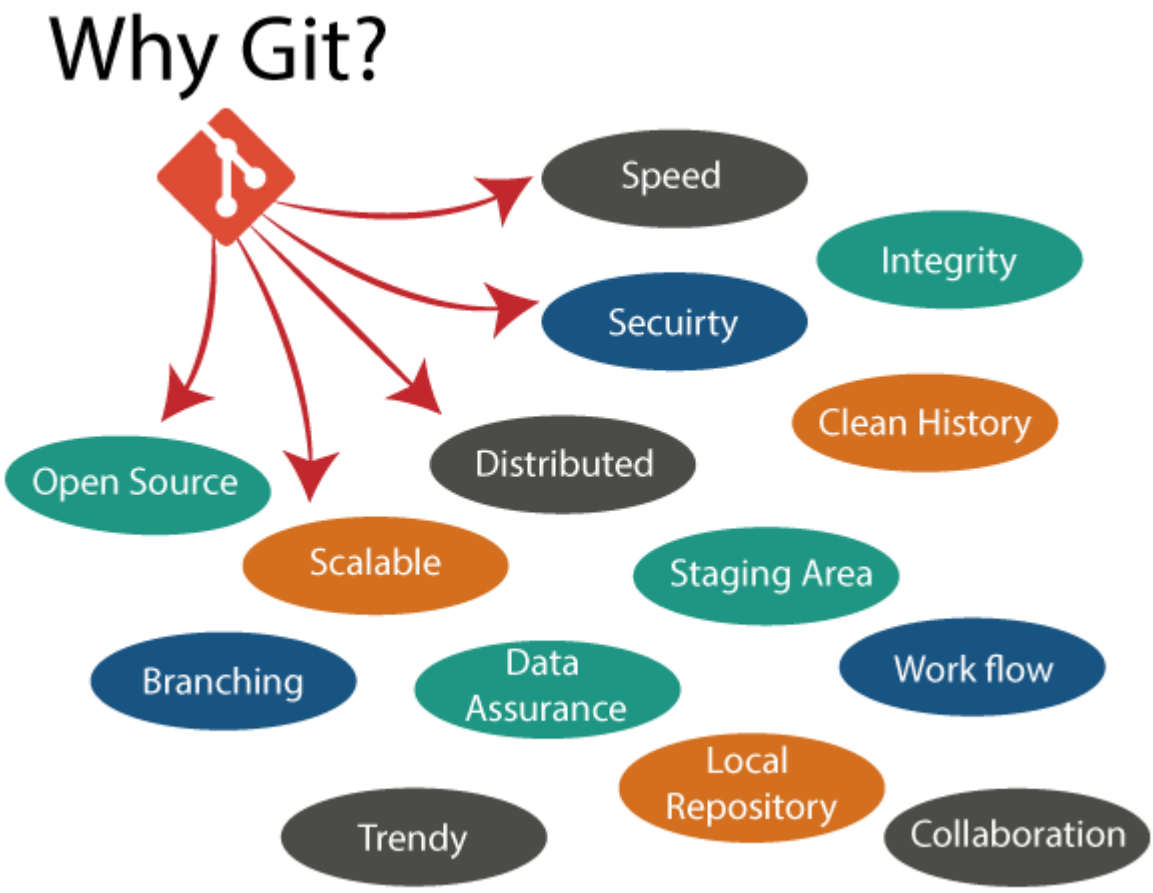
Day 8 Task: Basic Git & GitHub for DevOps Engineers.

What is Git?

Git is a mature, actively maintained free and open-source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. Git is a version control system that allows you to track changes to files and coordinate work on those files among multiple people with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning-fast performance. It means that you can keep a record of who made changes to what part of a file, and you can revert back to earlier versions of the file if needed. Git also makes it easy to collaborate with others, as you can share changes and merge the changes made by different people into a single version of a file. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows



What is GitHub?

GitHub is a product that allows you to host your Git projects on a remote server somewhere (or in other words, in the cloud).

It is a subsidiary of Microsoft, and it offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

It's important to remember that GitHub is not Git. GitHub is just a hosting service. There are other companies who offer hosting services that do the same thing as GitHub, such as Bitbucket and GitLab. GitHub is a very popular platform for developers to share and collaborate on projects, and it is also used for hosting open-source projects.

What is Version Control? How many types of version controls we have?

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

There are 3 main types of version control systems:

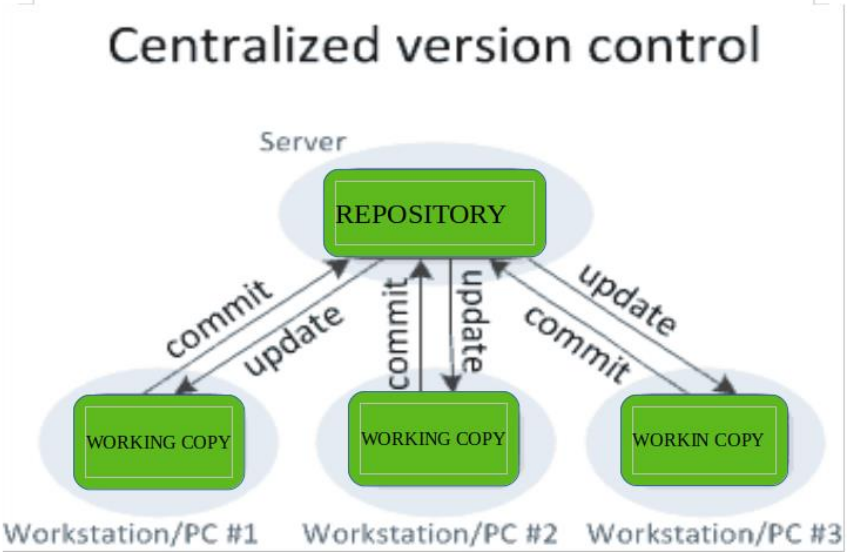
1. Local Version Control Systems
2. Centralized version control systems
3. Distributed version control systems.

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized version control system (CVCS) uses a central server to store all the versions of a project's files and Its contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

- You commit
- They update

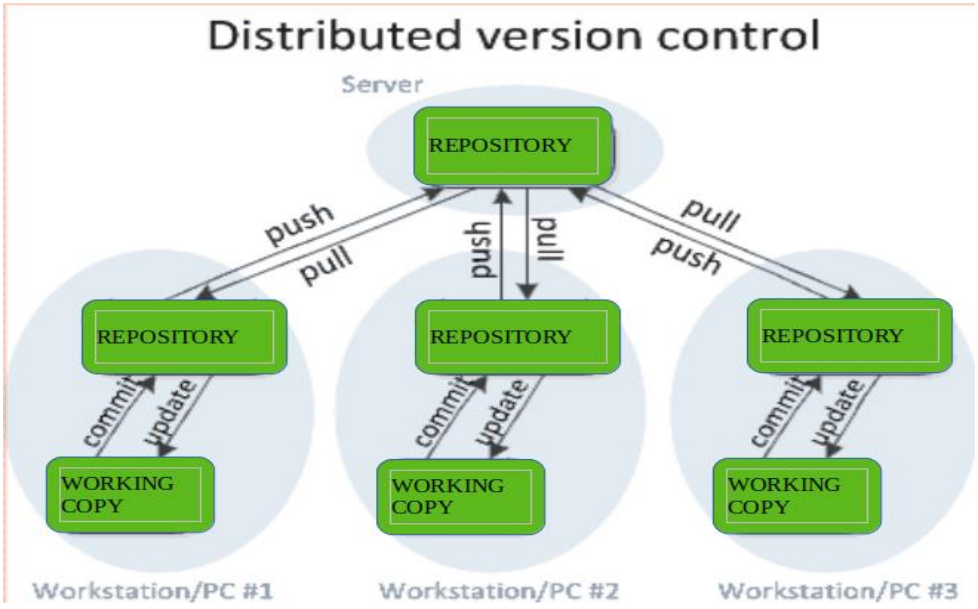


Distributed version control systems (DVCS) allows user to "clone" an entire repository, including the entire version history of the project. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, when you update, you do not get others' changes unless you have first pulled those changes into your repository. The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.

To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

.



Why we use distributed version control over centralized version control?

1. Better collaboration: In a DVCS, every developer has a full copy of the repository, including the entire history of all changes. This makes it easier for developers to work together, as they don't have to constantly communicate with a central server to commit their changes or to see the changes made by others.

2. Improved speed: Because developers have a local copy of the repository, they can commit their changes and perform other version control actions faster, as they don't have to communicate with a central server.

3. Greater flexibility: With a DVCS, developers can work offline and commit their changes later when they do have an internet connection. They can also choose to share their changes with only a subset of the team, rather than pushing all of their changes to a central server.

4. Enhanced security: In a DVCS, the repository history is stored on multiple servers and computers, which makes it more resistant to data loss. If the central server in a CVCS goes down or the repository becomes corrupted, it can be difficult to recover the lost data.

Overall, the decentralized nature of a DVCS allows for greater collaboration, flexibility, and security, making it a popular choice for many teams.

Task:

- Install Git on your computer (if it is not already installed).

  You can download it from the official website at https://git-scm.com/downloads

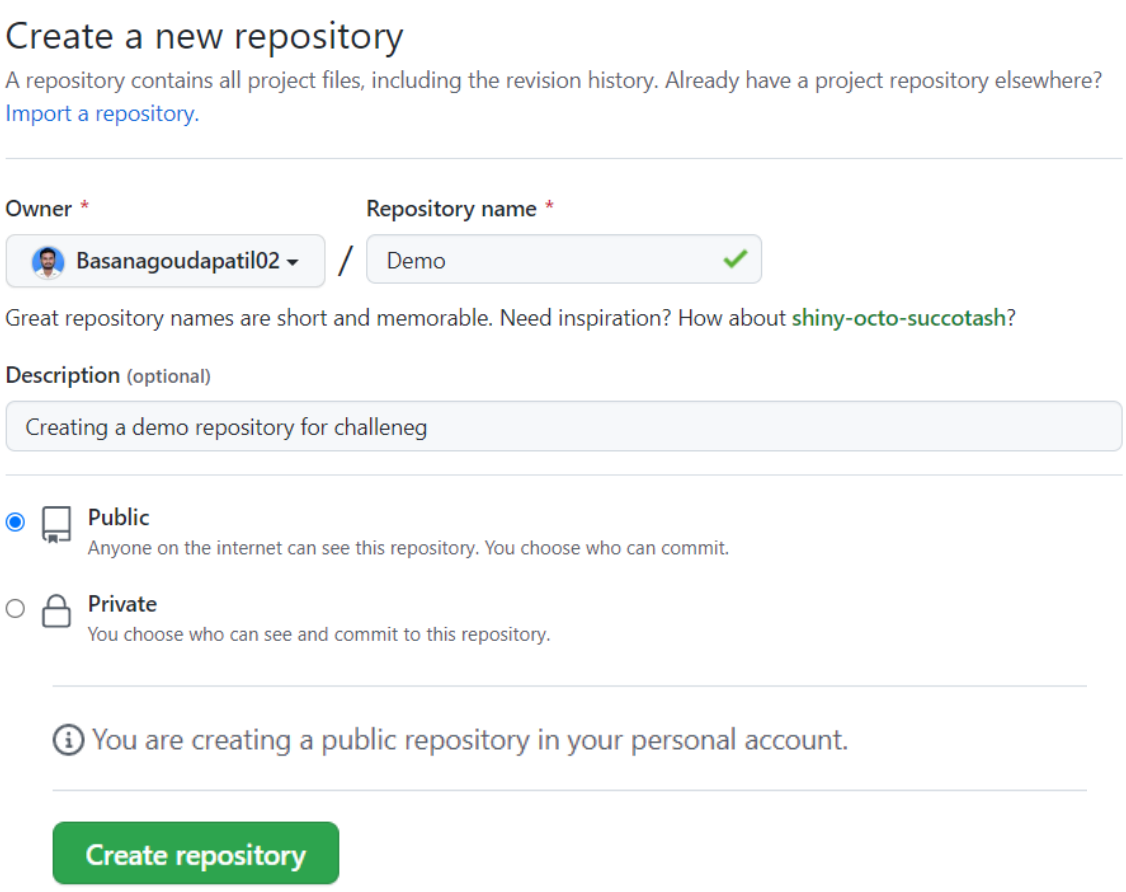
- Create a free account on GitHub (if you don't already have one).

  You can sign up at https://github.com/

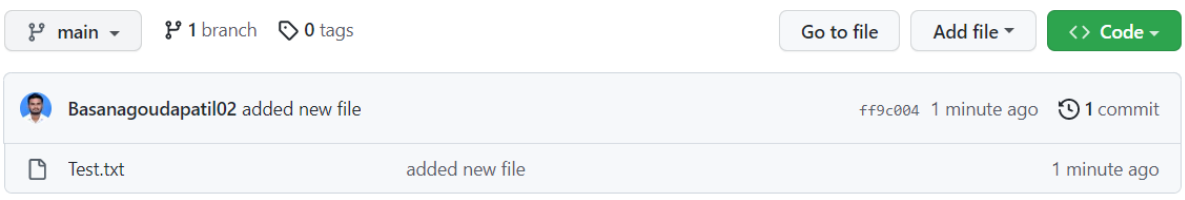
- Create a new repository on GitHub and clone it to your local machine
  Step 1: Installing git in local machine

```
ubuntu@ip-172-31-57-94:~/blog$ git init
hint: Using 'master' as the name for the initial branch. This default branch nam
e
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ubuntu/blog/.git/
```

Step 2: Creating a new repository in Github



- Step 3 : Create a file in github and a clone it to your local machine



Here a new file is creating in github by name as Test.txt , now we will clone it to our local machine using command git clone <HTTP link of test.txt gile from github>

```
ubuntu@ip-172-31-57-94:~/git_demo/git_hub$ git clone https://github.com/Basanag
udapati102/Demo.git
Cloning into 'Demo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

- Make some changes to a file in the repository and commit them to the repository using Git and Push the changes back to the repository on GitHub

  Create a new file in git after creating add it and commit it using command
  git add filename
  git commit -m filename

```
ubuntu@ip-172-31-57-94:~/git_demo/git_hub$ nano 8th_day
ubuntu@ip-172-31-57-94:~/git_demo/git_hub$ git add 8th_day
ubuntu@ip-172-31-57-94:~/git_demo/git_hub$ git commit -m 8th_day
[master (root-commit) 96dd40f] 8th_day
 Committer: Ubuntu <ubuntu@ip-172-31-57-94.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 git_hub/8th_day
ubuntu@ip-172-31-57-94:~/git_demo/git_hub$
```

  In the Command prompt, add the URL for the remote repository where your local repository will be pushed.

```
$ git remote add origin <REMOTE_URL>
# Sets the new remote
$ git remote -v
# Verifies the new remote URL
```

  Push the changes in your local repository to GitHub.com.

```
$ git push origin main
# Pushes the changes in your local repository up to the remote repository
you specified as the origin.
```