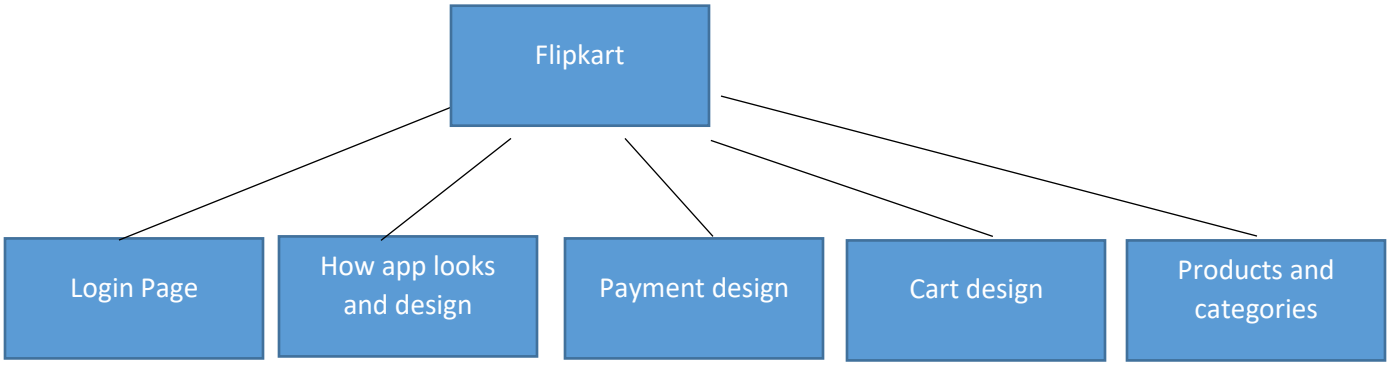
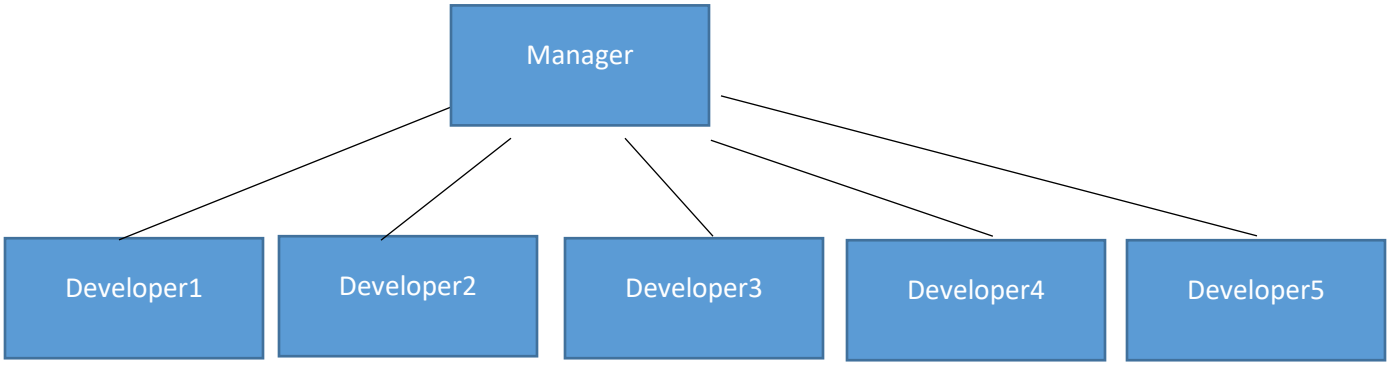
GIT Distributed Version Control System:

- **Software Configuration Management/Source code Management**





Versioning:- Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.

Why do we need Version Control System?

When multiple team members work on the same project, it is essential to have version control for the program. Version control system helps the team to share changes and merge changes made to artifacts seamless and efficient.

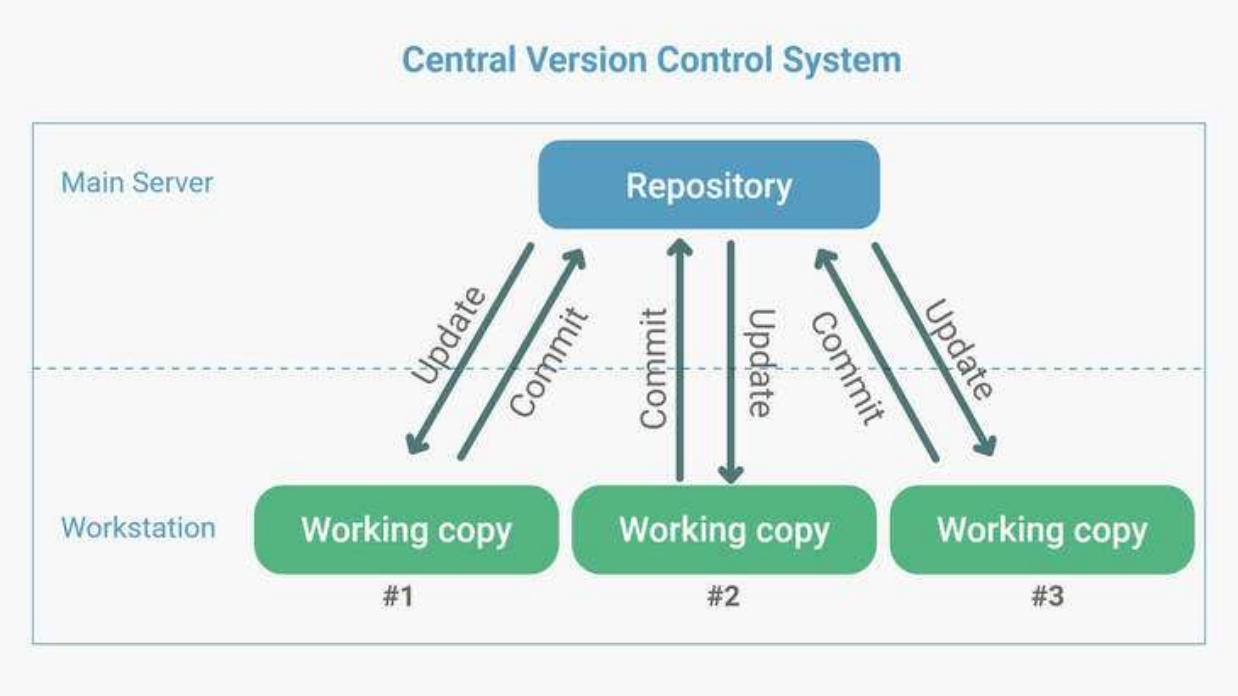
In DevOps, other than keeping track of changes, VCS also helps in developing and shipping the products faster.

VCS improves the following factors:

- Collaboration
- Storing Versions
- Backup
- Improves visibility
- Accelerate product delivery

Central Version Control System (CVCS)

In CVCS, the central server stores all the data. This central server enables team collaboration. It just contains a single repository, and each user gets their working copy. We need to commit, so the changes get reflected in the repository. Others can check our changes by updating their local copy.



Benefits of CVCS

- Easy to learn and manage
- Works well with binary files
- More control over users and their access.

CVS and SVN are some conventional Central Version Control systems.

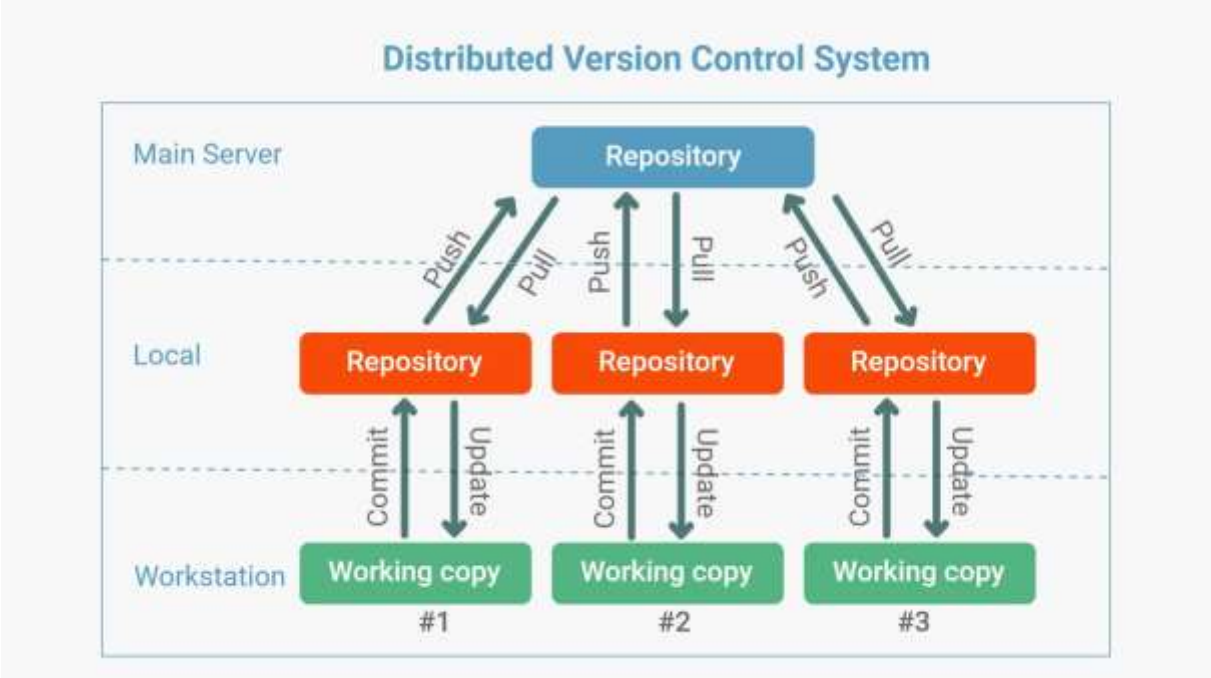
Drawbacks of CVCS

- It is not locally available, which means we must connect to the network to perform operations.
- During the operations, if the central server gets crashed, there is a high chance of losing the data.
- For every command, CVCS connects the central server which impacts speed of operation

The Distributed Version Control System was developed to overcome all these issues.

Distributed Version Control System (DVCS)

In DVCS, there is no need to store the entire data on our local repository. Instead, we can have a clone of the remote repository to the local. We can also have a full snapshot of the project history.



The User needs to update for the changes to be reflected in the local repository. Then the user can push the changes to the central repository. If other users want to check the changes, they will pull the updated central repository to their local repository, and then they update in their local copy.

Benefits of DVCS

- Except for pushing and pulling the code, the user can work offline in DVCS
- DVCS is fast compared to CVCS because you don't have to contact the central server for every command
- Merging and branching the changes in DVCS is very easy
- Performance of DVCS is better
- Even if the main server crashes, code will be stored in the local systems

Git and Mercurial are standard distributed version central systems. If we don't want a DVCS on our server, we can use either GitHub or BitBucket to store our central repository, and we can get the clone of the central repository to our local systems. GitHub and BitBucket are the most popular companies that provide cloud hosting for software development version control using Git.

DVCS is critical for DevOps because of the following reasons:

- Avoids dependency issues in modern containerized applications (Micro Services)
- Improves the performance of DevOps SDLC
- Supports in building more reliable applications

Git is a ***Distributed Version Control System***. So Git does not necessarily rely on a central server to store all the versions of a project's files. Instead, every user "clones" a copy of a repository (a collection of files) and has the ***full*** history of the project on their own hard drive. This clone has *all* of the metadata of the original while the original itself is stored on a self-hosted server or a third party hosting service like GitHub.

Git helps you ***keep track of the changes*** you make to your code.

Or you can simply see what changes you made to your code over time.

```
list1 = [1,2,3]
list1.append(4)
print(list1)
```

#initialFile

```
list1 = [1,2,3]
list1.append(4)
list1.append(5)
print(list1)
```

#addedALine

```
list1 = [1,2,3]
list1.pop()
print(list1)
```

#makingChanges

What is a Repository ?

A **repository** a.k.a. **repo** is nothing but a collection of source code.
A Kind of folder on server
A Kind of folder related to one product
Changes are personal to that particular repository

Server: It stores all repositories and contains metadata alsp

There are four fundamental elements in the Git Workflow.
Working Directory, **Staging Area**, **Local Repository** and **Remote Repository**.

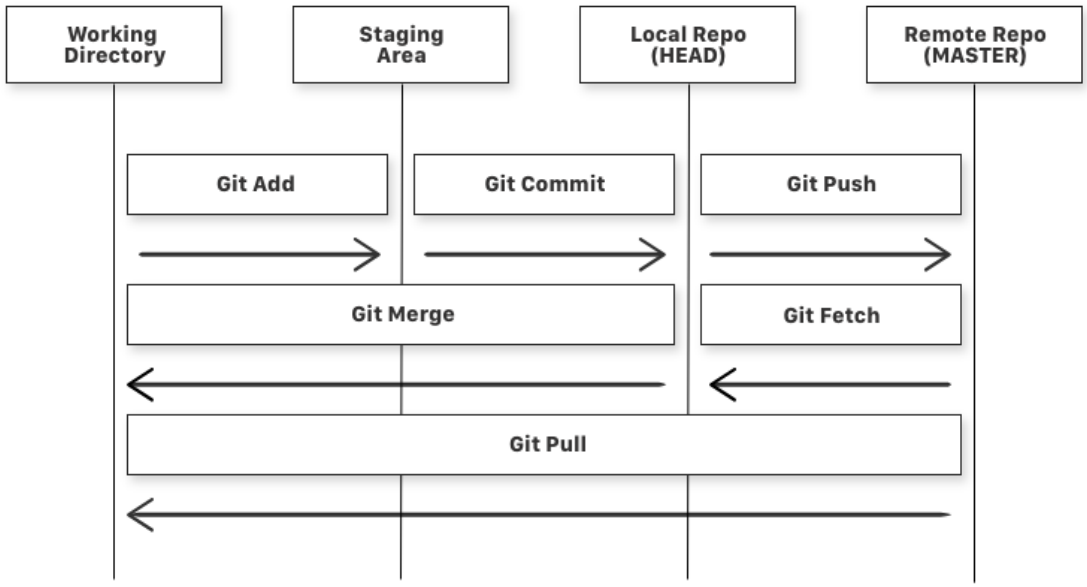
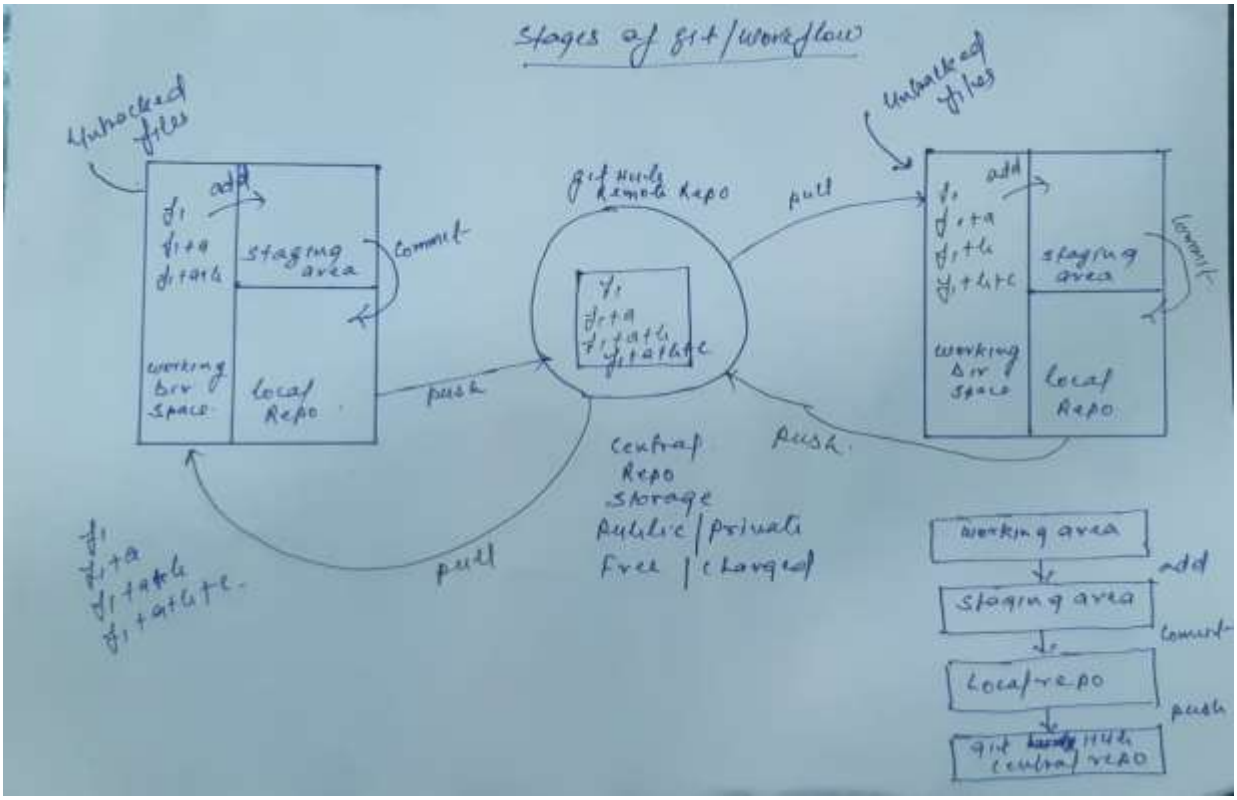


Diagram of a simple Git Workflow



If you consider a file in your Working Directory, it can be in three possible states.

1. **It can be staged.** Which means the files with the updated changes are marked to be committed to the local repository but not yet committed.
2. **It can be modified**. Which means the files with the updated changes are not yet stored in the local repository.
3. **It can be committed**. Which means that the changes you made to your file are safely stored in the local repository.

- **git add** is a command used to add a file that is in the working directory to the staging area.
- **git commit** is a command used to add all files that are staged to the local repository.
- **git push** is a command used to add all committed files in the local repository to the remote repository. So in the remote repository, all files and changes will be visible to anyone with access to the remote repository.
- **git fetch** is a command used to get files from the remote repository to the local repository but not into the working directory.

- **git merge** is a command used to get the files from the local repository into the working directory.
- **git pull** is command used to get files from the remote repository directly into the working directory. It is equivalent to a git fetch and a git merge .
- **Git is a software installed on a machine linux/windows local repository**
- **GitHub/GitLab is service on central Repository**

Repository
Repositories can be divided into two types based on the usage on a server. These are:

- **Bare Repositories:** These repositories are used to share the changes that are done by different developers. A user is not allowed to modify this repository or create a new version for this repository based on the modifications done.(Central repo)
- **Non-bare Repositories:** Non-bare repositories are user-friendly and hence allow the user to create new modifications of files and also create new versions for the repositories. Cloning process by default creates a non-bare repository if any parameter is not specified during the clone operation.(Local repo)

It is a place where we have all the codes.
Kind of folder related to one product on a server.
Changes are personal to that particular repository.

Server
It stores all the repository along with metadata.

Working directory/Area/tree/space
The Working Tree is the area where you are currently working. It is where your files live. This area is also known as the **"untracked" area of git**. Any changes to files will be marked and seen in the Working Tree.
Place where untracked files are present and you do all your modifications.
At a time you can work on a particular branch.

Staging Area
The primary function of the git add command, is to promote pending changes in the working directory, to the git staging area.

The Staging is like a rough draft space, it's where you can **git add** the version of a file or multiple files that you want to save in your next **commit** (in other words in the next version of your project).

Local repositories

Storage area where local committed file are present.

Commit

Stores changes in repository and will have a commit id(40 alphanumeric character.), SHA-1 Checksum, Helps in tracking the changes

Commit_id

Reference to identify each changes/To identify who made the changes

Tags

Meaningful name to a specific version in a repository

Snapshot

Represents some data at particular time

It is incremental and stores changes only i.e appended data not entire data

Synchronizing with Remote Repositories

Git allows the users to perform operations on the Repositories by cloning them on the local machine. This will result in the creation of various different copies of the project. These copies are stored on the local machine and hence, the users will not be able to sync their changes with other developers. To overcome this problem, Git allows performing syncing of these local repositories with the remote repositories.
This synchronization can be done by the use of two commands in the Git. These commands are:

- push
- pull

Push: This command is used to push all the commits of the current repository to the tracked remote repository. This command can be used to push your repository to multiple repositories at once.
Syntax:
$ git push -u origin master

To push all the contents of our local repository that belong to the master branch to the server(Global repository).

Pull: Pull command is used to fetch the commits from a remote repository and stores them in the remote branches. There **Syntax:**
$ git pull

Branch(Default Master)

Branching is required if one needs to experiment then we create a branch and all the code will be in that branch.

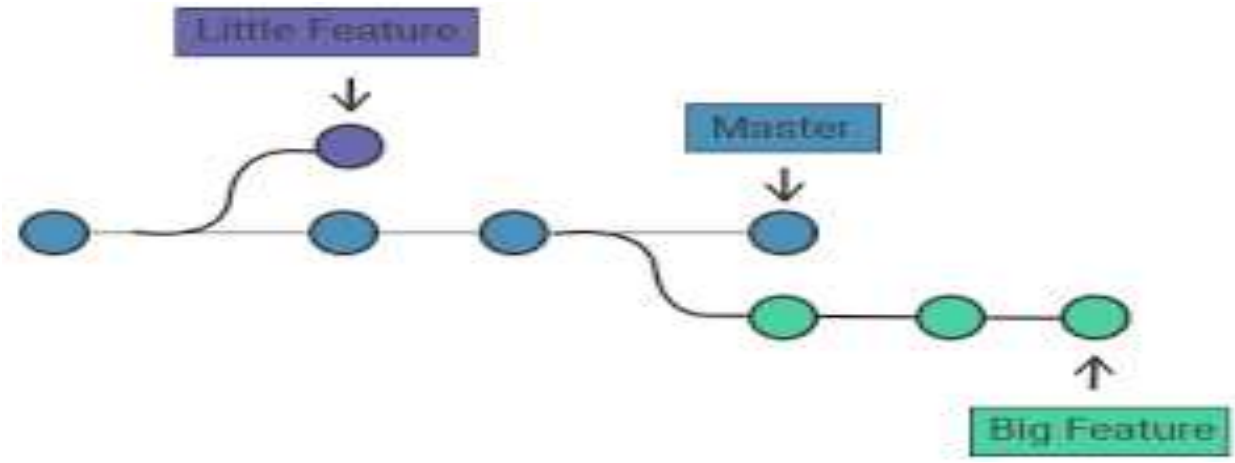
Product is same but different task

Each task has a separate branch/parallel working on different code

Finally merges (code) of all branches

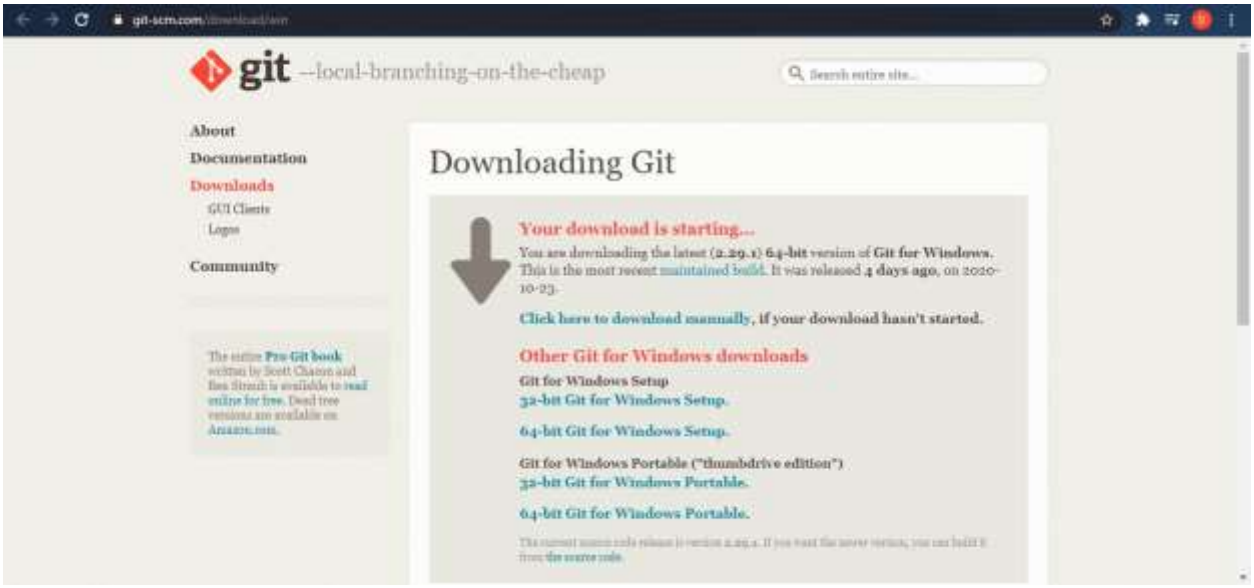
Changes are personal to that particular branch

File created in workspace will be visible in any of the branche workspace until you commit them.



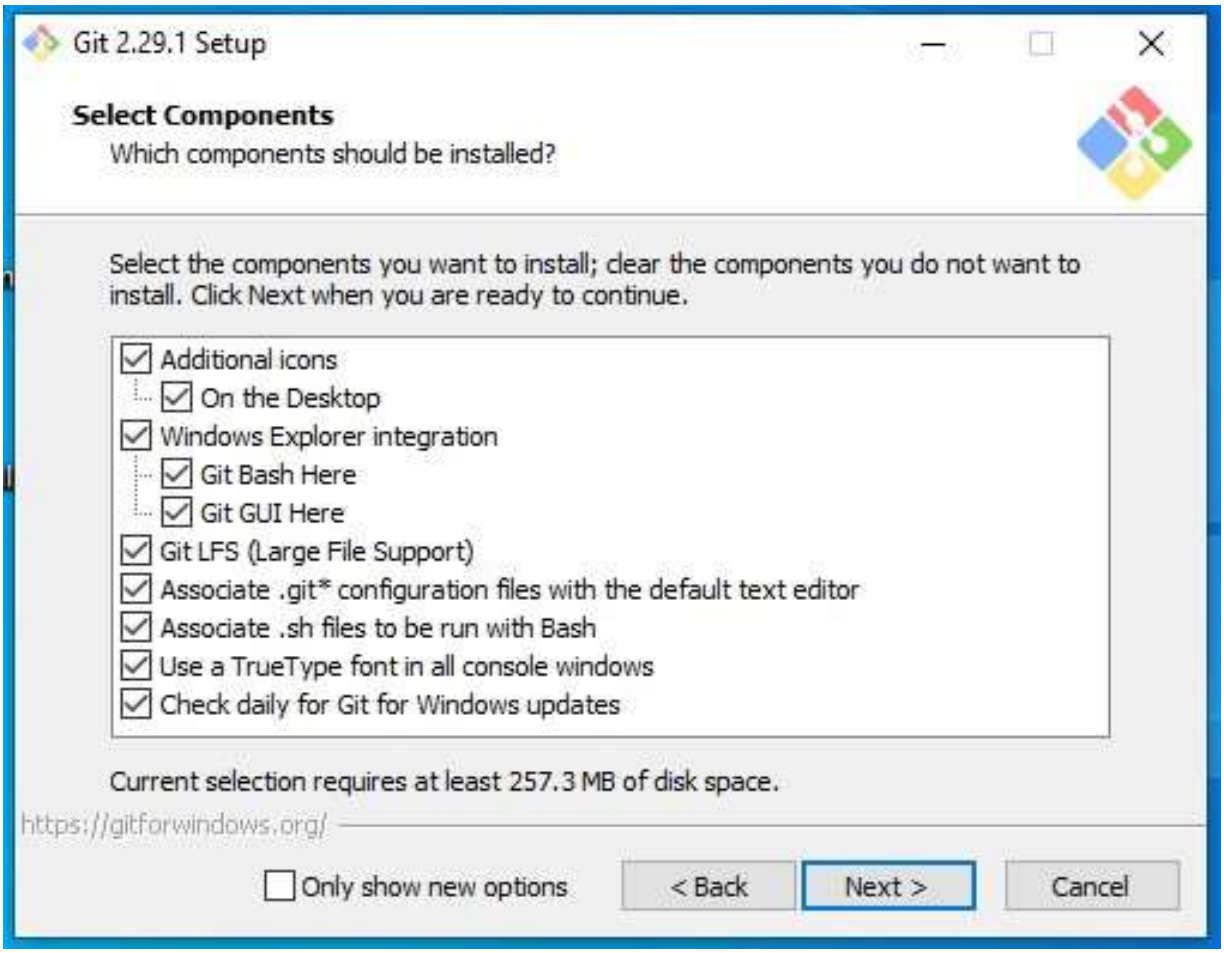
Installing Git For Windows

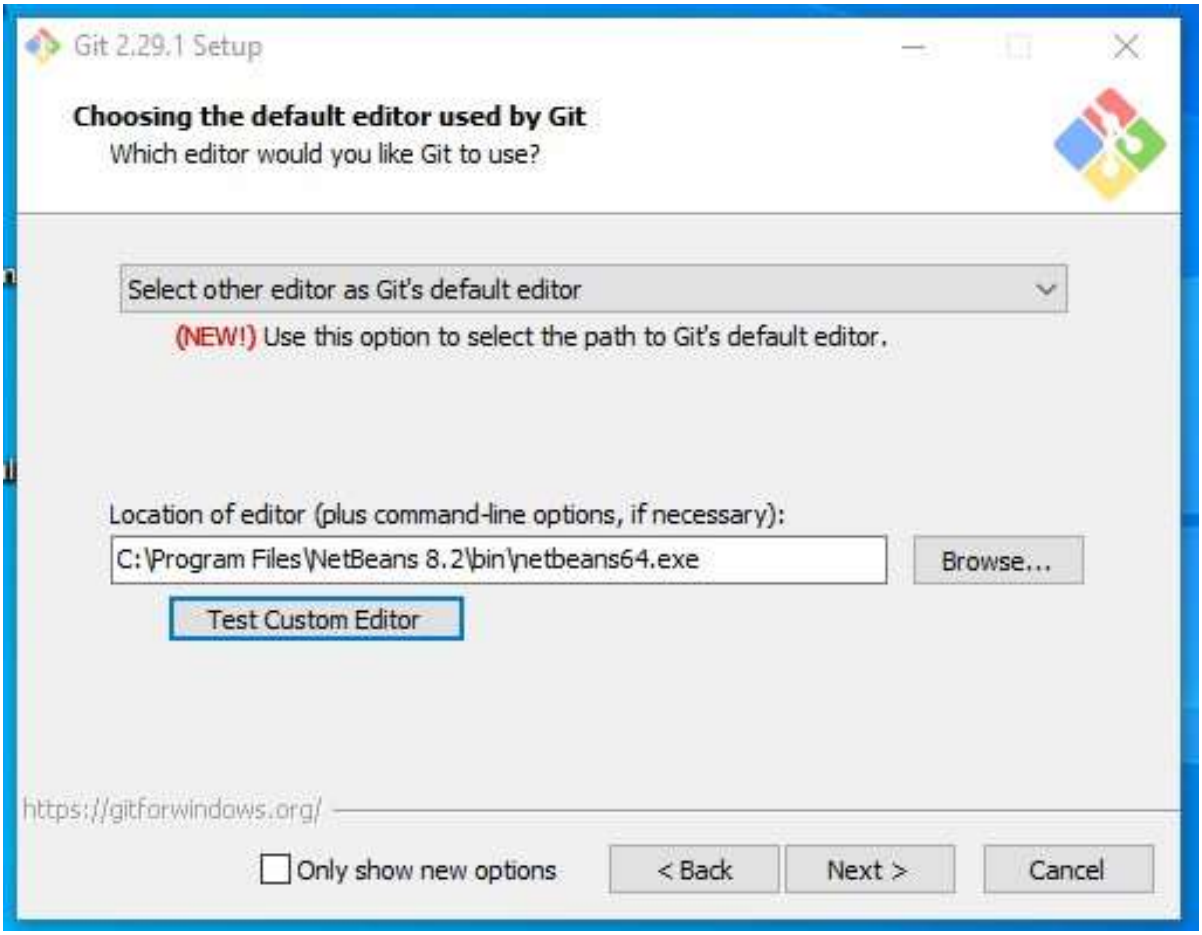
Link to download Git : https://git-scm.com/download/win

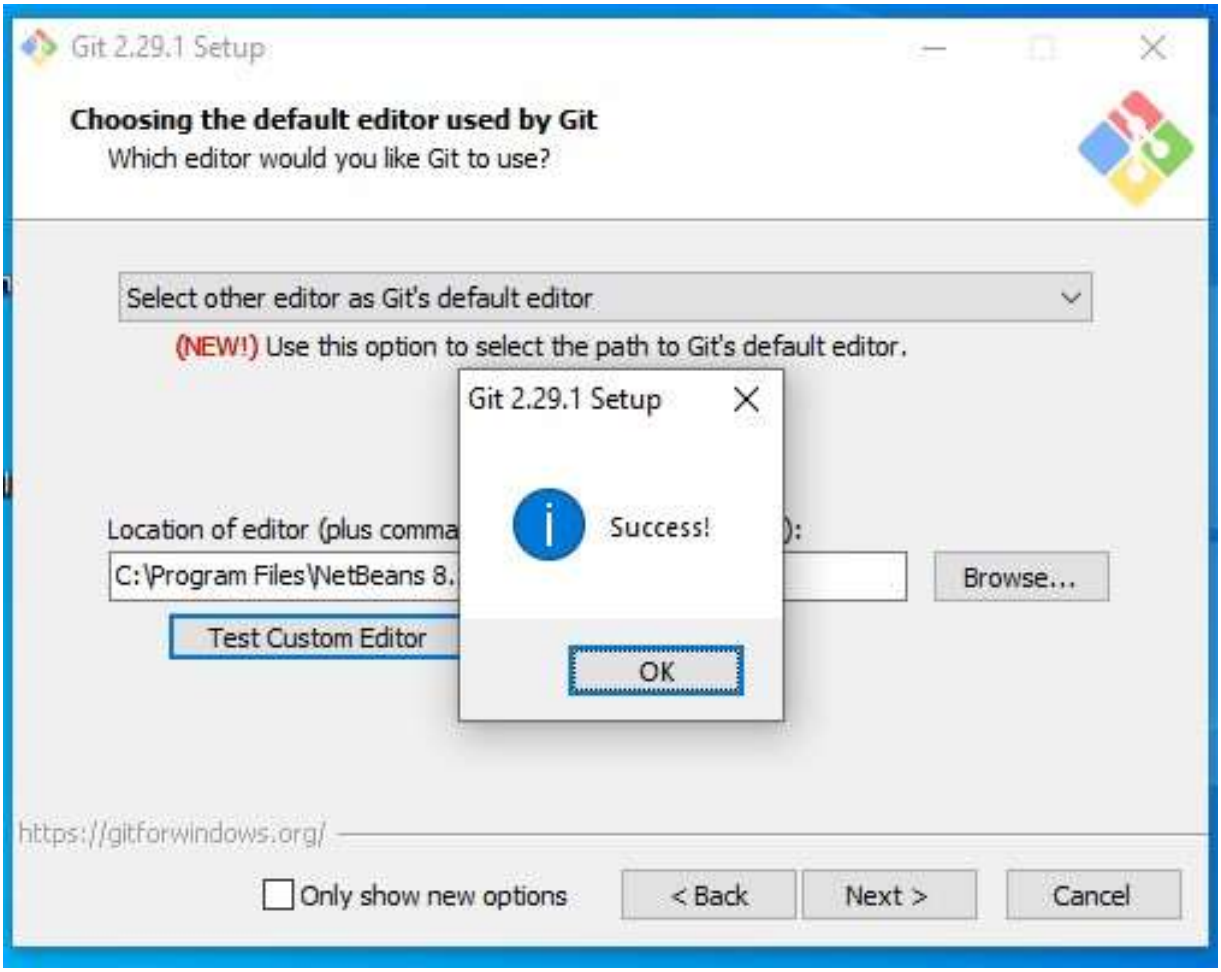


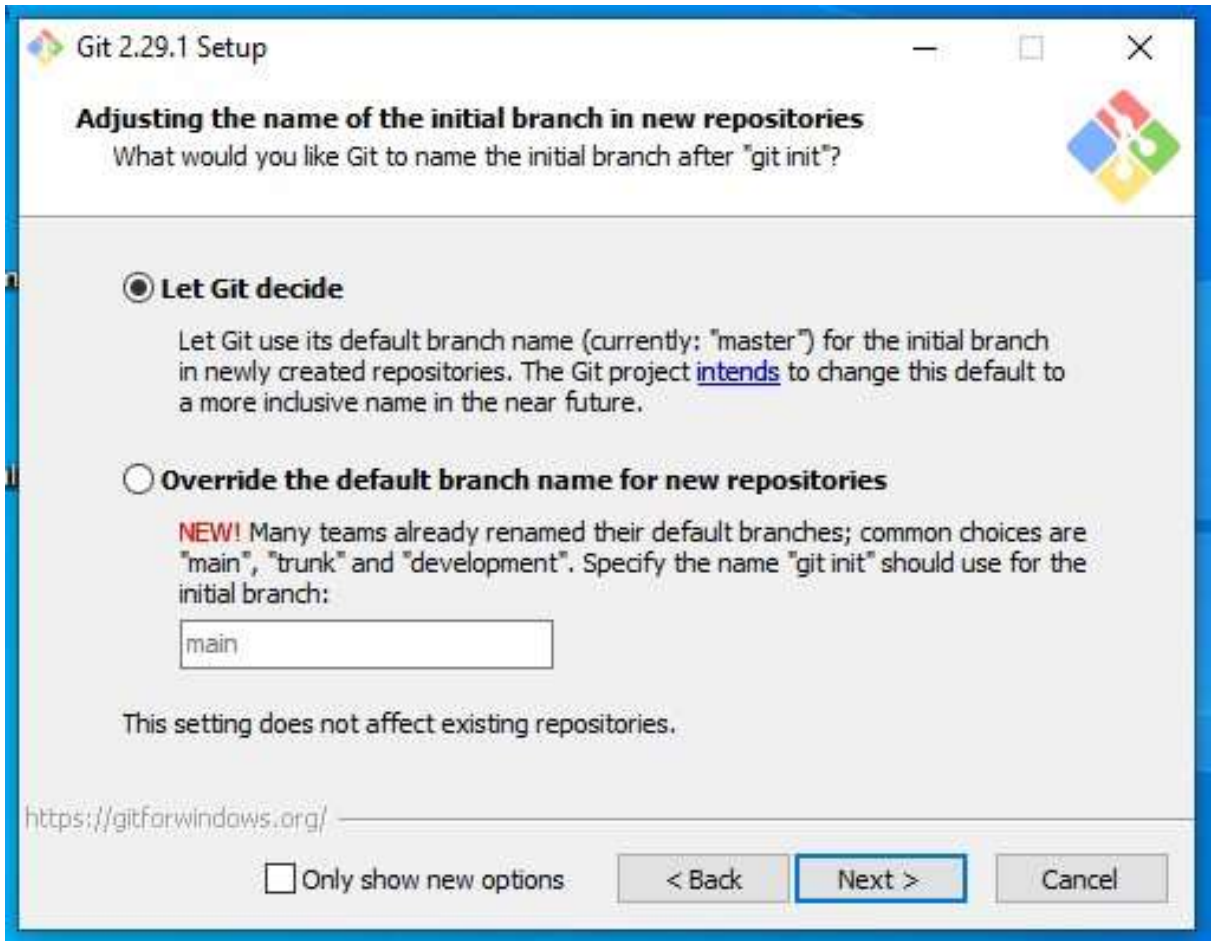
File downloaded : Git-2.29.1-64-bit.exe

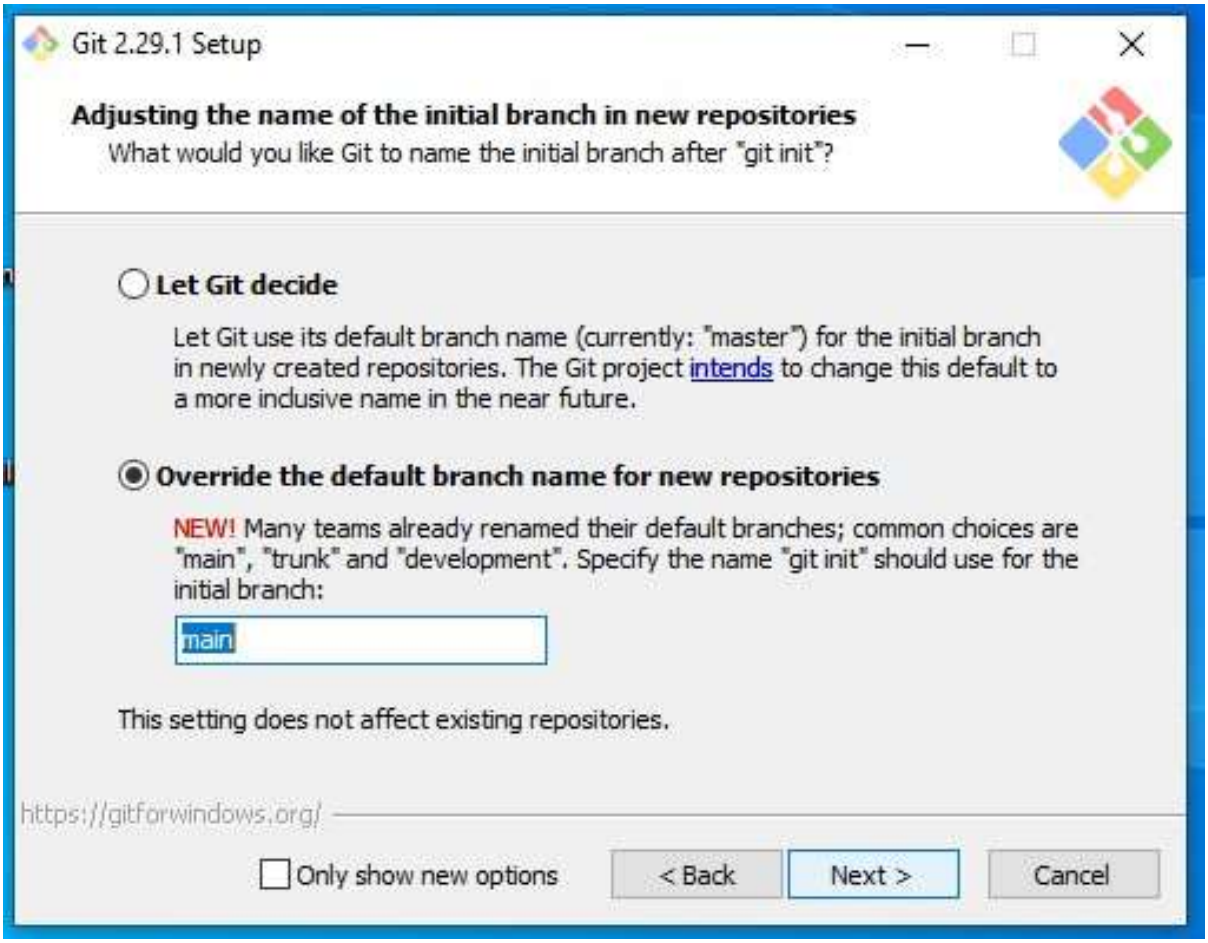

Git 2.29.1 Setup
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
GNU General Public License
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, 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
https://gitforwindows.org/
Only show new options
Next >
Cancel

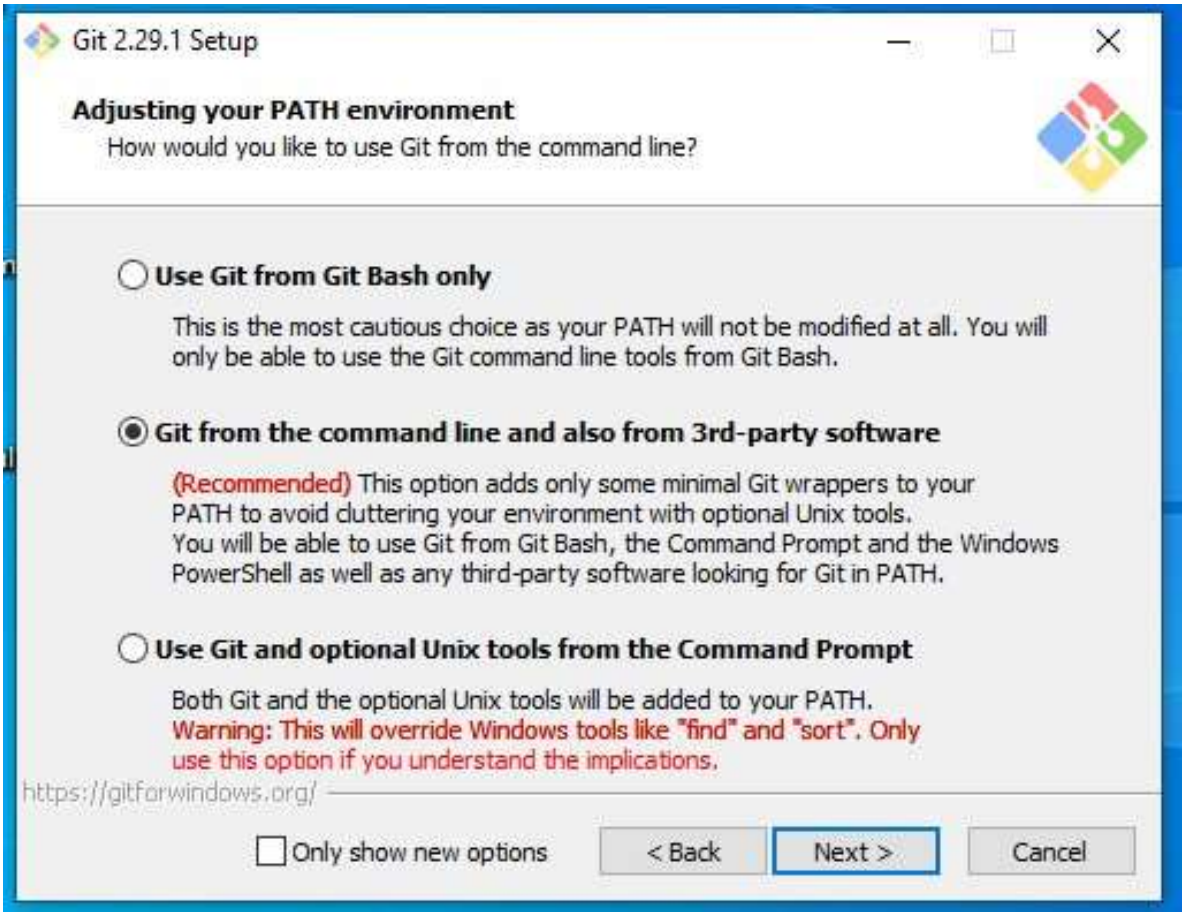

Git 2.29.1 Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
On the Desktop
Windows Explorer integration
Git Bash Here
Git GUI Here
Git LFS (Large File Support)
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
Check daily for Git for Windows updates
Current selection requires at least 257.3 MB of disk space.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel

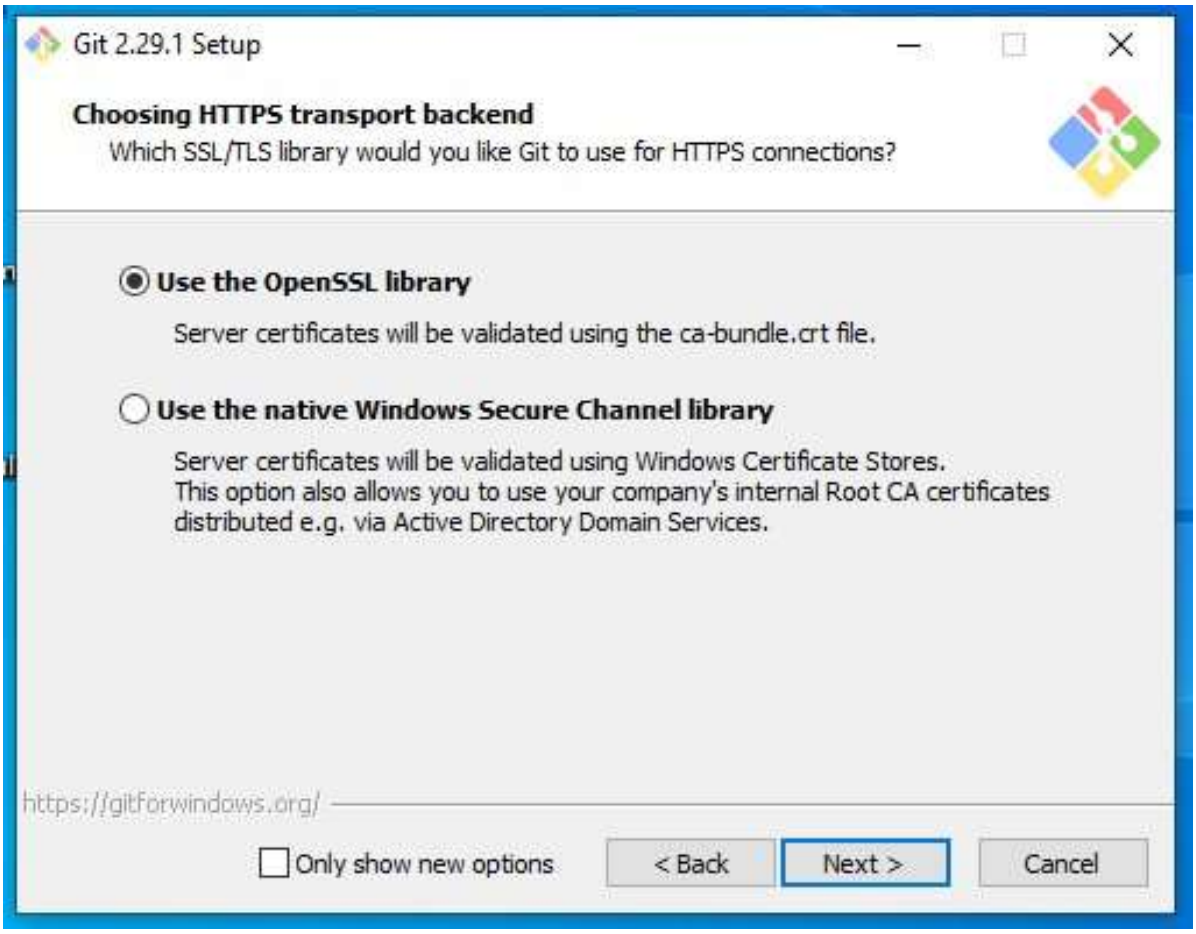

Git 2.29.1 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Select other editor as Git's default editor
(NEW!) Use this option to select the path to Git's default editor.
Location of editor (plus command-line options, if necessary):
C:\Program Files\NetBeans 8.2\bin\netbeans64.exe
Browse...
Test Custom Editor
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel

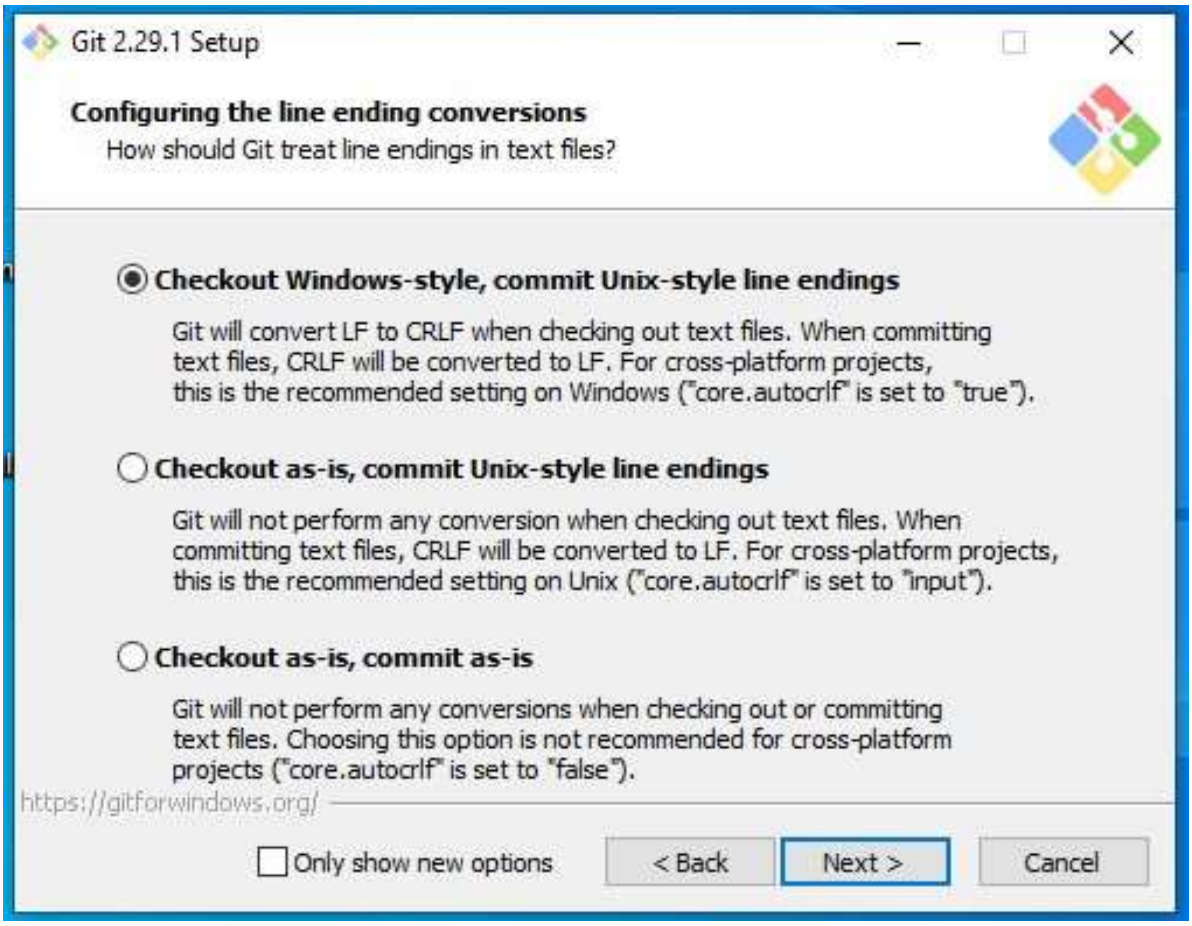

Git 2.29.1 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Select other editor as Git's default editor
(NEW!) Use this option to select the path to Git's default editor.
Location of editor (plus comma
C:\Program Files\NetBeans 8.
Browse...
Test Custom Editor
Git 2.29.1 Setup
Success!
OK
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel

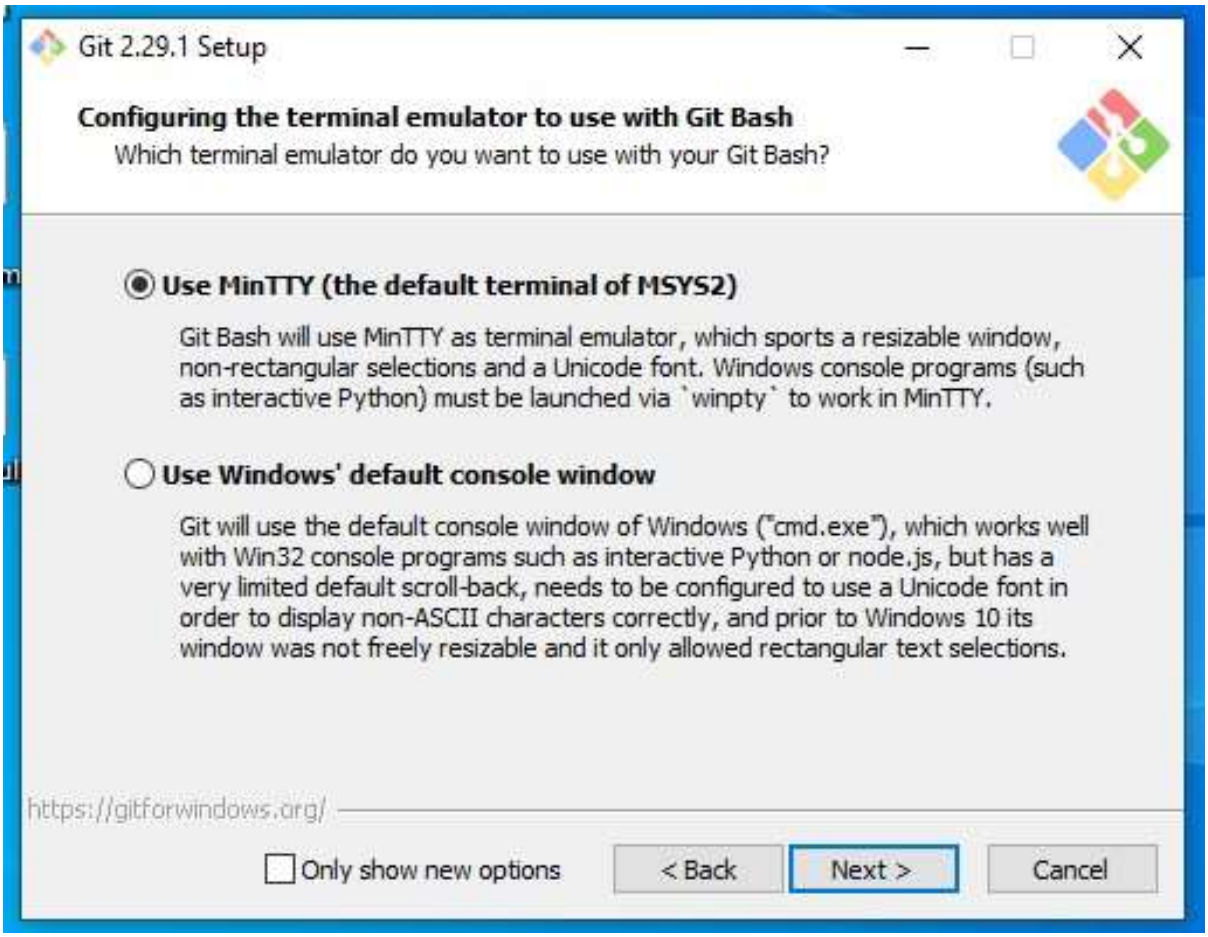

Git 2.29.1 Setup
Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?
Let Git decide
Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.
Override the default branch name for new repositories
NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:
main
This setting does not affect existing repositories.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel

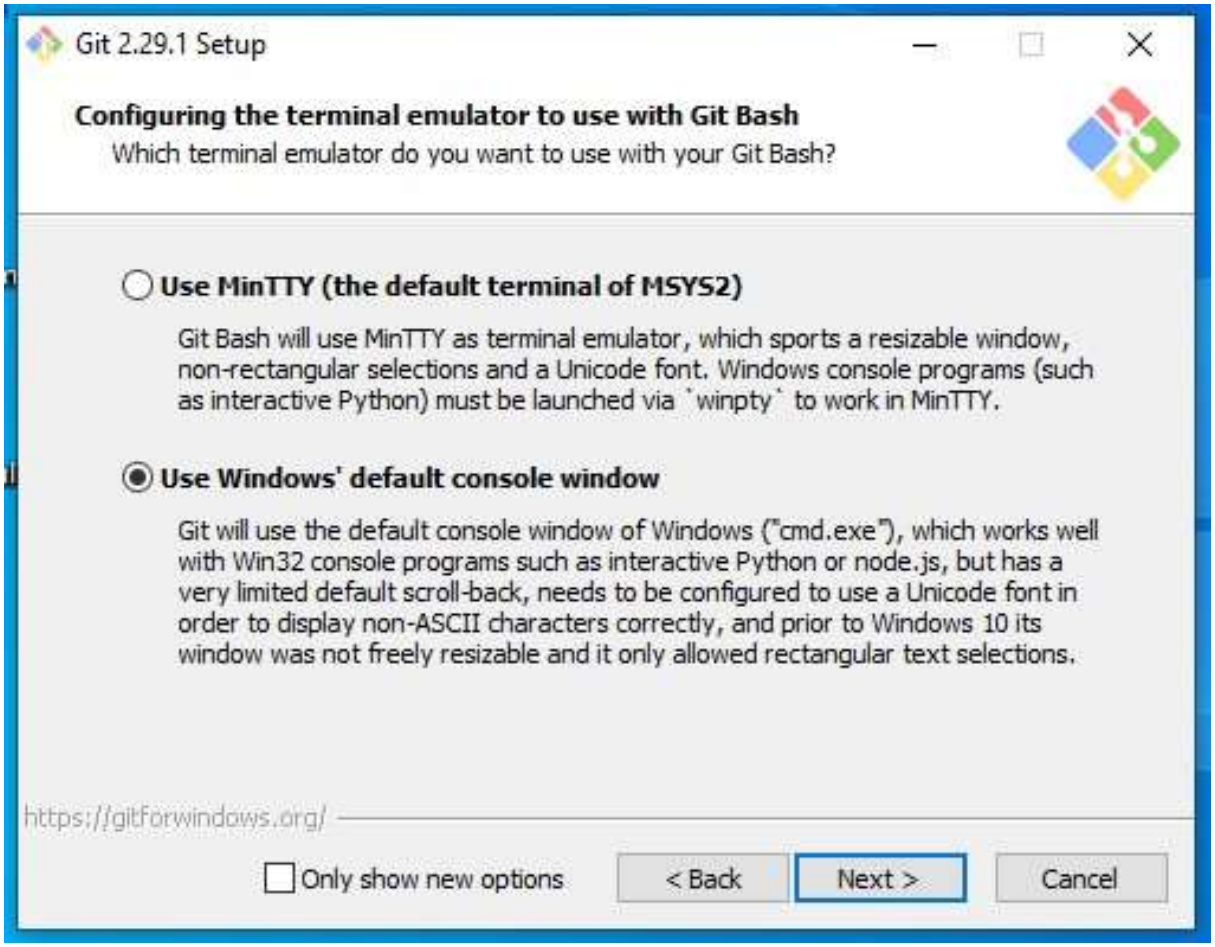

Git 2.29.1 Setup
Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?
Let Git decide
Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.
Override the default branch name for new repositories
NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:
main
This setting does not affect existing repositories.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel

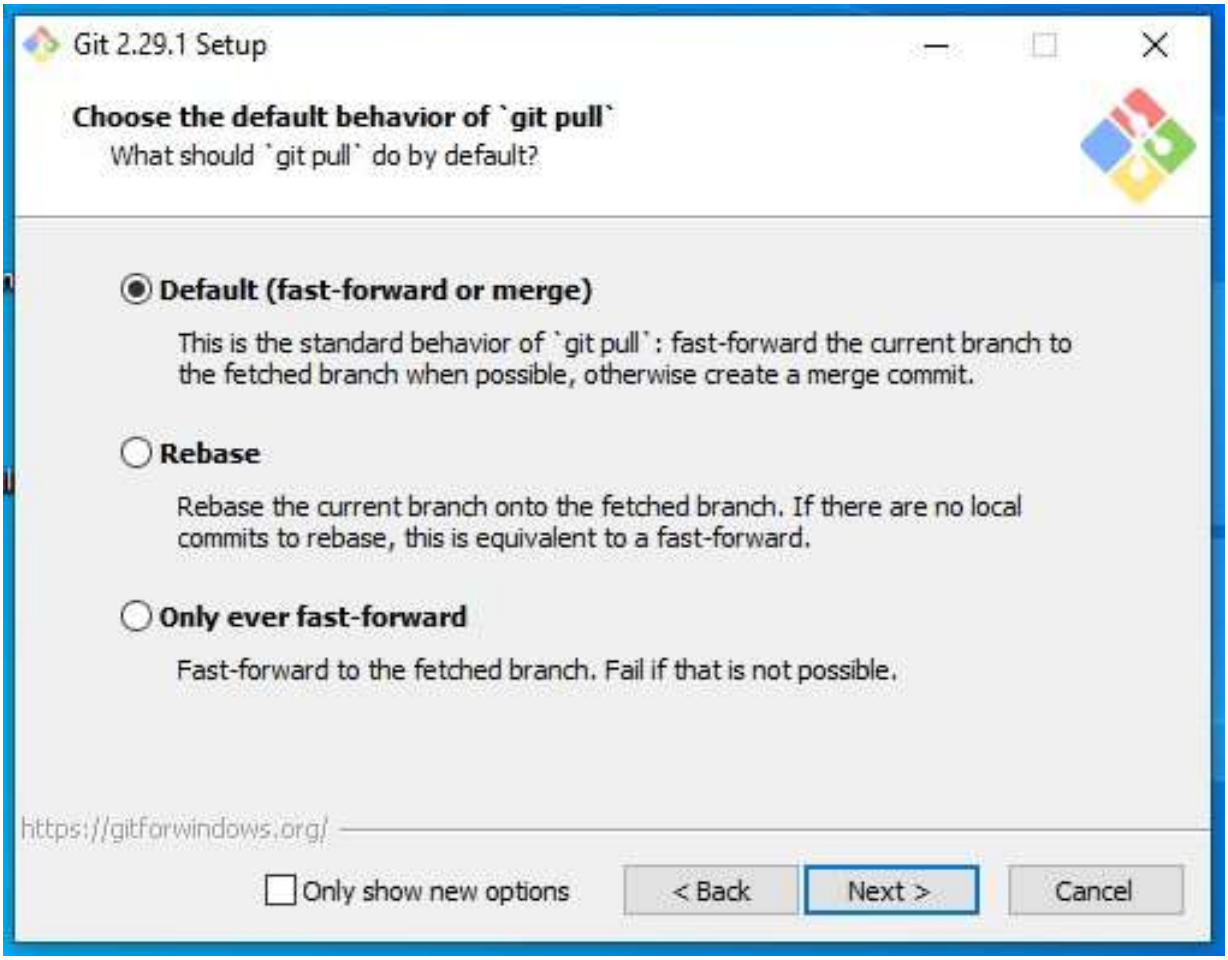

Git 2.29.1 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line 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 Windows 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/
Only show new options
< Back
Next >
Cancel

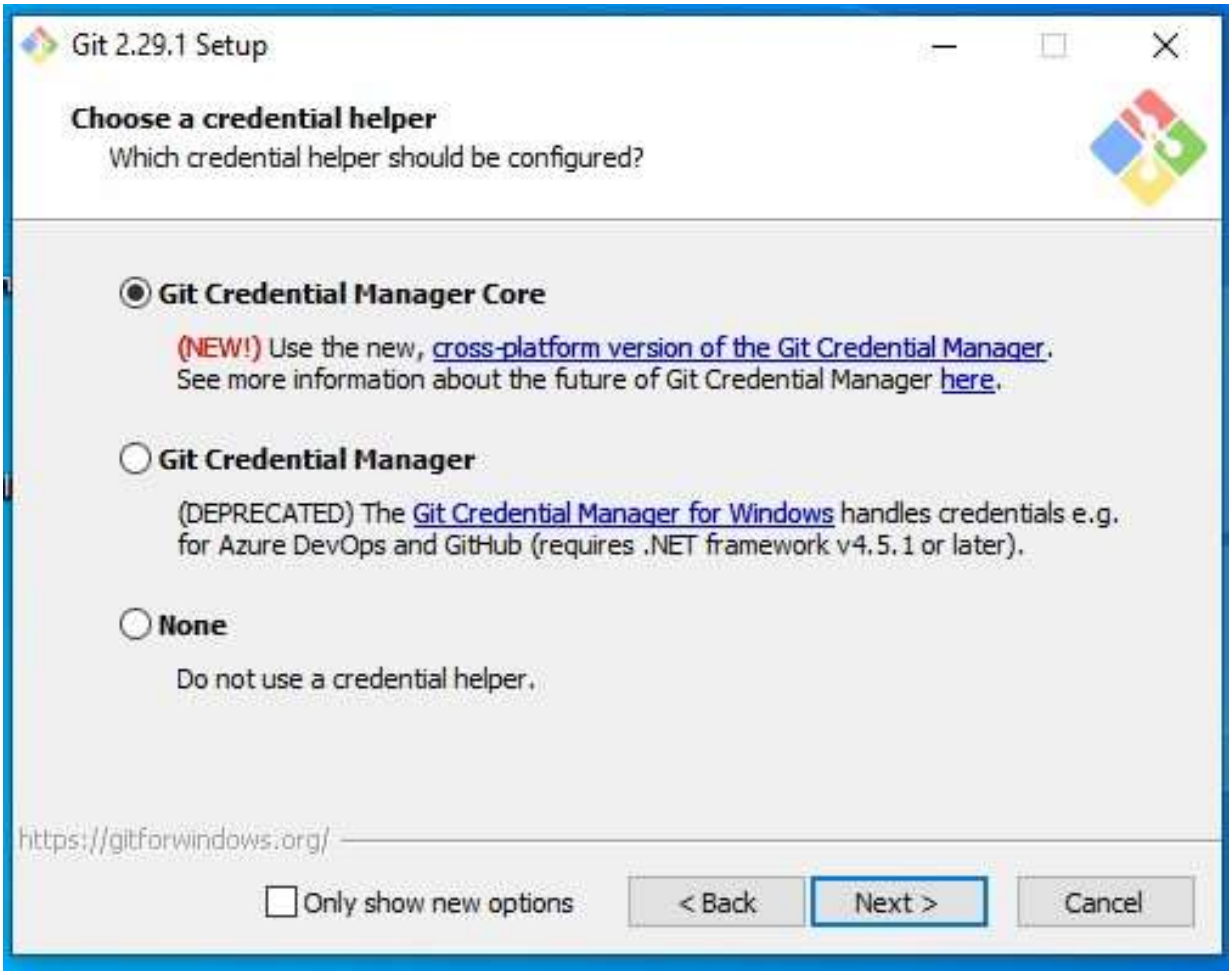

Git 2.29.1 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using 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/
Only show new options
< Back
Next >
Cancel

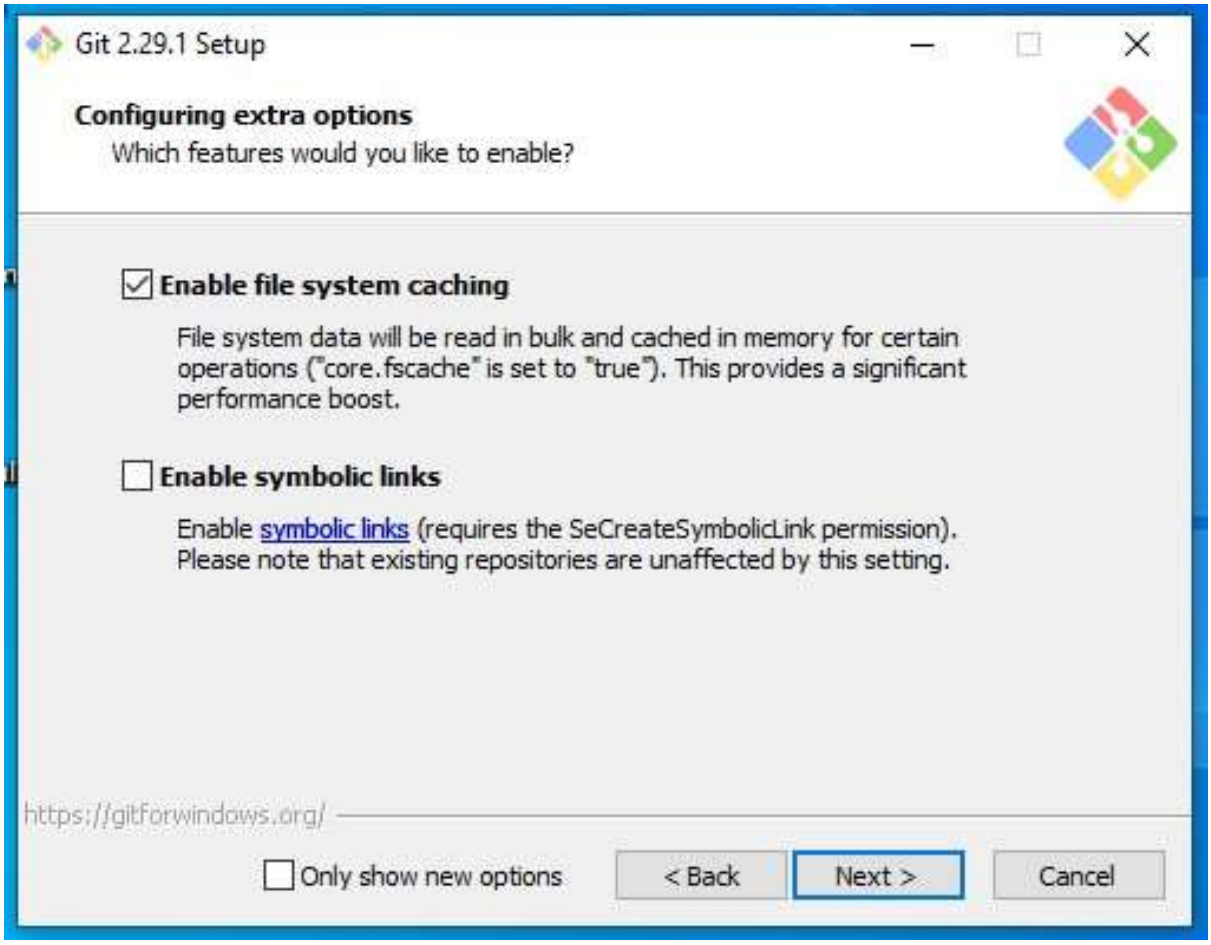

Git 2.29.1 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, 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/
Only show new options
< Back
Next >
Cancel

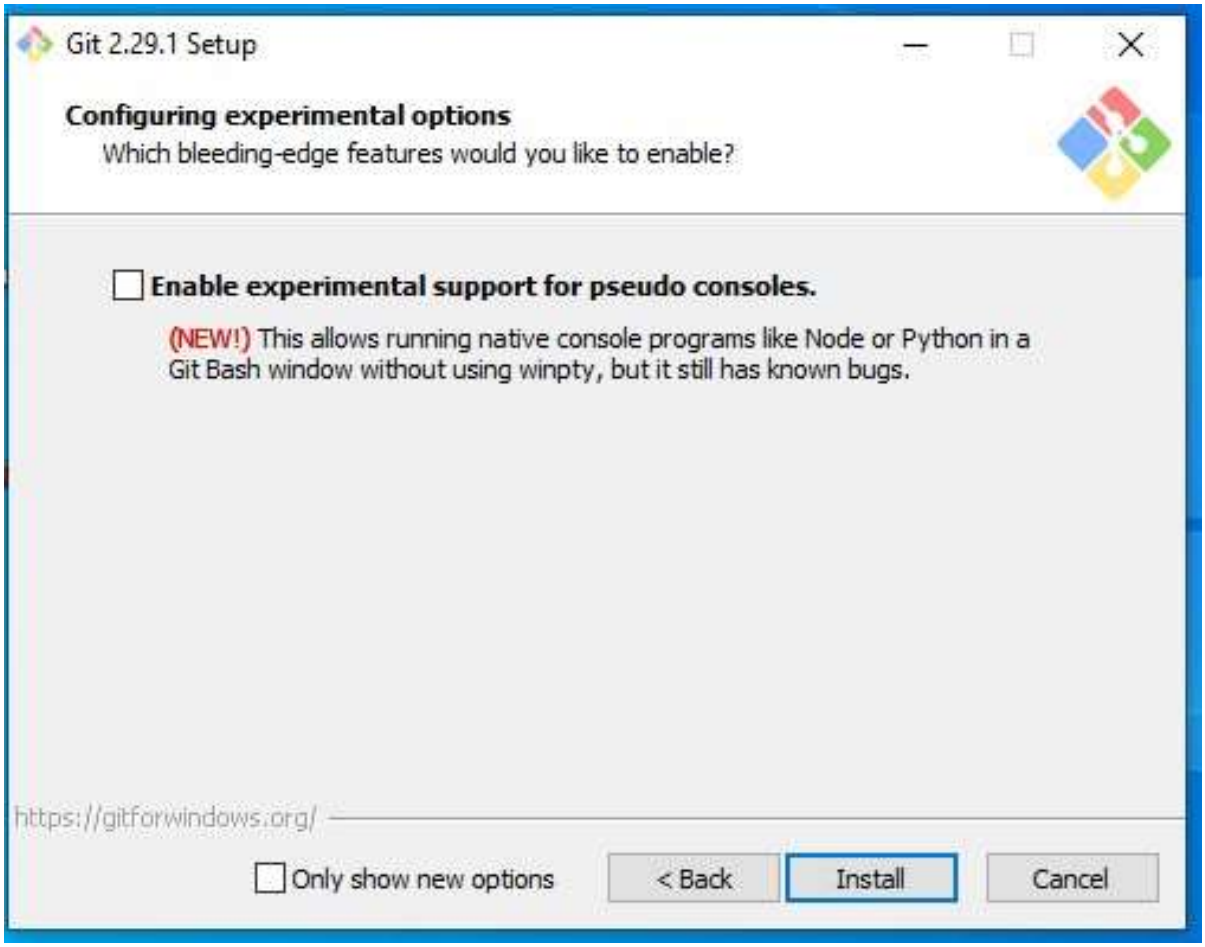

Git 2.29.1 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. 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 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/
Only show new options
< Back
Next >
Cancel

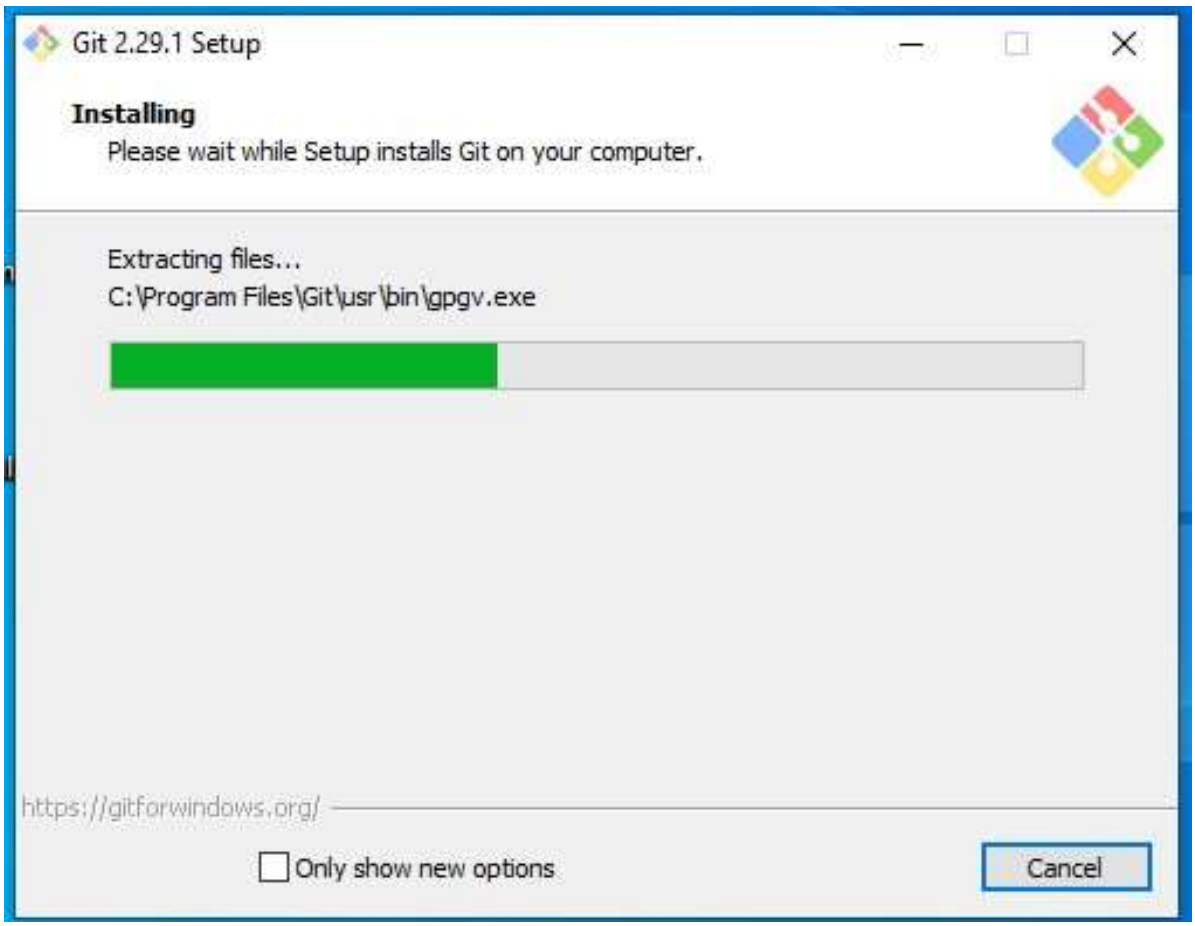

Git 2.29.1 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. 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 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/
Only show new options
< Back
Next >
Cancel


Git 2.29.1 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Default (fast-forward or merge)
This is the standard behavior of `git pull`: 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.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel


Git 2.29.1 Setup
Choose a credential helper
Which credential helper should be configured?
Git Credential Manager Core
(NEW!) Use the new, cross-platform version of the Git Credential Manager.
See more information about the future of Git Credential Manager here.
Git Credential Manager
(DEPRECATED) The Git Credential Manager for Windows handles credentials e.g.
for Azure DevOps and GitHub (requires .NET framework v4.5.1 or later).
None
Do not use a credential helper.
https://gitforwindows.org/
Only show new options
< Back
Next >
Cancel


Git 2.29.1 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/
Only show new options
< Back
Next >
Cancel


Git 2.29.1 Setup
Configuring experimental options
Which bleeding-edge features would you like to enable?
Enable experimental support for pseudo consoles.
(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty, but it still has known bugs.
https://gitforwindows.org/
Only show new options
< Back
Install
Cancel


Git 2.29.1 Setup
Installing
Please wait while Setup installs Git on your computer.
Extracting files...
C:\Program Files\Git\usr\bin\gpgv.exe
https://gitforwindows.org/
Only show new options
Cancel


Git 2.29.1 Setup
Completing the Git Setup Wizard
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Launch Git Bash
View Release Notes
Only show new options
Next >


Git 2.29.1 Setup
Completing the Git Setup Wizard
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Launch Git Bash
View Release Notes
Only show new options
Next >