|  |  |  |
| --- | --- | --- |
| **Ex.No: 1** | **INSTALLATION OF GIT AND CREATING REPOSITORIES** | **REG NO:**  **URK23CS1023** |
| **DATE:05/01/2024** |

# Aim:

# Efficiently install Git and establish repositories for streamlined version control and collaborative development

# Theory:

The installation of Git and creation of repositories are foundational steps in setting up a robust version control system for software development projects. Git, a distributed version control system, can be installed on various operating systems including Windows, macOS, and Linux, typically through package managers or by downloading and running installer packages. Once installed, Git provides a powerful set of commands for managing source code, enabling developers to track changes, collaborate efficiently, and maintain a history of project modifications. Creating repositories involves initializing a directory with Git, effectively establishing it as a repository where project files and their revisions are stored. With repositories in place, developers can start tracking changes, branching for feature development, merging changes, and collaborating effectively with team members, ensuring project integrity and facilitating seamless development workflows.

Top of Form

# Introduction of GIT

Git is one of the ways of implementing the idea of version control. It is Distributed Version Control System.

**Step 1: Installing GIT**

Before you start using Git, you have to make it available on your computer.

it’s already installed, it’s probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself.

# Step 2: Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to <https://git-scm.com/download/win> and the download will start automatically

To get an automated installation you can use the [Git Chocolatey package.](https://community.chocolatey.org/packages/git)

The easiest way to get Git is to download the executable from [the Git website](https://git-scm.com/download/win).

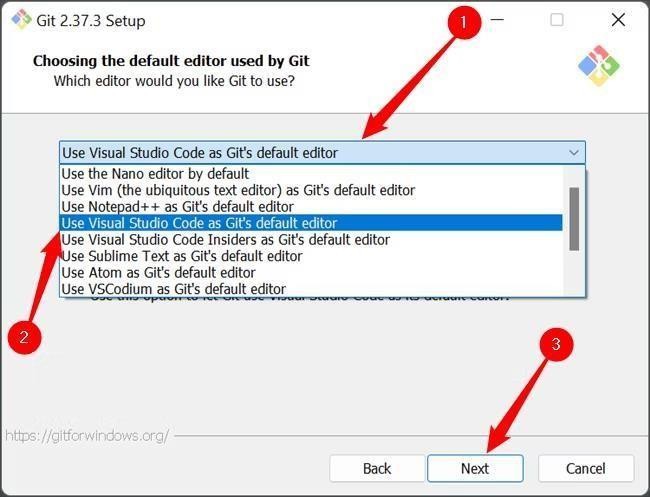
Click "64-bit Git for Windows Setup" to start the [download](https://www.howtogeek.com/771533/where-are-my-downloads-on-windows/), and then wait a moment — the download is only about 50 megabytes, so it shouldn't take very long.



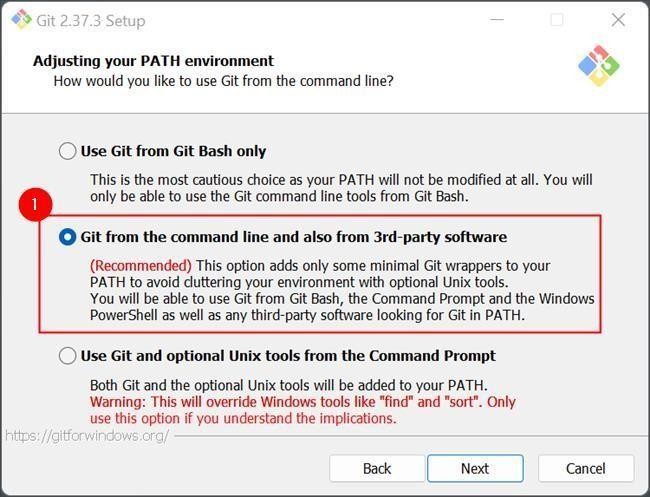
Double-click the executable you just [downloaded](https://www.howtogeek.com/771533/where-are-my-downloads-on-windows/), then click "Next" to move through the installation prompts.

The first is the text editor Git will use. The default selection is Vim. Vim is ubiquitous and a hallmark of command-line interfaces everywh17ere but learning to use its idiosyncratic

commands can be daunting. You should probably pick something else instead, like Visual Studio Code, Sublime, NotePad++, or any other [plain text editor](https://www.howtogeek.com/795509/why-you-need-a-plain-text-editor/) you like.



The second is the way Git integrates itself into your PC's [PATH.](https://www.howtogeek.com/118594/how-to-edit-your-system-path-for-easy-command-line-access/) Make sure that the "Git From The Command Line And Also From 3rd-Party Software" is selected.



Click through the remaining options, and wait for everything to finish downloading. The time requires to download everything will vary depending on what you chose to install. The default selection results in a download that is about 270 megabytes.

# Step3: Managing Private and Public RepositoryMaking a repository Private

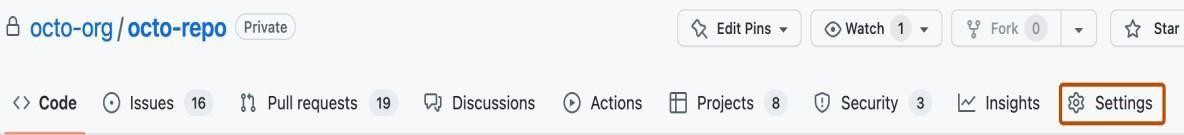
* GitHub will detach public forks of the public repository and put them into a new network. Public forks are not made private.
* If you're using GitHub Free for personal accounts or organizations, some features won't be available in the repository after you change the visibility to private. Any published GitHub Pages site will be automatically unpublished. If you added a customdomain to the GitHub Pages site, you should remove or update your DNS records before making the repository private, to avoid the risk of a domain takeover.
* GitHub will no longer include the repository in the GitHub Archive Program.
* GitHub Advanced Security features, such as code scanning, will stop working.

# Step 4 :Making a repository Public

* GitHub will detach private forks and turn them into a standalone private repository.
* If you're converting your private repository to a public repository as part of a move toward creating an open source project.
* Once your repository is public, you can also view your repository's community profile to see whether your project meets best practices for supporting contributors.
* The repository will automatically gain access to GitHub Advanced Security features.

# Step 5: Changing a repository’s Visibility

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**. If you cannot see the "Settings" tab, select the dropdown menu, then click **Settings**.



1. In the "Danger Zone" section, to the right of to "Change repository visibility", click **Change visibility**.
2. Select a visibility.
3. To verify that you're changing the correct repository's visibility, type the name of the repository you want to change the visibility of.

# Click I understand, change repository visibility.

**Step 6: Pushing changes to the repository**

1. On Github.com, ensure your repository is public.
2. Clone the repository to your local machine.
   1. Open a terminal on your local machine.
   2. Use the following command to clone the repository to your local machine. Replace 'your- username' and 'your-repo-name' with your GitHub username and repository name

*git clone* [*https://github.com/your-username/your-repo-name.git*](https://github.com/your-username/your-repo-name.git)

* 1. Move into the newly cloned repository

*cd your-repo-name*

1. Make a change to the README file (Create a README.md file if it does not exist).
2. Commit the change.
   1. Add the modified README file to the staging area using the following command:

*git add README.md*

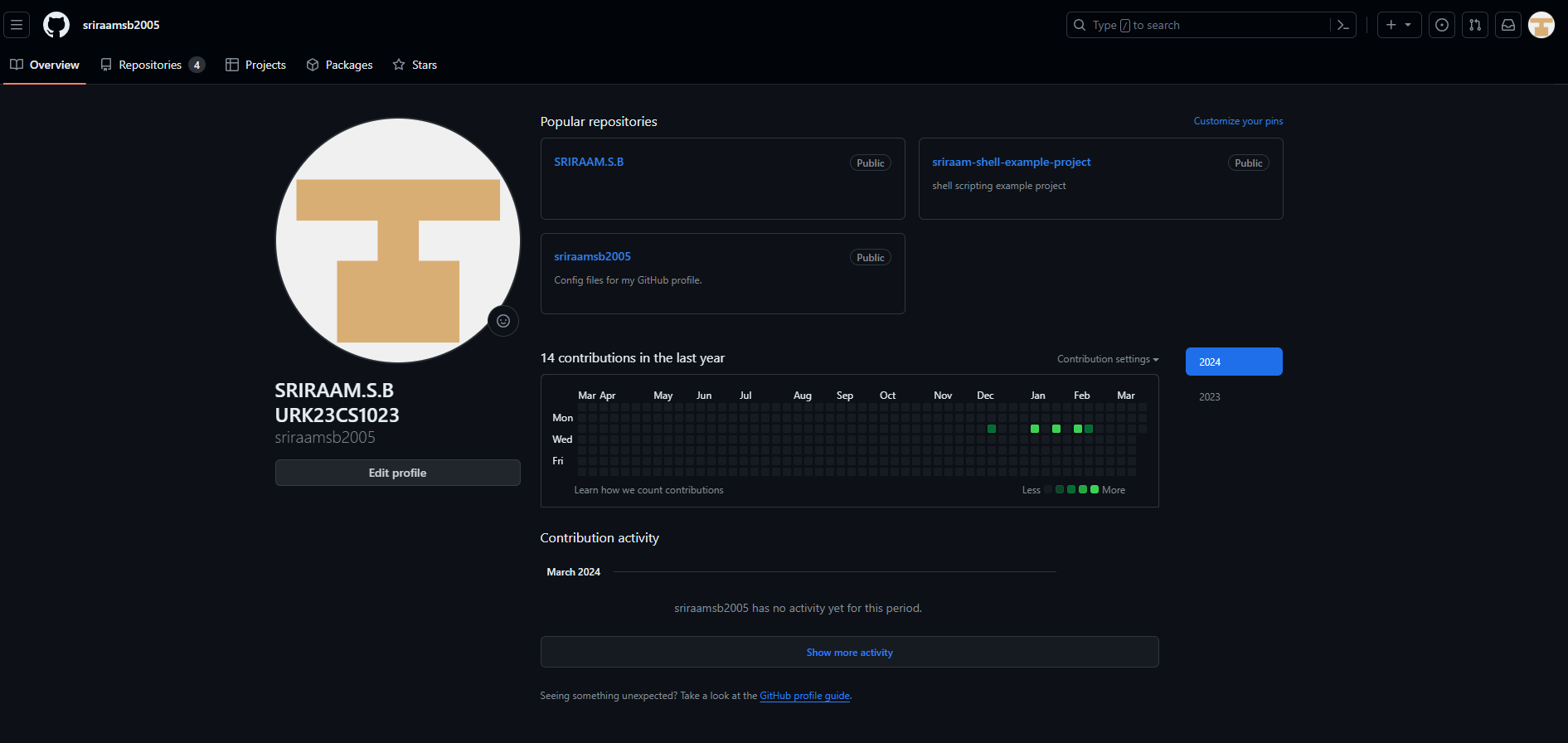
* 1. Commit the changes with a meaningful commit message:

*git commit -m "Update README file"*

1. Push it to the GitHub repository using the following command. If your main branch is called ‘master’, then replace ‘main’ with ‘master’.

*git push origin main*

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **EX. NO: 2** | **WORKING WITH REMOTE REPOSITORIES**  **ON LOCAL MACHINE.** | **REG NO:**  **URK23CS1023** |
| **DATE: 19/12/23** |

**Aim:**

Effectively manage and collaborate on remote repositories from the convenience of your local machine, enhancing productivity and teamwork in software development projects.

**Theory:**

Working with remote repositories on a local machine involves integrating the collaborative features of Git into the developer's workflow, facilitating seamless collaboration and version control across distributed teams. By configuring remote repositories on the local machine, developers can clone existing repositories hosted on platforms like GitHub, GitLab, or Bitbucket, enabling them to pull the latest changes, push their contributions, and synchronize their work with the central repository. This approach streamlines collaboration, allowing developers to access project files, contribute code, and manage branches locally, while ensuring that their changes are efficiently communicated and integrated with the broader development effort. Additionally, working with remote repositories on a local machine enhances flexibility, enabling developers to work offline, experiment with new features, and iterate on code changes before pushing them to the central repository, thereby optimizing the development process and fostering a more agile and collaborative environment.

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](mailto:your.email@example.com)"

Replace "Your Name" with your desired username and ["your.email@example.com"](mailto: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*

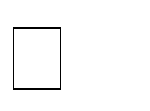
This command will display a list of your Git configuration settings, including your username and email address.

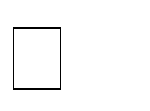
These steps should help you set up and configure Git on your local machine. Remember to replace "Your Name" and ["your.email@e](mailto:your.email@example.com)x[ample.com"](mailto:your.email@example.com) with your own information.

Working with Remote Repository

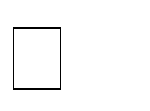
When working with remote repositories in Git, there are several commands and actions you can perform.

# Cloning a Remote Repository

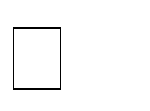
 The **git clone** command is used to create a local copy of a remoterepository on your machine.

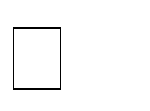
To clone a remote repository, use the following command git clone <remote-url>

Replace **<remote-url>** with the URL of the remote repository.

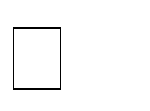
 For example, to clone a repository hosted on GitHub, you can use

**git clone** [**https://github.com/username/repository.git**](https://github.com/username/repository.git) **Adding a Remote Repository**

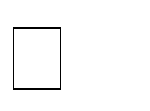
 If you have an existing local repository and want to connect it to a remote repository, you can use the git remote add command.

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 tothe remote repository using the git push command.

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

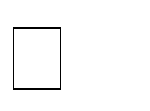
*git push <remote-name> <branch-name>*

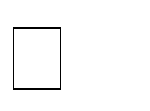
Replace <remote-name> with the name of the remote repository (e.g., "origin") and <branch- name> with the name of the branch you want to push.

For example:

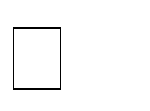
# git push origin main

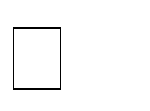
**Pulling Changes from a Remote Repository**

 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>

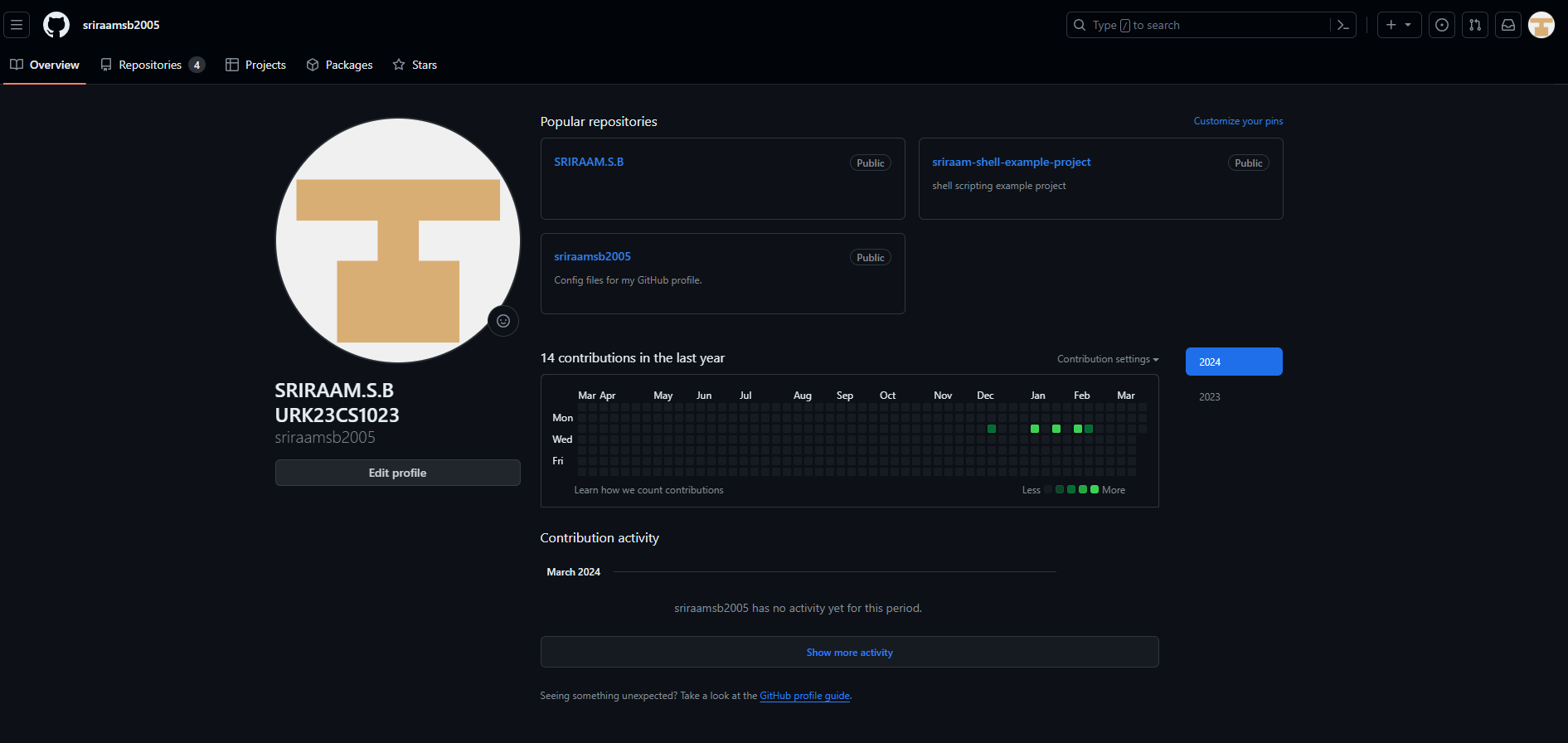
 Replace <remote-name> with the name of the remote repository (e.g., "origin") and <branch-name> with the name of the branch you want to pull.

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.

**Ouput:**

****

**RESULT:**

**To set up and configure git on your local machinecompleted.**

|  |  |  |
| --- | --- | --- |
| **EX NO:3** | **VERSION CONTROL IN SOFTWARE APPLICATION**  **DEVELOPMENT** | **REG NO:**  **URK23CS1023** |
| **DATE:**  **9/1/2024** |

**Aim:**

Implement effective version control practices to enhance collaboration, track changes, and maintain code integrity throughout the software development lifecycle.

**Theory:**

Version control is a crucial aspect of software application development, providing a systematic approach to managing changes to source code, documents, and other project assets. It involves the use of specialized tools, such as Git, Subversion, or Mercurial, to track modifications, maintain a history of revisions, and facilitate collaboration among developers. By adopting version control, development teams can effectively manage concurrent work on the same codebase, preventing conflicts and ensuring that changes are applied in a controlled manner. This enables developers to work on different features or bug fixes simultaneously, without risking the integrity of the overall codebase. Moreover, version control systems allow for easy identification and rollback of changes if necessary, providing a safety net for experimentation and troubleshooting. Additionally, version control facilitates collaboration by providing a centralized repository where developers can share code, review changes, and provide feedback. Ultimately, version control plays a pivotal role in ensuring the stability, reliability, and maintainability of software applications, thereby contributing to the success of development projects.

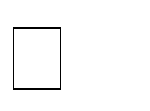
## What is a “version control system”?

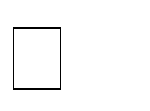
Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

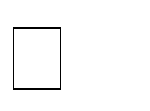
## Types of Version Control Systems:

* + 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 being identified by **git**.

**git status** command to check what changes are

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

*$ cd practice-git-skillz*

Next, run git status.

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:

Notice that when you run git status it returns: **working tree clean**. This means that there areno changes to any files in your repo - YET.

**$ git status**

On branch main

Your branch is up to date with

'origin/main'.nothing to commit,

working tree clean

(e.g. README.md) that can be added to version control.

modified: README.md

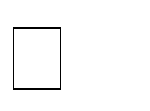
no changes added to commit (use "git add" and/or "git commit -a")

The output from the git status command above indicates that you have modified a file

# Important Git Commands

|  |  |
| --- | --- |
| 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 control tracking. To add a single file, run the command: git add file-name.extension

file or groups of files to version

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

# Creating branches

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

1. Clone the repository.
2. Move into the cloned repository
3. Create a new branch using the command (replace *feature-branch* with your desired branch name).

*git checkout -b feature-branch*

1. Make modifications to files in your project.
2. Use git add to add the changes to the staging area
3. Commit the changes with a meaningful message

# Merging branches

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

1. Switch back to the main branch.

*git checkout main*

1. Merge the branch into the main branch

*git merge feature-branch*

1. Resolve conflicts (if necessary)
   1. Open the conflicting files and resolve the conflicts manually.
   2. After resolving conflicts, add the changes to the staging area and commit:

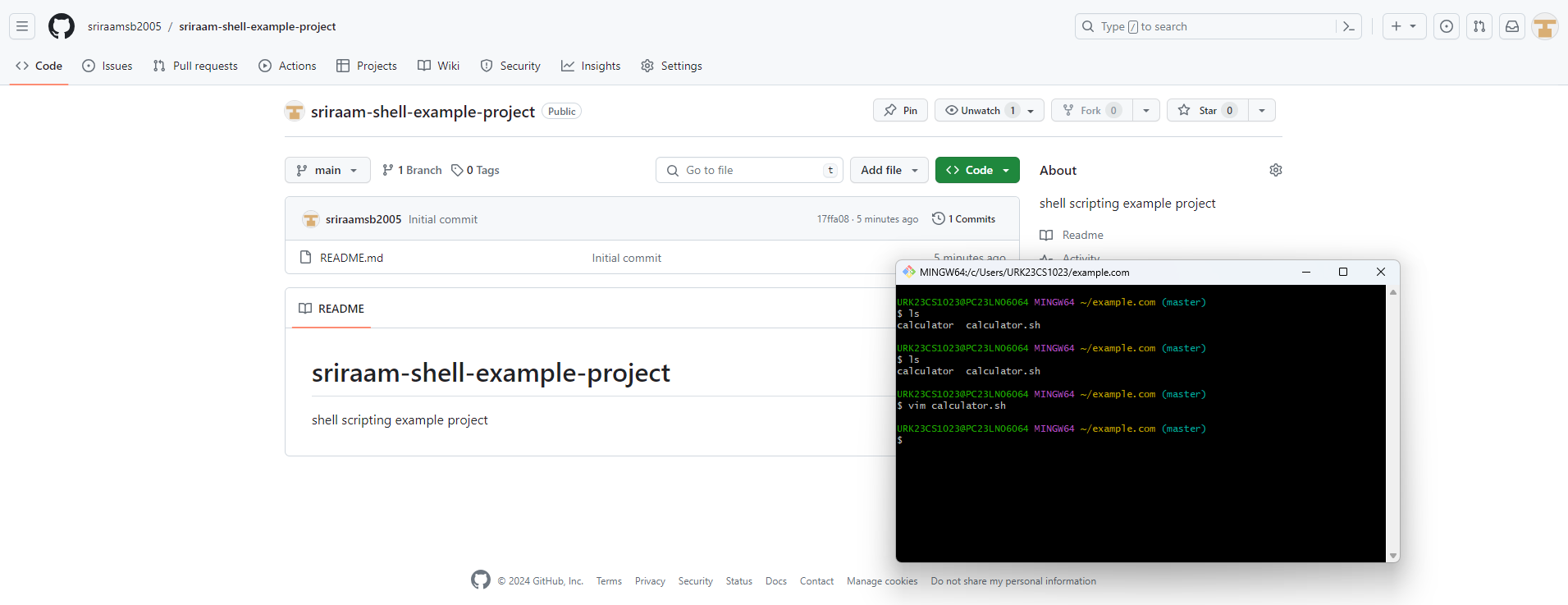
*git add .*

*git commit -m "Merge feature-branch into main"*

1. Push changes to github using the following command.

*git push origin main*

**Output:**

****

**RESULT:**

**To set up version control in software application has been implemented successfully.**

|  |  |  |
| --- | --- | --- |
| **Ex. No: 4** | **GITHUB ISSUES** | **REG NO:**  **URK23CS1023** |
| **DATE:6/2/2024** |

**Aim:**

Address and resolve identified software bugs efficiently to improve application performance and user experience.

**Theory:**

GitHub issues serve as a vital component of collaborative software development, providing a structured platform for tracking bugs, feature requests, and other tasks within a project. These issues act as a centralized repository for identifying, discussing, and prioritizing work items, enabling developers to effectively manage project workflows and address potential issues. By creating detailed descriptions of problems or desired enhancements, developers can foster clear communication and collaboration among team members. Moreover, GitHub issues facilitate transparency and accountability by documenting the status of tasks, assigning responsibilities, and tracking progress through labels, milestones, and assignees. This centralized approach streamlines the development process, empowering teams to prioritize tasks, allocate resources efficiently, and ultimately deliver high-quality software products that meet user expectations. Additionally, GitHub issues encourage community involvement, allowing users to report bugs, suggest improvements, and contribute to open-source projects, thereby fostering a vibrant ecosystem of collaboration and innovation.

# Github Issues:

GitHub Issues is a robust project management feature integrated into the GitHub platform, designed to streamline collaboration, communication, and issue tracking within software development projects.

It serves as a centralized hub for tracking tasks, enhancements, bug reports, and other project-related activities within a GitHub repository. It provides a structured and organized way for development teams to manage their workflow, ensuring that issues are identified, discussed, and resolved efficiently.

# Steps to Create and Address GitHub Issues

1. Create a new issue in your repository:
   1. Go to your GitHub repository.
   2. Click on the "Issues" tab.
   3. Click the "New issue" button.
   4. Fill in the issue title and description.
   5. Optionally, assign the issue to a collaborator, apply labels, and set milestones.
2. Assign the issue to a collaborator:
   1. Within the issue, use the "Assignees" section to assign the issue to a collaborator.
   2. The assigned collaborator will receive notifications about the issue.
3. Label the issue with appropriate tags:
   1. Apply labels to categorize and prioritize issues.
   2. Create and use labels like "bug," "feature," or any relevant labels for your project.
   3. Labels can be added when creating the issue or edited later.
4. Close the issue using a commit message:
   1. Make changes to the codebase to address the issue.
   2. In your local repository, commit the changes with a commit message that references the issue number:

*git commit -m "Fix #1: Resolve issue with feature X"*

Replace "1" with the actual issue number.

* 1. Push the changes to GitHub:

*git push origin main*

The issue will be automatically closed when the commit is pushed.

1. Optional: Comment and Discuss:
   1. Encourage collaboration by adding comments to the issue.
   2. Mention collaborators using @username to notify them.
   3. Discuss the issue, provide additional information, or ask for feedback.

**Result:**

The github code has been successfully.

|  |  |  |
| --- | --- | --- |
| **Ex no: 5** | **RESPOSITORY MANAGEMENT USING**  **GIT** | **REG NO:**  **URK23CS1023** |
| **DATE:20/2/2024** |

**Aim:**

Efficiently manage collaborative repository workflows with Git, fostering seamless cooperation and version control in team-based software development projects.

**Theory:**

Working with collaborative repository management entails the coordinated effort of multiple developers in managing source code, documents, and project assets within a shared repository. This approach leverages version control systems like Git to facilitate seamless collaboration, enabling developers to work concurrently on the same codebase while maintaining code integrity and project consistency. Collaborative repository management encompasses various practices such as branching, merging, and code review, which are essential for ensuring the smooth progression of software development projects. By branching off from the main codebase, developers can work on specific features or fixes independently, reducing conflicts and enabling parallel development efforts. Through frequent communication and code reviews, team members can provide feedback, identify issues, and ensure that changes meet project standards and requirements before being merged into the main branch. Additionally, collaborative repository management promotes transparency and accountability by tracking changes, documenting discussions, and assigning tasks, allowing teams to monitor progress, identify bottlenecks, and allocate resources effectively. Ultimately, by embracing collaborative repository management practices, development teams can streamline workflows, enhance productivity, and deliver high-quality software products in a collaborative and efficient manner.

Top of Form

Working with collaborative repository management using Git involves collaborating with other developers and effectively managing changes to ashared 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 theforked 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 theupstream repository, you can add an "upstream" remote.
* Use the `git remote add` command to add the upstream remote URL. Forexample: 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 mainbranch: **`git merge upstream/main`**
* Push the updated main branch to your forked repository: **`git push originmain`**

# 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**: `gitcheckout -b**

# <branch-name>`

**Pushing Changes and Creating Pull Requests**

* Once you've made changes on a branch, you can push the branch to yourforked 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 clickthe "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 pullrequest" 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 theupstream repository.

**Result:**

The github code has been successfully.

|  |  |  |
| --- | --- | --- |
| **EX. NO: 6** | **IMPLEMENTATION OF**  **REVIEW IN GIT** | **REG NO**  **URK23CS1024** |
| **DATE:12/3/2024** |

**Aim:**

Implement effective code review processes in Git to enhance code quality and collaboration, ensuring adherence to coding standards and fostering continuous improvement in software development projects.

**Theory:**

The implementation of code review processes in Git is a fundamental practice in software development aimed at ensuring code quality, fostering collaboration, and promoting continuous improvement within development teams. Code review involves systematically examining source code changes by peers or team members to identify potential issues, improve code readability, and ensure adherence to coding standards and best practices. By leveraging Git's version control capabilities, developers can create and review pull requests, facilitating structured discussions and feedback exchanges around proposed code changes. This collaborative approach not only helps catch bugs and logic errors early in the development cycle but also encourages knowledge sharing and mentoring among team members. Moreover, code reviews in Git promote accountability and transparency by documenting discussions, decisions, and suggested improvements, thereby facilitating learning opportunities and driving overall team growth. By embracing code review as a core practice in Git workflows, development teams can enhance code quality, minimize technical debt, and ultimately deliver more robust and reliable software solutions to end-users.

# Introduction

Git is a powerful tool for version control, but it’s also a crucial tool for collaborating with remote teams. When working remotely, effective communication and collaboration are key to ensure that your team is working together towards a common goal.

# Use a shared repository

Using a shared repository is the foundation of effective collaboration withGit. By hosting your repository on a remote server, you can give your team members 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 allow you to work on multiple versions of your code simultaneously, without interfering with each other’s work. By using branching and merging effectively, your team can work 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 from differentteam members before they are merged into the main codebase. By using pull 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, and other issues that need to be addressed in your code. By using an issue tracking system like GitHub Issues, you can assign tasks to team members, track progress, 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 in a remote

team setting. Use tools like Slack or Microsoft Teams to stay in touch with yourteam members, and use video calls or screen sharing to discuss code changes or work 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 in the Gitworkflow. You can use Git hooks to automate repetitive tasks, enforce coding standards, 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 inside other repositories. You can use Git submodules to manage dependencies, or to includeshared 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. You can useGit aliases to save time and improve your productivity when working with Git.

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

Use a Git alias: **git [alias name]**

**Result:**

The github code has been successfully.