# WEEK-8

1. **Hands-on: Git Basics**

- **git –-version**

  git version 2.46.2.windows.2

- **git config –global user.name “username”**

  git config –global user.name “pranavi”

- **git config –global user.email “email”**

  git config –global user.email “aspranavi111-@gmail.com”

- **git config –global –list**

  git config –global –list

  user.name=pranavi

  user.email= aspranavi1110@gmail.com

- **Add a file to source code repository**

  $ mkdir GitDemo

  $ cd GitDemo

  $ git init

  Initialized empty Git repository in C:/Users/HP/GitDemo/.git/

  $ ls -a

  ./  ../  .git/

  $ ls -al

  total 32

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:14 ./

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:14 ../

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:14 .git/

  $ echo "Welcome to Git Version Control" > welcome.txt

  $ ls -al

  total 33

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:17 ./

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:14 ../

  drwxr-xr-x 1 HP 197609 0 Aug  9 13:14 .git/

  -rw-r--r-- 1 HP 197609 40 Aug  9 13:17 welcome.txt

```
$ cat welcome.txt
Welcome to Git Hands-On Version Control

$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        welcome.txt

nothing added to commit but untracked files present (use "git add" to track)

$ git status
On branch master

No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   welcome.txt
```

**2. Hands-On: Implementing .gitignore**

- Explain git ignore

  Git ignore is a feature in Git that allows you to specify files and directories that you don't want to track or commit to your repository. It's a way to tell Git to simply ignore these files, so they won't show up as "untracked files" when you run git status.

- Explain how to ignore unwanted files using git ignore

  A simple way to tell Git to ignore certain files and directories is to create a .gitignore file in the root of your repository. In this file, you can list the names of the files and folders you want Git to disregard. For example, to ignore all files ending with .log, you would add *.log to a new line in your .gitignore file. To ignore an entire directory named build, you would simply add build/. You can also ignore a specific file by writing its name, like .env. Git will then ignore these files, preventing them from appearing as "untracked" and ensuring they aren't accidentally committed to your repository.

  ```
  touch myapp.log
  mkdir log
  touch log/debug.log
  git status
  ```

On Branch Master

No Commits Yet

Changes yet to be committed:
(use “git rm --cached <file>.. ” to unstage)

new file: myapp.log

```
*.log
log/
git add .gitignore
git commit -m "Add .gitignore to ignore log files and folders"
git push origin master
```

### 3. Hands-On: Introduction to Git Branching and Merging

- Explain Branching and Merging

  Branching is a core concept in Git that allows developers to work on new features, bug fixes, or experiments without affecting the main project code. It creates a separate line of development from the main codebase. Think of it like making a copy of a document to edit, while the original remains untouched. This enables multiple people to work on different things simultaneously without causing conflicts. Merging is the process of combining those separate lines of development back into a single branch. Once a feature or fix is complete and tested on its own branch, it can be merged back into the main branch, integrating all the changes.

- Explain about creating a Branch request in GitLab

  To create a new branch in GitLab, you navigate to your repository's main page. From there, you can use the "Branches" tab to see all existing branches. To make a new one, you can either click the "New branch" button or, more commonly, navigate to the "Repository" > "Files" view. At the top of the file list, you'll see a dropdown menu showing the current branch. You can type the name of your new branch in the search bar of this dropdown and select "Create branch: [your branch name]". This creates a new branch based on the one you were on, usually the main branch.

- Explain about creating a Merge request in GitLab

  A merge request (MR) is a formal proposal to merge a branch into another, typically the main one. It's a way to let others know you've completed your work and want to integrate it. To create one, you first need to have a branch with your changes pushed to GitLab. You can then navigate to the "Merge Requests" tab on the left sidebar and click "New merge request." GitLab will automatically compare your branch to the main branch. In the form that follows, you'll be able to give your merge request a title and a description, explaining the purpose of your changes. You can also assign it to specific team members for review and add labels or milestones. Submitting the merge request notifies others that your work is ready to be reviewed, discussed, and eventually, merged.

$ git init

Reinitialized existing Git repository in C:/Users/HP/GitDemo/.git/

$ echo "This is the initial content of my project." > README.md

$ git add README.md

warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it

$ cat README.md

This is the initial content of my project.

$ git commit -m "Initial commit of Project"

[master (root-commit) 06a77d5] Initial commit of Project

2 files changed, 2 insertions(+)

create mode 100644 README.md

create mode 100644 welcome.txt

$ git branch GitNewBranch

$ git checkout GitNewBranch

Switched to branch 'GitNewBranch'

$ echo "This file was added on a new branch." > feature.txt

$ git add feature.txt

warning: in the working copy of feature.txt', LF will be replaced by CRLF the next time Git touches it

$ git commit -m "Added a new Feature"

[GitNewBranch f2df54b] Added a new Feature

1 file changed, 1 insertion(+)

create mode 100644 feature.txt

$ git checkout master

Switched to branch 'master'

$ git merge GitNewBranch

Updating 06a77d5..f2df54b

Fast-forward

feature.txt | 1 +

1 file changed, 1 insertion(+)

create mode 100644 feature.txt

$ git log --oneline --graph

* f2df54b (HEAD -> master, GitNewBranch) Added a new Feature

* 06a77d5 Initial commit of Project

$ git branch -d GitNewBranch

Deleted branch GitNewBranch (was f2df54b).

**4. Hands-On: Understanding and Resolving Git Merge Conflicts**

- Explain how to resolve the conflict during merge.

To resolve a merge conflict, you first need to identify the files that Git has marked as having a conflict. When a merge fails, Git inserts special markers (<<<<<<<, =======, and >>>>>>>) into the conflicted files to show the competing changes. You then manually open these files and edit them to remove the markers, choosing which changes you want to keep. You can select the code from your current branch, the code from the branch you're merging, or a combination of both. Once you've manually resolved all the conflicts and the file looks correct, you save the file, stage it with git add, and then complete the merge by creating a new commit with git commit. This final commit records the resolution and concludes the merge process.

git status

git branch GitWork

git checkout GitWork

echo "<hello>Hello, GitWork Branch!</hello>" > hello.xml

echo "<hello>Hello, updated GitWork Branch!</hello>" > hello.xml

git status

git add hello.xml

```
git commit -m "Added and updated hello.xml on GitWork branch"

git checkout master

echo "<hello>Hello, Master Branch!</hello>" > hello.xml

git add hello.xml

git commit -m "Added hello.xml to master"

git log --oneline --graph --decorate --all

git diff master GitWork

git difftool master GitWork

git merge GitWork

git mergetool

git commit -m "Merged GitWork into master and resolved conflicts"

git status

echo "*.orig" >> .gitignore

git add .gitignore

git commit -m "Added .orig files to .gitignore"

git branch

git branch -d GitWork

git log --oneline --graph --decorate
```

## 5. Hands-On: Pushing and Cleaning Up Your Git Repository

- **Explain how to clean up and push back to remote Git**

  To clean up your local Git repository and push your changes to a remote server, you'll typically follow a few steps. First, ensure your working directory is clean by checking the status with git status. You should have no uncommitted changes. Then, you'll want to remove any local branches that have already been merged into the main branch. You can do this with git branch -d <branch_name>. To fetch the latest changes from the remote repository without merging them, run git fetch origin. This updates your

remote-tracking branches. After fetching, you can rebase your local branch on top of the latest remote changes with git rebase origin/main (or origin/master), which creates a cleaner, more linear history than a merge. Finally, to share your cleaned-up commits with the remote repository, you push your changes using git push origin <branch_name>. If you rebased, you might need to use git push --force-with-lease if others are also working on the same branch to avoid overwriting their work.

git status

git branch

git pull origin master

git checkout Git-T03-HOL_002

git push origin Git-T03-HOL_002