|  |  |
| --- | --- |
| EX. No: 2 | **VERSION CONTROLLING IN SOFTWARE APPLICATION DEVELOPMENT** |

**What is a “version control system”?**

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.

**Types of Version Control Systems:**

* Local Version Control Systems
* Centralized Version Control Systems
* Distributed Version Control Systems

**Three important steps of version control:**

* **git add** changed files to version control tracking.
* **git commit** the changed files to create a unique snapshot of the local repository.
* **git push** those changed files from the local copy of a repository to the cloud

**Check the Status of Changes Using GIT Status**

Once you start working, you can use the **git status** command to check what changes are being identified by **git**.

To practice working with this command, use the **terminal** to navigate to your git practice repository:

$ cd practice-git-skillz

Next, run git status.

$ git status

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Notice that when you run git status it returns: **working tree clean**. This means that there are no changes to any files in your repo - YET.

Next, open and make a small change to the README.md file in a text editor. Then, run the command git status to check that changes have been made to your file(s).

git status

On branch main

Your branch is up-to-date with 'origin/main'.

Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

The output from the git status command above indicates that you have modified a file (e.g. README.md) that can be added to version control.

**Adding and Committing File Changes to Version Control**

To keep track of changes to this file using **git**, you need to:

1. first git add the changes to tracking (or staging area), and then
2. git commit the changes to version control.

These two commands make up the bulk of many workflows that use **git** for version control:

* git add: takes a modified file in your working directory and places the modified version in a staging area for review.
* git commit: takes everything from the staging area and makes a permanent snapshot of the current state of your repository that has a unique identifier.

Add Changed Files Using git add

After making changes, you can add either an individual file or groups of files to version control tracking. To add a single file, run the command:

git add file-name.extension

For example, to add the README.md file, you would use:

git add README.md

You can also add all of the files that you have edited at the same time using:

git add .

Commit Changed Files Using git commit

Once you are ready to make a snapshot of the current state of your repository (i.e. move changes from staging area), you can run git commit. The git commit command requires a commit message that describes the snapshot (i.e. changes) that you made in that commit.

A commit message should outline what changed and why. These messages:

1. help collaborators and your future self understand what was changed and why.
2. allow you and your collaborators to find (and undo if necessary) changes that were previously made.

When you are not committing a lot of changes, you can create a short one line commit message using the -m flag as follows:

git commit -m "Update title and author name in homework for week 3"

|  |  |
| --- | --- |
| **EX. NO: 3** | **SETUP AND CONFIGURATION OF GIT IN LOCAL MACHINE.** |

To set up and configure Git on your local machine, you can follow these steps:

**Step 1: Install Git**

- Download the Git installer from the official Git website.

- Run the installer and follow the step-by-step instructions in the setup wizard.

- Make sure to select the appropriate options during the installation process.

**Step 2: Configure Git**

- Open a terminal or command prompt.

- Set your username and email address by running the following commands:

**git config --global user.name "Your Name"**

**git config --global user.email "your.email@example.com"**

  Replace "Your Name" with your desired username and "your.email@example.com" with your email address.

**Step 3: Set up the default branch name (optional)**

- By default, Git creates a branch called "master" when you initialize a new repository with `git init`.

- If you prefer to use a different name for the initial branch, you can set it using the following command:

**git config --global init.defaultBranch main**

  This command sets the default branch name to "main". Replace "main" with your preferred branch name if desired.

**Step 4: Verify your Git configuration**

- To check your Git configuration settings, you can use the following command:

**git config --list**

This command will display a list of your Git configuration settings, including your username and email address.

These steps should help you set up and configure Git on your local machine. Remember to replace "Your Name" and "your.email@example.com" with your own information.

|  |  |
| --- | --- |
| **Ex. No: 4** | **WORKING WITH REMOTE REPOSITORIES** |

When working with remote repositories in Git, there are several commands and actions you can perform.

**Cloning a Remote Repository**

* The **git clone** command is used to create a local copy of a remote repository on your machine.

To clone a remote repository, use the following command

**git clone <remote-url>**

* Replace **<remote-url>** with the URL of the remote repository.
* For example, to clone a repository hosted on GitHub, you can use

**git clone** [**https://github.com/username/repository.git**](https://github.com/username/repository.git)

#### Adding a Remote Repository

* If you have an existing local repository and want to connect it to a remote repository, you can use the git remote add command.
* To add a remote repository, use the following command:

**git remote add <remote-name> <remote-url>**

* Replace <remote-name> with a name for the remote repository (e.g., "origin") and <remote-url> with the URL of the remote repository.
* For example

**git remote add origin** [**https://github.com/username/repository.git**](https://github.com/username/repository.git)

#### Pushing Changes to a Remote Repository

After making changes to your local repository, you can push those changes to the remote repository using the git push command.

To push changes to a remote repository, use the following command:

**git push <remote-name> <branch-name>**

Replace <remote-name> with the name of the remote repository (e.g., "origin") and <branch-name> with the name of the branch you want to push.

For example:

**git push origin main**

#### Pulling Changes from a Remote Repository

* To update your local repository with the latest changes from the remote repository, you can use the git pull command.
* To pull changes from a remote repository, use the following command:

**git pull <remote-name> <branch-name>**

* Replace <remote-name> with the name of the remote repository (e.g., "origin") and <branch-name> with the name of the branch you want to pull.
* For example

**git pull origin main**

These are some basic commands for working with remote repositories in Git. Remember to replace <remote-url>, <remote-name> and <branch-name> with the appropriate values for your specific repository.

|  |  |
| --- | --- |
| **EX.No: 5** | **WORKING WITH COLLABORATIVE REPOSITORY MANAGEMENT USING GIT** |

Working with collaborative repository management using Git involves collaborating with other developers and effectively managing changes to a shared codebase. Here are some key practices and concepts to consider:

**Forking a Repository**

- When you want to contribute to a project hosted on a remote repository, it's common to start by forking the repository.

- Forking creates a personal copy of the repository under your GitHub account or another hosting platform, where you can freely make changes without affecting the original repository.

- To fork a repository on GitHub, navigate to the repository's page and click the "Fork" button.

**Cloning a Forked Repository**

- After forking a repository, you need to clone the forked repository to your local machine to start making changes.

- Use the `git clone` command, as explained in a previous answer, to clone the forked repository.

**Adding an Upstream Remote**

- The original repository that you forked is known as the "upstream" repository. It represents the source of truth for the project.

- To synchronize your forked repository with the latest changes from the upstream repository, you can add an "upstream" remote.

- Use the `git remote add` command to add the upstream remote URL. For example:

**git remote add upstream <upstream-url>**

**Keeping Your Forked Repository Up to Date**

- To incorporate the latest changes from the upstream repository into your forked repository, follow these steps:

  - Fetch the latest changes from the upstream remote: **`git fetch upstream`**

  - Checkout your local main branch: **`git checkout main`**

  - Merge the changes from the upstream/main branch into your local main branch: **`git merge upstream/main`**

  - Push the updated main branch to your forked repository: **`git push origin main`**

**Creating Branches for Collaborative Work**

- When working on a collaborative project, it's common to create branches for new features, bug fixes, or other changes.

- Use the `git branch` command to create a new branch: **`git branch <branch-name>`**

- Switch to the newly created branch using `git checkout`: **`git checkout <branch-name>`**

- Alternatively, you can create and switch to a new branch in one command**: `git checkout -b <branch-name>`**

**Pushing Changes and Creating Pull Requests**

- Once you've made changes on a branch, you can push the branch to your forked repository using **`git push origin <branch-name>`.**

- After pushing the branch, you can create a pull request on the upstream repository to propose your changes for merging into the main codebase.

- On the upstream repository's page, find the "Pull requests" section and click the "New pull request" button.

- Select the appropriate branches for the base (usually main) and compare (your branch) branches.

- Provide a title and description for your pull request, then click "Create pull request" to submit it.

**Reviewing and Merging Pull Requests**

- Collaborators or maintainers of the upstream repository can review and comment on your pull request.

- They may request changes or provide feedback before merging the changes.

- Once the pull request is approved, it can be merged into the main branch of the upstream repository.

|  |  |
| --- | --- |
| **EX. NO: 6** | **IMPLEMENTATION OF CORE REVIEW IN GIT** |

**Introduction**

Git is a powerful tool for version control, but it’s also a crucial tool forcollaborating with remote teams. When working remotely, effectivecommunication and collaboration are key to ensure that your teamis workingtogether towards a common goal.

**Use a shared repository**

Using a shared repository is the foundation of effective collaborationwithGit. By hosting your repository on a remote server, you can give your teammembers access to the latest version of your code, regardless of their location.

Create a new repository: **git init**

Clone a repository: **git clone [respository URL]**

Add a remote repository: **git remote add [name] [repository URL]** Fetch changes from a remote repository: **git fetch** Pull changes from a remote repository**: git pull**

Push changes to a remote repository: **git push Use branching and merging**

Branching and merging are powerful features of Git that allowyoutowork on multiple versions of your code simultaneously, without interferingwitheach other’s work. By using branching and merging effectively, your teamcanwork on different features or bug fixes in parallel, without causing conflicts.

Create a new branch: **git branch [name]**

Switch to a branch: **git checkout [name]**

Merge a branch: **git merge [name]**

Resolve merge conflicts: **git mergetool**

Delete a branch: **git branch -d [name]**

**Use Pull requests**

Pull requests are a great way to review and merge changes fromdifferent team members before they are merged into the main codebase. By usingpull requests, you can ensure that your code is reviewed and tested before it is merged into the main branch, which can help prevent bugs and other issues.

Create a new pull request: **git request-pull [branch] [repository URL]** Review and merge a pull request**: git pull-request**

Close a pull request: **git request-pull -C [branch] [repository URL]**

**Use Issue Tracking**

Issue tracking is a great way to keep track of bugs, feature requests, andother issues that need to be addressed in your code. By using an issue trackingsystem like GitHub Issues, you can assign tasks to teammembers, trackprogress, and ensure that all issues are addressed in a timely manner.

Create a new issue: **git commit -m “[issue number] [commit message]”**Assign an issue to a team member: **git assign [username]** Clone an issue: **git clone [issue number]**

Reopen an issue: **git reopen [issue number]**

**Communicate Effectively**

Effective communication is key to collaboration, especially inaremoteteam setting. Use tools like Slack or Microsoft Teams to stay in touchwithyourteam members, and use video calls or screen sharing to discuss code changesorwork on problems together.

Start a video call: **git video-call [username]**

Share your screen**: git screen-share**

Send a message: **git message [username] [message]**

**Use Git hooks**

Git hooks are scripts that Git runs automatically at certain points intheGitworkflow. You can use Git hooks to automate repetitive tasks, enforcecodingstandards, or perform other tasks that are important to your team’s workflow.

Install a git hook: **git init [hook name]**

Write a git hook script: **nano.git/hooks/[hook name]** Make the Git hook script executable: **chmod +x .git/hooks/[hook name] Use Git submodules**

Git submodules are repositories that are embedded insideotherrepositories. You can use Git submodules to manage dependencies, or toincludeshared code in multiple projects.

Add a Git submodule: **git submodule add [repository URL]** Update a Git submodule: **git submodule update**

Remove a Git submodule: **git submodule deinit [submodule path]**

**Use Git aliases**

Git aliases are shortcuts for commonly used Git commands. YoucanuseGit aliases to save time and improve your productivity when working withGit.

Set up a Git alias: **git config –global alias.[alias name] ‘[Git command]’** Use a Git alias: **git [alias name]**