ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY

CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY USING DevOps

Exp-4
GitHub

By
B Manikyala Rao M.Tech(Ph.d)
Assistant Professor
Dept of Computer Science & Engineering
Aditya College of Engineering & Technology
Surampalem

GitHub

when we were developers and writing code as part of a team we encountered recurring problems that were for the most part as follows:

- How to share my code with my team members
- How to version the update of my code
- How to track changes to my code
- How to retrieve an old state of my code or part of it

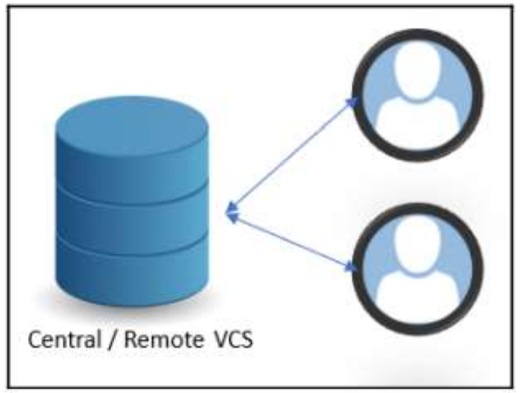
These issues have been solved with the emergence of source code managers, also called a Version Control System (VCS)

The goals of these VCSes are mainly to do the following:

- Allow collaboration of developers' code.
- Retrieve the code.
- Version the code.
- Track code changes.

we will see how to use one of the best-known VCSes, which is Git.

VCS



There are two types of VCS: i)Centralized and ii)Distributed systems.

i)centralized systems:

➢ These systems consist of a remote server that centralizes the code of all developers.

➢ All developers can archive and retrieve their code on the remote server.

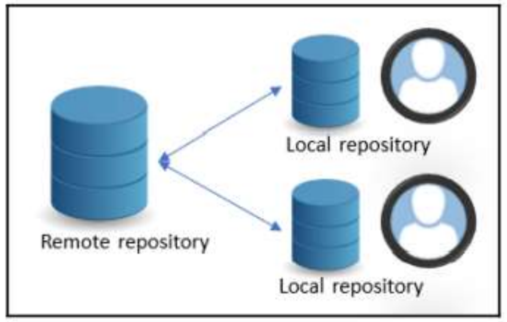
Drawbacks:

- In case of no connection (for a network problem or internet disconnection) between the developers and the remote server, no more archiving or code recovery actions can be performed.
- If the remote server no longer works, the code, as well as the history, will be lost.

VCS

ii)Distributed system:

- These systems consist of a remote repository and a local copy of this repository on each developer's local machine.
- with this distributed system, even in the event of disconnection from the remote repository, the developer can continue to work with the local repository, and synchronization will be done when the remote repository is accessible again.
- Git is, therefore, a distributed CVS that was created in 2005 by Linus Torvalds and the Linux development communit



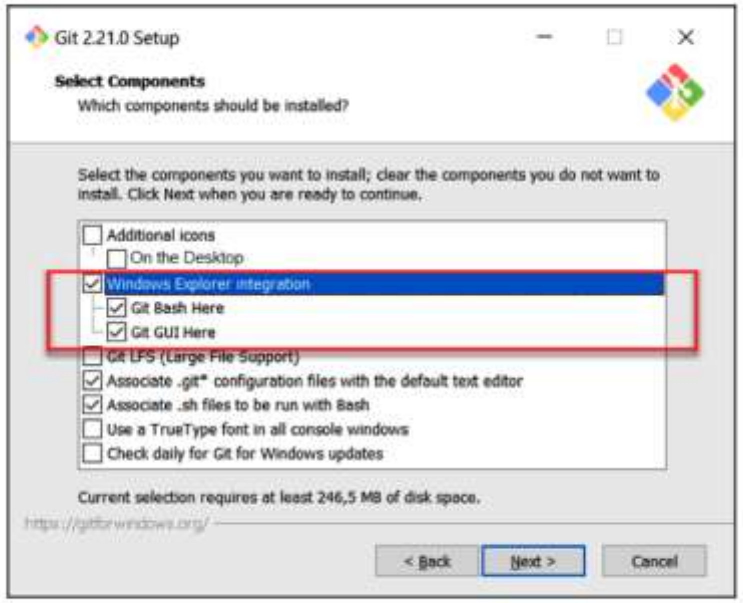
Git

- Git is a free, cross-platform tool, and it can be installed on a local machine for people who manipulate code, that is, in client mode, but can also be installed on servers to host and manage remote repositories.
- Git is a command-line tool with a multitude of options.
- there are many graphical tools, such as Git GUI, Git Kraken, GitHub desktop, or SourceTree, that allow you to interact with Git operations more easily and graphically without having to use command lines.
- For remote repositories, there are several clouds and free solutions such as GitHub, GitLab, Azure DevOps, or Bitbucket cloud

Git installation

To install on a Windows machine manually, we must download the Git for Windows tool executable from **https://gitforwindows.org/** and, once downloaded, we click on the executable file and follow the following different configuration steps during the installation:

1. Choose the integration of Git in Windows Explorer by marking the Windows Explorer integration checkbox.



Git 2.21.0 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.

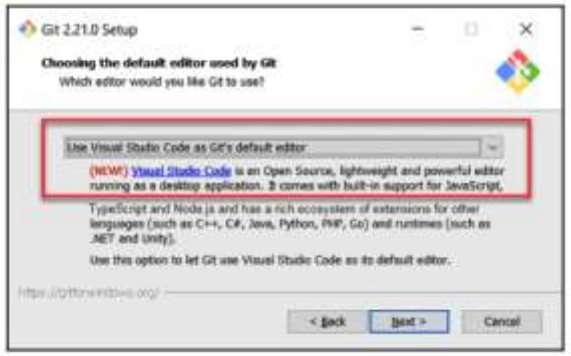
- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [ ] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 246,5 MB of disk space.

https://gitforwindows.org/

Git

2. Choose your code editor IDE; in our case, we use Visual Studio Code by selecting the Use Visual Studio Code as Git's default editor option.
3.Choose the PATH environment variable option or we can leave the default choice proposed by the installer:
4. Choose the type of HTTPS transport that we will also leave by default with the Use the OpenSSL library option:



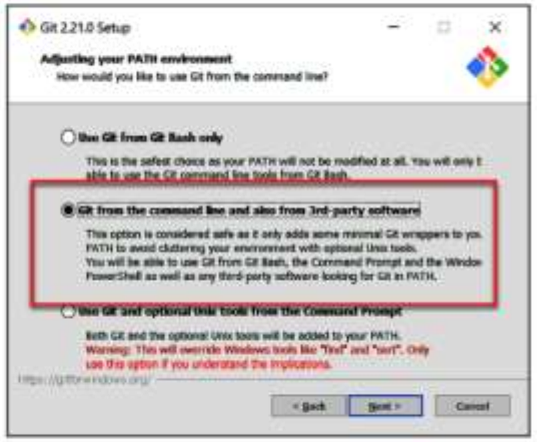
Choosing the default editor used by Git
Which editor would you like Git to use?

Use Visual Studio Code as Git's default editor

(NEW!) Visual Studio Code is an Open Source, lightweight and powerful editor running as a desktop application. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Use this option to let Git use Visual Studio Code as its default editor.

https://gitforwindows.org/



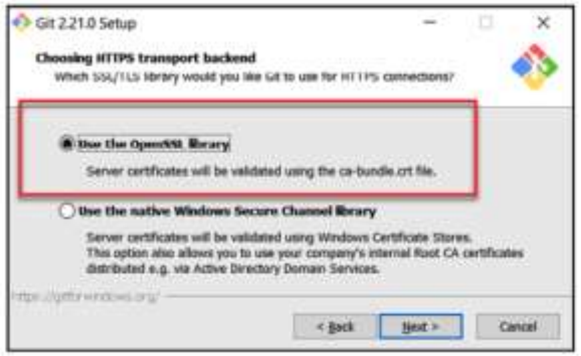
Adjusting your PATH environment
How would you like to use Git from the command line?

○ 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.

● Git from the command line and also from 3rd-party software
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 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/



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.

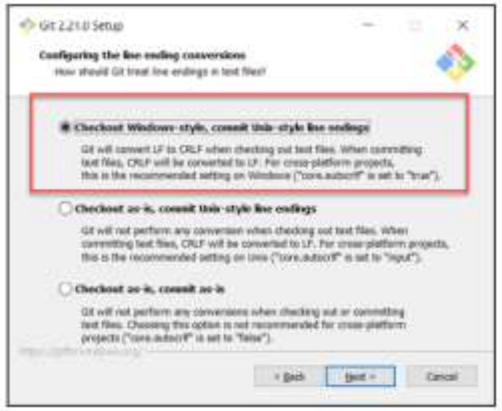
https://gitforwindows.org/

Git

5. Choose the encoding of the end files;

   we will also select the default option,

   which archives in Unix format:



Git 2.21.0 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").

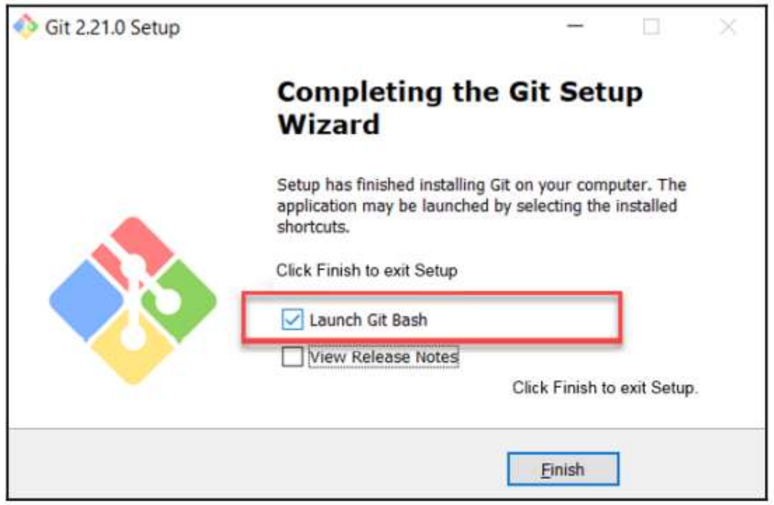
- Checkout as-is, commit Unix-style line endings

  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").

6. Then, finish the installation configuration by clicking on the Install button. At the end of the installation, the installation utility proposes to open Git Bash, which is a command-line Terminal Linux emulator dedicated to Git commands



Git 2.21.0 Setup

Completing the Git Setup Wizard

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

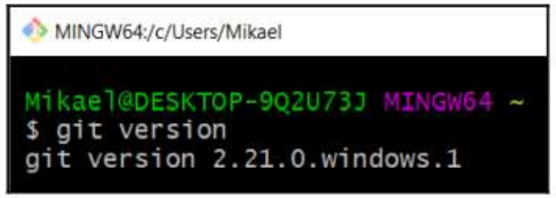
Click Finish to exit Setup

- [✓] Launch Git Bash
- [ ] View Release Notes

Click Finish to exit Setup.

Git

After the installation, we can immediately check the status of the Git installation directly in the Git Bash window by running the git version command.



```
MINGW64:/c/Users/Mikael

Mikael@DESKTOP-9Q2U73J MINGW64 ~
$ git version
git version 2.21.0.windows.1
```

Configuration Git:

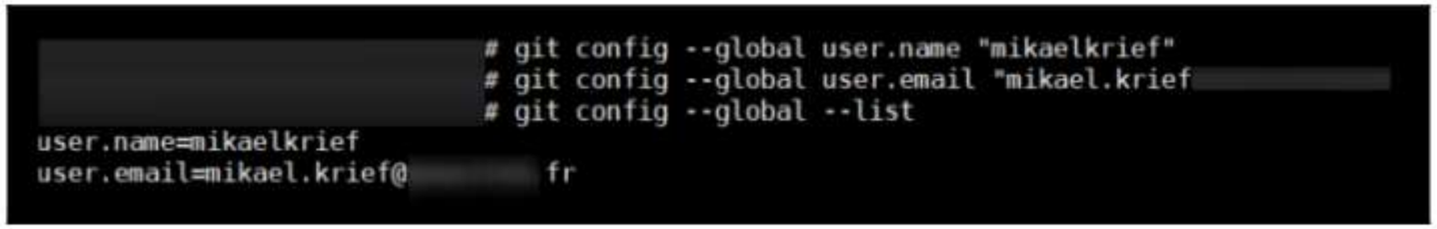
Git configuration requires us to configure our username and email that will be used during code commit. To perform this configuration, we execute the following commands in a Terminal.

git config --global user.name ""

git config --global user.email ""

we can check the configuration values by executing the following command:

git config --global --list



```
[redacted] # git config --global user.name "mikaelkrief"
           # git config --global user.email "mikael.krief[redacted]
           # git config --global --list
user.name=mikaelkrief
user.email=mikael.krief@[redacted] fr
```

Git vocabulary

Git vocabulary:

Repository: A repository is the basic element of Git; it is the storage space where the sources are tracked and versioned.

Clone: Cloning is the act of making a local copy of a remote repository.

Commit: A commit is a change made to one or more files, and the change is saved to the local repository. Each commit is unique and is identified by a unique number called SHA-1, by which code changes can be tracked.

Branch: The code that is in the repository is stored by default in a master branch. A branch can create other branches that will be a replica of the master on which developers make changes, and that will allow us to work in isolation without affecting the master branch.

Merge: This is the action that consists of merging the code of one branch with another

Checkout: This is the action that allows to switch from one branch to another.

Fetch: This is the action of retrieving the code from the remote repository without merging it with the local repository.

Pull: This is the action that consists of updating your local repository from the remote repository. A pull is equivalent to fetch and merge.

Push: A push is the reverse action of a pull—it allows us to update the remote repository from the local repository.

Git command lines

Retrieving a remote repository : The first command line to know is the clone command that makes a copy of a remote repository to create a local repository.

git clone <URL of remote repository>

Initializing a local repository:init is the Git command that allows you to create a local repository. To do this in the directory that will contain your local repository, run the following simple command.

git init

This command creates a .git directory that contains all of the folders and configuration files of the local repository

Configuring a local repository : After the init command, the new local repository must be configured by setting up the linked remote repository. To make this setting, we will add remote with the following command:

git remote add <name> <URL of remote reposotiry>

It is also possible to configure several remote devices on our local repository.

Adding a file for the next commit: Making a commit (which we will see next) is to archive our changes in our local repository. When we edit files, we can choose which ones will be included in the next commit; it's the staged concept.

git add <files path to add>

if we want all of the files modified at the next commit, execute the git add . command

Git

Creating a commit :A commit is the Git entity that contains a list of changes made to files and that have been registered in the local repository. Making a commit, therefore, consists of archiving changes made to files that have been previously selected with the add command. The command to create a commit is as follows: **git commit -m "your commit meassage"**

The message is very important because we will be able to identify the reason for the changes in the files.

It is also possible to commit all files modified since the last commit

git commit -a -m "message"

Updating the remote repository : When we make commits, they are stored in the local repository and when we are ready to share them with the rest of the team for validation or deployment, we must publish them to the remote repository. To update a remote repository from commits made on a local repository, a push operation is performed with this command:

git push <alias> <branch>

Synchronizing the local repository from the remote:the Git command line is used to update the remote repository from the local repository. Now, to perform the reverse operation, that is, update the repository with all of the changes of the other members that have been pushed on the remote repository, we will perform a pull operation with this command:

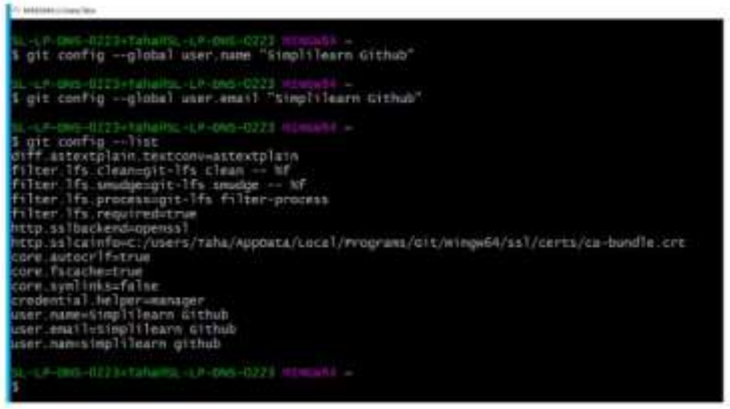
git pull

Git

1.Let's begin with opening Git Bash and configuring it with a user name and email ID.

To configure, we use the following commands:

- Git config --global user.name “manikbollu"
- Git config --global user.email “manik.bollu@gmail.com”
- Git config –list



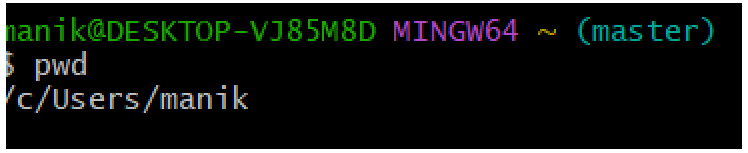
```
SL-LP-DNS-0223+TahaRSL-LP-DNS-0223 MINGW64 ~
$ git config --global user.name "Simplilearn Github"

SL-LP-DNS-0223+TahaRSL-LP-DNS-0223 MINGW64 ~
$ git config --global user.email "Simplilearn Github"

SL-LP-DNS-0223+TahaRSL-LP-DNS-0223 MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/users/Taha/AppData/Local/Programs/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager
user.name=Simplilearn Github
user.email=simplilearn Github
user.nam=simplilearn github

SL-LP-DNS-0223+TahaRSL-LP-DNS-0223 MINGW64 ~
$
```

2.Then, let's check the current working directory: pwd



```
nanik@DESKTOP-VJ85M8D MINGW64 ~ (master)
$ pwd
/c/Users/manik
```

3. To create a repository in the working directory, use the following commands:

- mkdir Git_Demo
- cd Git_Demo
- pwd

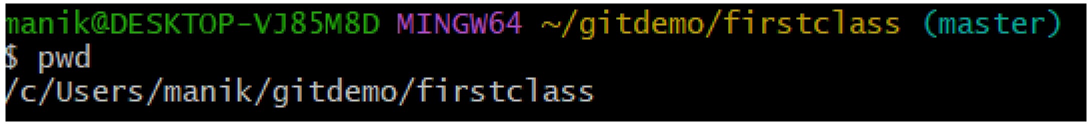
We can go to the directory location and check the Git_demo folder.

The directory "Git_demo" will be empty for now

Git

4.Let's create a folder for the repository:

- mkdir firstclass
- cd firstclass
- Pwd



```
manik@DESKTOP-VJ85M8D MINGW64 ~/gitdemo/firstclass (master)
$ pwd
/c/Users/manik/gitdemo/firstclass
```

5. The folder “firstclass" is empty. We will now initialize a repository to our folder.

- Git init

5.Moving further, let's make some commits. For that, I will create two notepads and commit them one by one.

For the first notepad, the commands are as follows:

- touch alpha.txt
- notepad alpha.txt

A notepad opens on the screen. Type anything inside it, save it and close it.

Git

Next, let's check the status of the file that was created.

- **git status**

7. For Git to track that file, add command is used. If you know the exact name of the file, you can specify it and simply type the following command:

- **git add .**

8. The next step is to commit the file.

- **git commit -m "alpha“**

9. Next, check all the information regarding the commits that were made.

- **git log**

This displays the commit ID, author's name, and email ID used. You can also find the date and commit message on the screen.

10. Let's make one more commit. Repeat the same process again.

11. Now, let's push the two notepads on GitHub. Open your GitHub account, and create a new repository. The name of the repository will be "Firstclass”

12. Copy the "git remote add origin" URL.

13. Now, let's push the content on to the remote repository.

- **git push -u origin master**

Git

14.The repository is created on the server, and the content is pushed into that repository. It links the master branch on the local repository to the master branch on the server.

15. Next, refresh the GitHub page, and you can find all the commits there.

- Each commit has a hash ID, which contains the details of each commit.
- You can open each notepad and check the content inside.

- Apply git pull request from git bash and check whether the files are updated or not

Git pull

Notepad filename.txt

- Again do modifications and commit change and repeat the same process.

ANY QUERIES