Hitesh Rochlani
T23 – 136

whExperiment No 3

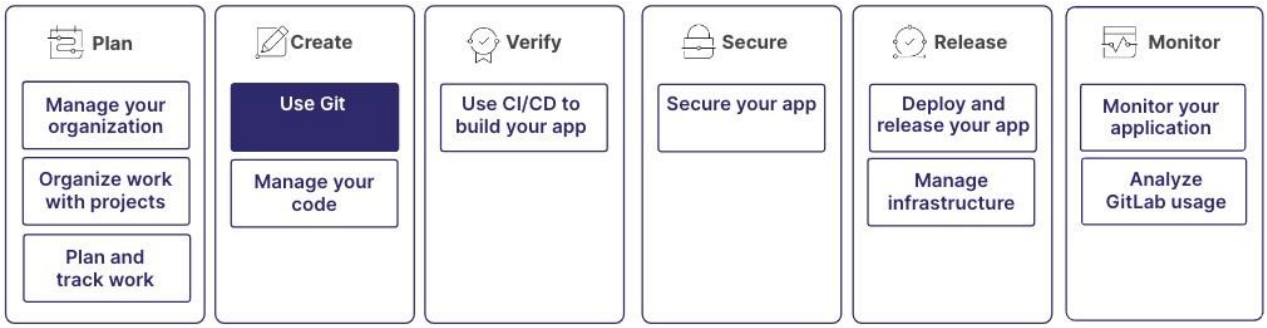
Aim: To perform various Git operations on local and remote repositories using GIT cheat-sheet.

Theory:

Git is a version control system you use to track changes to your code and collaborate with others. GitLab is a web-based Git repository manager that provides CI/CD and other features to help you manage your software development lifecycle.

You can use GitLab without knowing Git. However, it is advantageous to understand Git when you use GitLab for source control.

Learning Git is part of a larger workflow:



Repositories

A Git repository is a directory that contains all the files, folders, and version history of your project. It serves as a central hub where Git manages and tracks changes to your code.

When you initialize a Git repository or clone an existing one, Git creates a hidden directory, .git, inside the project directory. The directory contains all the essential metadata and objects Git uses to manage your repository, including the complete history of all changes made to the files. Git tracks changes at the file level, so you can view the modifications made to individual files over time.

For more information, see Repositories.

Working directories

Your working directory is where you make changes to your code. When you clone a Git repository, you create a local copy of the repository in your working directory. You can edit files, add new ones, and test your code. To collaborate, you can:

- **Commit**: After you make changes in your working directory, commit those changes to your local repository.
- **Push**: Push your changes to a remote Git repository hosted on GitLab. This makes your changes available to other team members.
- **Pull**: Pull changes made by others from the remote repository, and ensure that your local repository is updated with the latest changes.

For more information, see Common Git commands.

Branches

In Git, you can use branches to work on different features, bug fixes, or experiments simultaneously without interfering with each other's work. Branching enables you to create an isolated environment where you can make and test changes without affecting the default branch. In GitLab, the default branch is usually called main.

Merge a branch

After a feature is complete or a bug is fixed, you can merge your branch into the default branch. You can do this in a Merge request. Merging is a safe way to bring changes from one branch into another while preserving the history of the changes.

If there are conflicts between the branches, for example, if you modify the same lines of code in both branches, GitLab flags these as merge conflicts. These must be resolved manually by reviewing and editing the code.

Delete a branch

After a successful merge, you can delete the branch if it is no longer needed. Deleting unnecessary branches helps keep your repository organized and manageable.

To ensure no work is lost, verify all changes are incorporated into the default branch before you delete the branch after the final merge.

For more information, see Branches.

Understand the Git workflow

You can manage your code, collaborate with others, and keep your project organized with a Git workflow. A standard Git workflow includes the following steps:

1. **Clone a repository**: Create a local copy of the repository by cloning it to your machine. You can work on the project without affecting the original repository.
2. **Create a new branch**: Before you make any changes, it's recommended to create a new branch. This ensures that your changes are isolated and don't interfere with the work of others on the default branch.
3. **Make changes**: Make changes to files in your working directory. You can add new features, fix bugs, or make other modifications.
4. **Stage changes**: After you make changes to your files, stage the changes you want to commit. Staging tells Git which changes should be included in the next commit.
5. **Commit changes**: Commit your staged changes to your local repository. A commit saves a snapshot of your work and creates a history of the changes to your files.

6. **Push changes**: To share your changes with others, push them to the remote repository. This makes your changes available to other collaborators.
7. **Merge your branch**: After your changes are reviewed and approved, merge your branch into the default branch. For example, main. This step incorporates your changes into the project.

Forks

Some organizations, particularly those working with open-source projects, may use different workflows. For example, Forks.

A fork is a personal copy of the repository that exists in your own namespace. Use this workflow when contributing to open-source projects or when your team uses a centralized repository.

Install Git

To use Git commands and contribute to GitLab projects, you should download and install the Git client on your computer.

The installation process varies depending on your operating system. For example, Windows, MacOS, or Linux. For information on how to install Git, see Install Git.

Git commands

To interact with Git from the command line, you can use Git commands:

- git clone: Clone a repository to your local machine.
- git branch: List, create, or delete branches in your local repository.
- git checkout: Switch between different branches in your local repository.
- git add: Stage changes for commit.
- git commit: Commit staged changes to your local repository.
- git push: Push local commits to the remote repository.
- git pull: Fetch changes from the remote repository and merge them into your local branch.

For more comprehensive information and detailed explanations, see Command Git commands guide.

Output:

Hitesh Rochlani
T23 – 136

```
AI&DS203@203-20 MINGW64 ~ (main)
$ cd Documents/

AI&DS203@203-20 MINGW64 ~/Documents (main)
$ mkdir T23_136_SEPM

AI&DS203@203-20 MINGW64 ~/Documents (main)
$ cd T23_136_SEPM/

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM (main)
$ mkdir git-dvcs

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM (main)
$ cd git-dvcs/
```

```
AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ git config --globaluser.name: "hitesh-rochlani"^C

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ git config --global user.name: "hitesh-rochlani"
error: invalid key: user.name:

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ ^C

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ ^[[200~git config --global user.name "hitesh-rochlani"
bash: $'\E[200~git': command not found

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ git config --global user.name "hitesh-rochlani"

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ git config --global user.email "hiteshrochlani11@gmail.com"

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ git config --global --list
user.name=hitesh-rochlani
user.email=hiteshrochlani11@gmail.com

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ cat ~/.gitconfig
[user]
        name = hitesh-rochlani
        email = hiteshrochlani11@gmail.com

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ mkdior git-demo-project
bash: mkdior: command not found

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ mkdir git-demo-project

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs (main)
$ cd git-demo-project/

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs/git-demo-project (main
)
$ git init
Initialized empty Git repository in C:/Users/AI&DS203/Documents/T23_136_SEPM/git
-dvcs/git-demo-project/.git/

AI&DS203@203-20 MINGW64 ~/Documents/T23_136_SEPM/git-dvcs/git-demo-project (mast
er)
$ ls -a
./  ../  .git/
```

Hitesh Rochlani
T23 – 136