What is version control:

Version control is a system that tracks changes made to a file or a set of files over time. It allows developers or teams to collaborate on a project and keep track of different versions of their code. Version control systems (VCS) are commonly used in software development to manage code changes, but they can also be used for any type of files, such as documents or media files.

With version control, every change made to a file is recorded and can be accessed at any time. This allows developers to keep track of the changes they have made and to easily revert to a previous version if necessary. Version control also enables multiple developers to work on the same project simultaneously without overwriting each other's changes.

There are two main types of version control systems: centralized and distributed. Centralized version control systems (CVCS) store files in a central repository, and developers check out and check in files as needed. Distributed version control systems (DVCS) create multiple copies of the repository on each developer's computer, allowing them to work offline and merge changes back to the central repository when they are online. Examples of popular version control systems include Git, Subversion (SVN), and Mercurial.

What is Git:

Git is a distributed version control system (DVCS) that is widely used for software development. It was created by Linus Torvalds in 2005 and is now maintained by the community. Git is free and open-source software that can run on Windows, macOS, Linux, and other operating systems.

Git enables developers to track changes made to their code over time and collaborate with others on a project. With Git, each developer has a complete copy of the project's code on their local machine, which they can work on independently. Developers can create branches to work on specific features or changes without affecting the main codebase. They can then merge their changes back into the main codebase when they are ready.

Git provides a number of powerful features for managing code changes, such as:

- Branching and merging: Git allows developers to create branches of code and merge changes back into the main codebase.
- Distributed development: Git allows developers to work on code offline and synchronize changes with other developers when they are online.
- Commit history: Git keeps a detailed history of all changes made to the code, making it easy to see who made changes and when they were made.

- Staging area: Git provides a staging area where developers can review changes before committing them to the repository.
- Version tagging: Git allows developers to tag specific versions of the code for easy reference.

Git has become one of the most popular version control systems in use today and is used by millions of developers and companies worldwide.

Basic git command:

Configure username and email parameter:
git config --global user.name "mssquareglobal"
git config --global user. Email mssquareglobal.gmail.com
git config –list

Useful git commands:

1. git init: Initializes a new Git repository in the current directory.

```
$ git init
```

2. git status: Shows the status of the working directory and staging area.

```
$ git status
```

3. git add: Adds files or changes to the staging area.

```
$ git add file1.txt
$ git add .
```

4. git commit: Commits changes to the repository.

```
$ git commit -m "Added file1.txt"
```

5. git log: Shows the commit history.

```
$ git log
```

6. git reset: Unstages changes from the staging area.

```
$ git reset file1.txt
```

7. git reset: Reset to specific commit.

```
$ git reset <coommit id>
```

8. git reset, move to specific commit with discarding current chagnes

```
$ git reset --hard <coommit id>
```

9. git branch: Lists all branches in the repository.

```
$ git branch
```

10. git checkout: Switches to a different branch or commit.

```
$ git checkout master
```

11. git merge: Merges changes from one branch into another.

```
$ git merge feature-branch
```

12. git push: Pushes changes to a remote repository.

```
$ git push origin master
```

13. git pull: Pulls changes from a remote repository.

```
$ git pull origin master
```

14. git remote: Lists all remote repositories.

```
$ git remote -v
```

13. git fetch: Downloads changes from a remote repository, but does not merge them.

```
$ git fetch origin
```

14. git diff: Shows the differences between files or commits.

```
$ git diff file1.txt
$ git diff commit1 commit2
```

15. git reset: Unstages changes from the staging area.

```
$ git reset file1.txt
```

16. git revert: Reverts a commit by creating a new commit that undoes the changes.

```
$ git revert commit1
```

17. git tag: Creates a new tag at the current commit.

```
$ git tag v1.0
```

18. git stash: Temporarily saves changes that are not ready to be committed.

```
$ git stash save "Work in progress"
```

19. git remote add: Adds a new remote repository.

```
$ git remote add upstream https://github.com/user/repo.git
```

20. git remote remove: Removes a remote repository.

```
$ git remote remove upstream
```