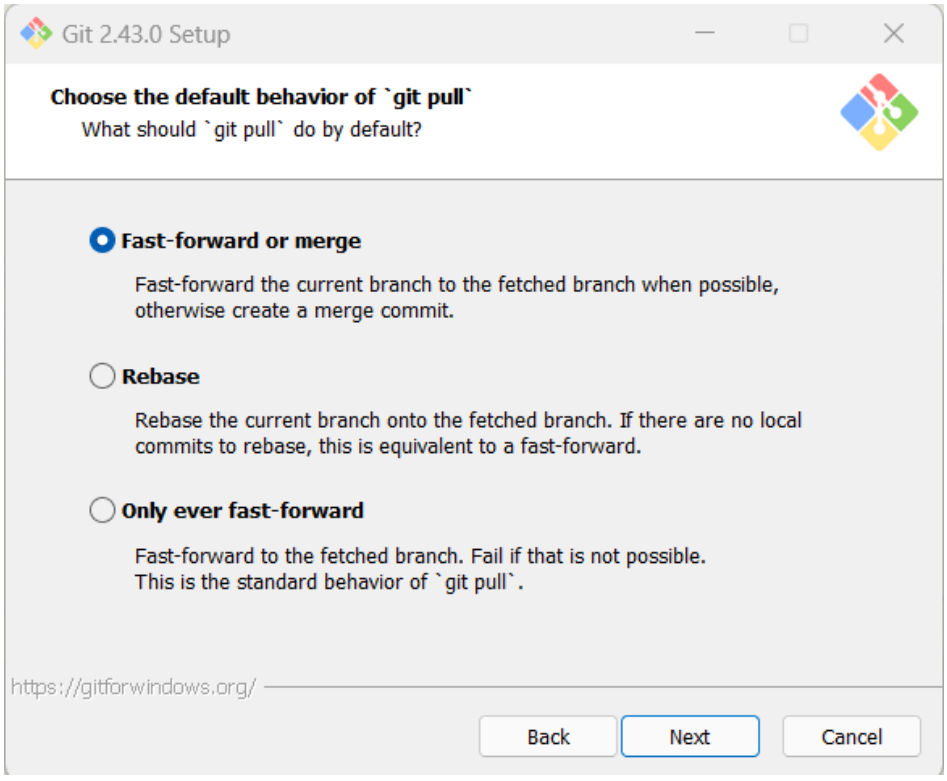

Git 2.43.0 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You w
only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your
PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windov
PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the 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.
https://gitforwindows.org/
Back
Next
Cancel

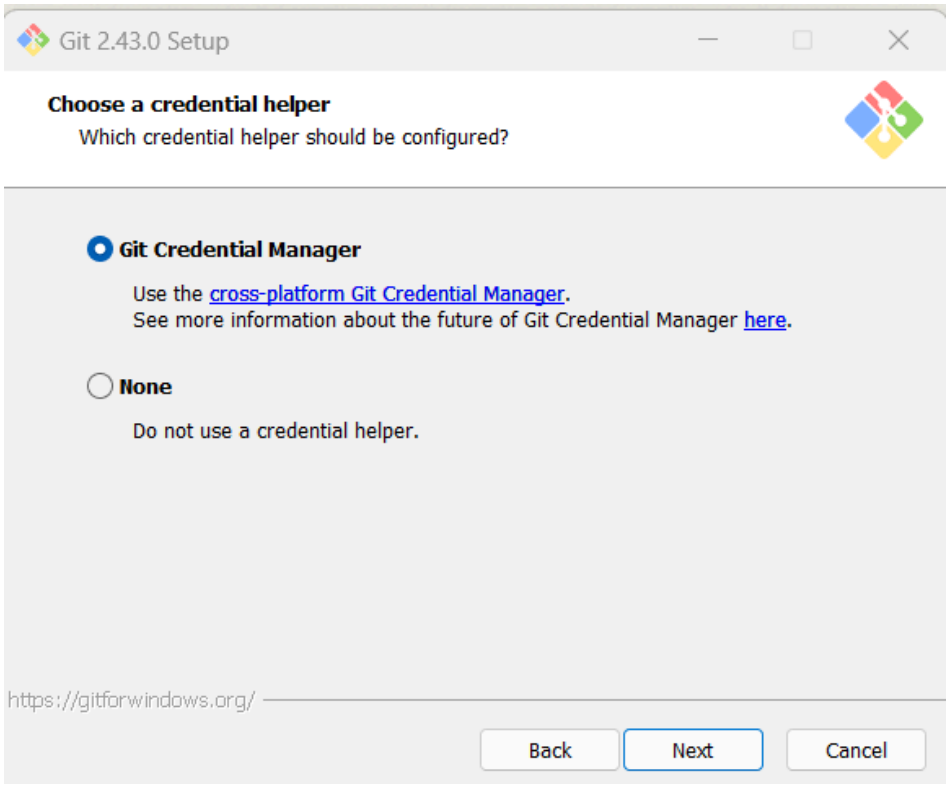

Git 2.43.0 Setup
Choosing the SSH executable
Which Secure Shell client program would you like Git to use?
Use bundled OpenSSH
This uses ssh.exe that comes with Git.
Use external OpenSSH
NEW! This uses an external ssh.exe. Git will not install its own OpenSSH
(and related) binaries but use them as found on the PATH.
https://gitforwindows.org/
Back
Next
Cancel

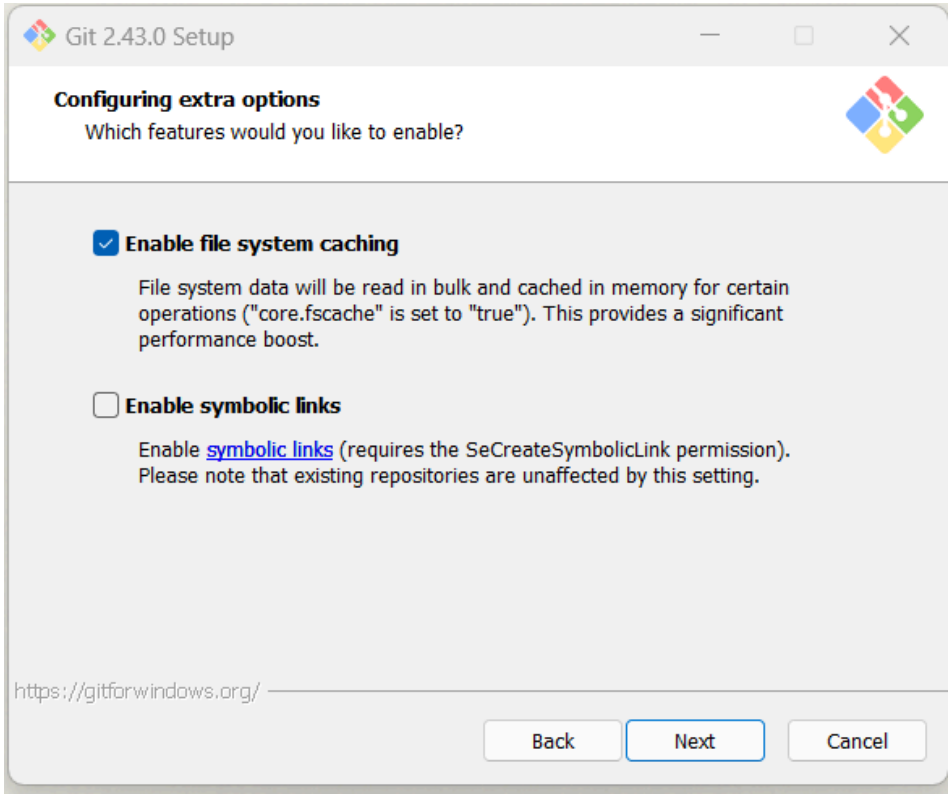

Git 2.43.0 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
Back
Next
Cancel

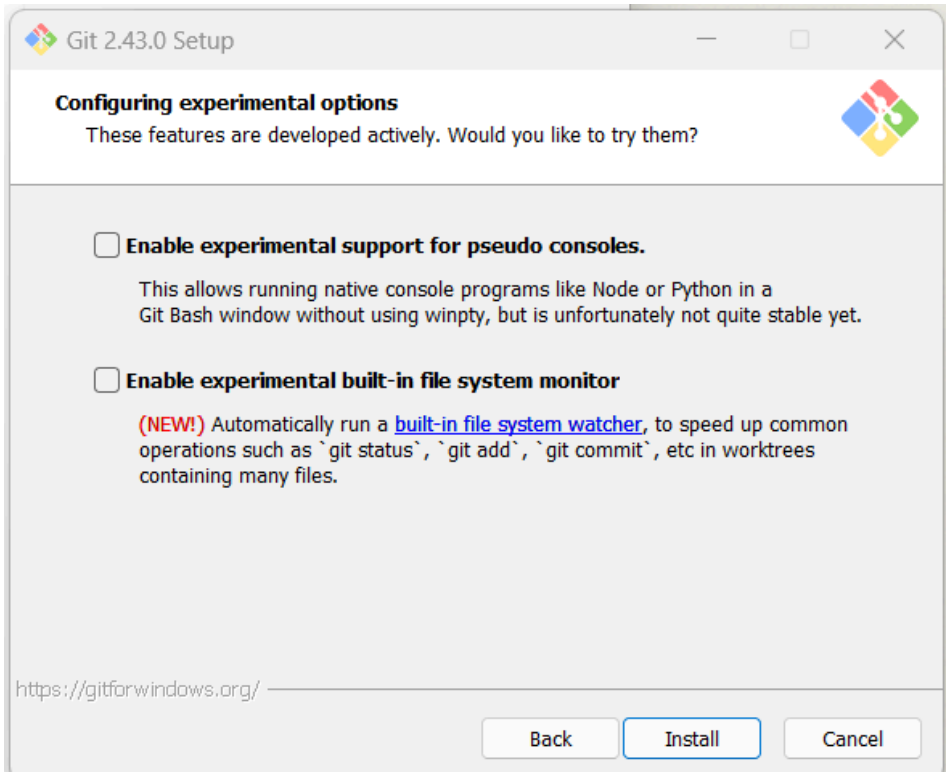

Git 2.43.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing
text files, CRLF will be converted to LF. For cross-platform projects,
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").
https://gitforwindows.org/
Back
Next
Cancel

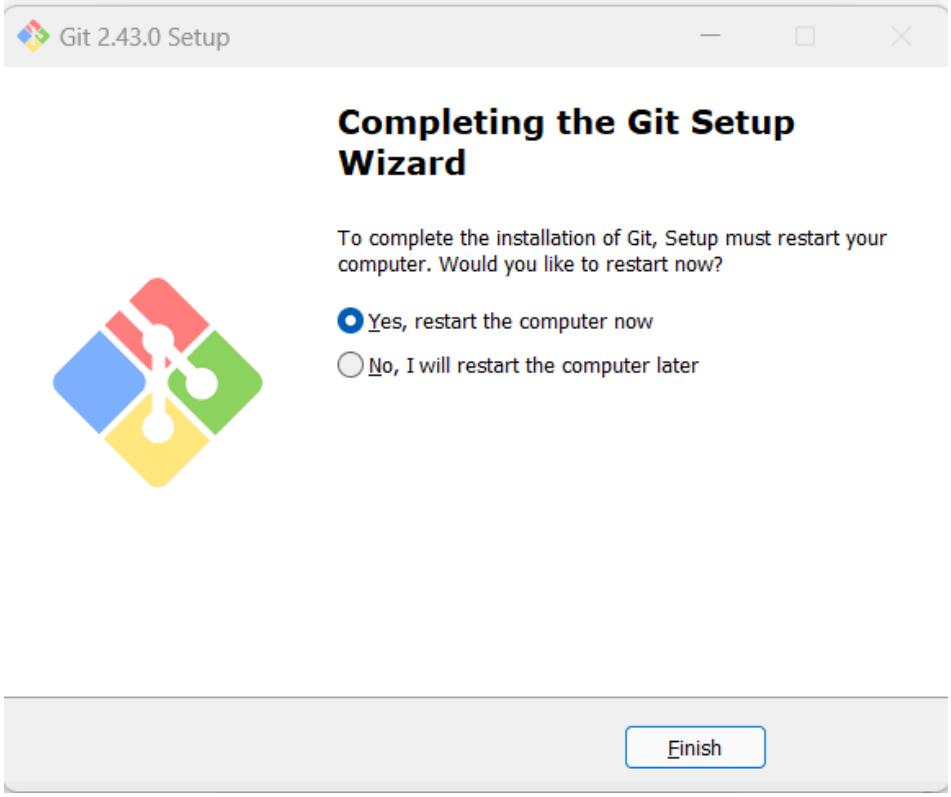

Git 2.43.0 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable windov
non-rectangular selections and a Unicode font. Windows console programs (suc
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 v
with Win32 console programs such as interactive Python or node.js, but has a
very limited default scroll-back, needs to be configured to use a Unicode font in
order to display non-ASCII characters correctly, and prior to Windows 10 its
window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.43.0 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Fast-forward or merge
Fast-forward the current branch to the fetched branch when possible,
otherwise create a merge commit.
Rebase
Rebase the current branch onto the fetched branch. If there are no local
commits to rebase, this is equivalent to a fast-forward.
Only ever fast-forward
Fast-forward to the fetched branch. Fail if that is not possible.
This is the standard behavior of `git pull`.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.43.0 Setup
Choose a credential helper
Which credential helper should be configured?
Git Credential Manager
Use the cross-platform Git Credential Manager.
See more information about the future of Git Credential Manager here.
None
Do not use a credential helper.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.43.0 Setup
Configuring extra options
Which features 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.
Enable symbolic links
Enable symbolic links (requires the SeCreateSymbolicLink permission).
Please note that existing repositories are unaffected by this setting.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.43.0 Setup
Configuring experimental options
These features are developed actively. Would you like to try them?
Enable experimental support for pseudo consoles.
This allows running native console programs like Node or Python in a Git Bash window without using winpty, but is unfortunately not quite stable yet.
Enable experimental built-in file system monitor
(NEW!) Automatically run a built-in file system watcher, to speed up common operations such as `git status`, `git add`, `git commit`, etc in worktrees containing many files.
https://gitforwindows.org/
Back
Install
Cancel


Git 2.43.0 Setup
Completing the Git Setup Wizard
To complete the installation of Git, Setup must restart your computer. Would you like to restart now?
Yes, restart the computer now
No, I will restart the computer later
Finish

3. Clone the repo and make your first branch

It is time to clone the repo and you may need to log into git now.
Open the Git Bash terminal - you will find it by typing **`Git Bash`** in the Windows search bar.

WARNING! Branch names, file names, folder names are case sensitive on the github repo structure. Windows is not case sensitive for any of those things. Switching a branch to a different cased branch on your Windows machine will take you where you thought you were going. But when you push origin head, it will get set up on a different branch in the cloud than the one you were working on. And you'll wonder where your code went...

Change directory to c:

```
$ cd c:
```

Looking to see if uas4stem folder exists:

```
$ ls
```

Make a new directory if it doesn't:

```
$ mkdir uas4stem
```

Change directories to that directory:

```
$ cd uas4stem
```

==========

Clone the repo:

```
$ git clone https://github.com/steve-snow/overlake-uas4stem
```

==========

Set github username and associated email, and password if asked:

```
$ git config --global user.name
```

If not correct, set it:

```
$ git config --global user.name <your username here>
```

verify:

```
$ git config user.name

$ git config --global user.email

$ git config --global user.email <your email here>
```

verify:

```
$ git config user.email
```

==========

Looking to see if `overlake-uas4stem` *folder exists:*

```
$ ls

$ cd overlake-uas4stem
```

Make sure everything is good:

```
$ git status
```

A good branch name pattern to follow here is (use ALL lowercase, NO spaces, if need middle initial or number to differentiate you may do so)

Pattern:

`lastname-firstinitial-notes`

Example:

`snow-s-notes`

Create a new branch using your last name and first initial (-b is the new branch):

`$ git checkout -b <enter a new branch name>`

becomes:

```
$ git checkout -b snow-s-notes
```

Now, when you make changes to the files in the repo, it's easy to push them up to the cloud. Add a file in the tracked folder - lastname-journal.txt.
Make a note in the file with today's date and what you worked on. Save the file.

Show the changes:

```
$ git status
```

Stage the changes: (note it is a capital A, not a lowercase a)

```
$ git add -A
```

Show the changes:

```
$ git status
```

Changes are now a different color!

Commit the changes:

```
$ git commit -m 'some meaningful message about what you changed'
```

Example:

```
$ git commit -m 'journal entry 2-4-2024'
```

Push changes to the cloud into your branch:

```
$ git push origin head
```

---------- IF YOU HAVE THE FOLLOWING ERROR: -----

```
warning: the `credential.authority` and `GCM_AUTHORITY` settings are deprecated.
warning: see https://aka.ms/gcm/authority for more information.
warning: the `credential.authority` and `GCM_AUTHORITY` settings are deprecated.
warning: see https://aka.ms/gcm/authority for more information.
warning: the `credential.authority` and `GCM_AUTHORITY` settings are deprecated.
warning: see https://aka.ms/gcm/authority for more information.
```

Your commit still got pushed.

You can make the error go away with the following steps. Commits are still unsigned / unverified, but they will go up.

WARNING!! If you are doing these on your parent's computer / login, make sure you have their permission! It may affect their work settings. Use your own login to Windows if possible and remove the –global if you need to share the computer with someone who already uses git for their work.

```
$ git config –global credential.helper manager
```

verify:

```
$ git config –global credential.helper
```

Returns `manager`

```
computerX /c/cap/overlake-uas4stem/journals (snow-changes)
$ git config --global --unset credential.github.com.authority

computerX /c/cap/overlake-uas4stem/journals (snow-changes)
$ git config --global credential.github.com.provider

computerX /c/cap/overlake-uas4stem/journals (snow-changes)
$ git config --global credential.github.com.provider github

computerX /c/cap/overlake-uas4stem/journals (snow-changes)
$ git config --global credential.github.com.provider
```

Returns `github`

```
computerX /c/cap/overlake-uas4stem/journals (snow-changes)
```

```
$ git config --global credential.github.com.allowWindowsAuth false

-----------------------------------------------
```

Now if you check the status

```
$ git status
```

You should get an indication there are no changes waiting to be committed. It is safe to switch branches when there are no changes left uncommitted.

Switch branches:

```
$ git checkout main
```

Look for the file you created. Is it there?

```
$ ls
```

Switch branches back:

```
$ git checkout <your-branch-name-here>
```

Alternately: Switch branches back:

```
$ git checkout -
```

Look for the file you created. Is it there?

```
$ ls
```

But what if I can't remember the name of my branch, or another branch I was working on a while back?

Let's get a list of branches that are on the local computer:

```
$ git branch
```

Let's get a list of branches that are on the cloud repository:

```
$ git branch -r
```

4. Update your branch when main changes

What do you do if the main branch gets updated? You get the latest version of main and merge it into your branch. Before you change branches, you should check to see if there are any changes you need to stage and commit, or discard before you switch branches. This is VERY IMPORTANT!! **Do not switch branches back to the main branch with pending changes.**

Show the changes:

```
$ git status
```

If there are changes - stage the changes: (note it is a capital A, not a lowercase a)

```
$ git add -A
```

Commit the changes:

```
$ git commit -m 'some meaningful message about what you changed'
```

Verify changes are all committed:

```
$ git status
```

Switch to main branch:

```
$ git checkout main
```

Pull any changes from the remote (cloud) main branch to local main branch:

```
$ git pull origin main
```

Verify nothing weird happened:

```
$ git status
```

If you see "...MERGING…" somewhere in the message, then something weird just happened. Talk to an experienced git user to untangle your repo before continuing. Otherwise, let's continue:

Switch to your branch:

```
$ git checkout -
```

Bring the new changes (commits) from main into your branch, so your branch has everything main branch has, plus any changes you made on your branch:

```
$ git merge main
```

At this point various things can happen. The easiest is that nothing new was added because your branch already has all the changes committed to main. If you were expecting changes, then this becomes a bit more complicated, but you can repeat the steps above again (starting with section 4) to see if you can get all the changes you thought you were supposed to get.

The next easiest is that the merge brought all the changes in and there were no conflicts. If you check the status again it will show no issues or pending files to stage / commit.

The third option gets a little trickier. If there is a MERGING tag line in your branch name now, there was a conflict. You will also get a conflict message in response to the merge command.

You will need to resolve the merge before finishing the commit and moving on.

Visual Studio git merge conflict resolution tool is a relatively easy one.

Visual Studio Code git merge conflict resolution tool is trickier to use.

You can avoid conflicts by only changing the files you are responsible for, but it's not a big deal if you do have a conflict on merge. **Conflict resolution is part of the fun of git.**

5. Resources

Now that you have a repo, you are ready to learn about how git branching, committing, merging work. Here is a tool that will help you practice:

https://learngitbranching.js.org/

Terminology

Repository / repo - a root folder that contains files and folders

Branch - one version of all the tracked files in the repository

Commit - a record of the changes to files being tracked in a repository

Pull request - a formal means to update a branch with your code - merge your branch with main branch so that your code is on the main branch

Issue - a tool to indicate a problem or a suggestion to some aspect of the repo

Blame - a way to connect an issue with a particular branch / commit / file that caused the issue

Tracked / untracked - files can be ignored so they are not tracked and this is set in the .gitignore file in the root folder

Feature - some type of functionality that is provided

Bug - when some feature or aspect of the program is not behaving as it was designed to

- Some bug reports are actually feature requests because the user is reporting the bug based on how they expect the program to function but it was not designed to do that.

Git Bermuda Triangle - where you merge code to the main branch through a pull request, and then on the subsequent pull request and merge to main, your code from the previous merge just disappears. Fear not, unlike the real Bermuda Triangle, it is relatively easy to get the code back, but the hard part is realizing that it disappeared in the first place.

We can talk about merging to the main branch another time. For now, just focus on being able to get updates to main, into your working branch (branch that you are working on currently).