**SEPM**

**Experiment 3**

**Aim:**

To Perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet

**Theory:**

Git commands are used like utilities to interact with our version control system

After installing git, one can get the list of all available commands by typing '*git --help*' in the CLI.

Following is the list of frequently used commands:

1. ***git add***

Moves changes from the working directory to the staging area.

This gives you the opportunity to prepare a snapshot before committing it to the official history.

1. ***git branch***

This command is your general-purpose branch administration tool.

It lets you create isolated development environments within a single repository.

1. ***git checkout***

In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches.

Combined with the basic Git commands, it’s a way to work on a particular line of development.

1. ***git clean***

Removes untracked files from the working directory.

This is the logical counterpart to git reset, which (typically) only operates on tracked files.

1. ***git clone***

Creates a copy of an existing Git repository.

Cloning is the most common way for developers to obtain a working copy of a central repository.

1. ***git commit***

Takes the staged snapshot and commits it to the project history.

Combined with git add, this defines the basic workflow for all Git users.

1. ***git commit --amend***

Passing the --amend flag to git commit lets you amend the most recent commit.

This is very useful when you forget to stage a file or omit important information from the commit message.

1. ***git config***

A convenient way to set configuration options for your Git installation.

You’ll typically only need to use this immediately after installing Git on a new development machine.

1. ***git fetch***

Fetching downloads a branch from another repository, along with all of its associated commits and files. But, it doesn't try to integrate anything into your local repository.

This gives you a chance to inspect changes before merging them with your project.

1. ***git init***

Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

1. ***git log***

Lets you explore the previous revisions of a project.

It provides several formatting options for displaying committed snapshots.

1. ***git merge***

A powerful way to integrate changes from divergent branches.

After forking the project history with the git branch, git merge lets you put it back together again.

1. ***git pull***

Pulling is the automated version of git fetch.

It downloads a branch from a remote repository, then immediately merges it into the current branch.

This is the Git equivalent of svn update.

1. ***git push***

Pushing is the opposite of fetching (with a few caveats).

It lets you move a local branch to another repository, which serves as a convenient way to publish contributions.

This is like svn commit, but it sends a series of commits instead of a single changeset.

1. ***git rebase***

Rebasing lets you move branches around, which helps you avoid unnecessary merge commits.

The resulting linear history is often much easier to understand and explore.

1. ***git rebase -i***

The -i flag is used to begin an interactive rebasing session.

This provides all the benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way.

1. ***git reflog***

Git keeps track of updates to the tip of branches using a mechanism called reflog.

This allows you to go back to changesets even though they are not referenced by any branch or tag.

1. ***git remote***

A convenient tool for administering remote connections.

Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.

1. ***git reset***

Undoes changes to files in the working directory.

Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.

1. ***git revert***

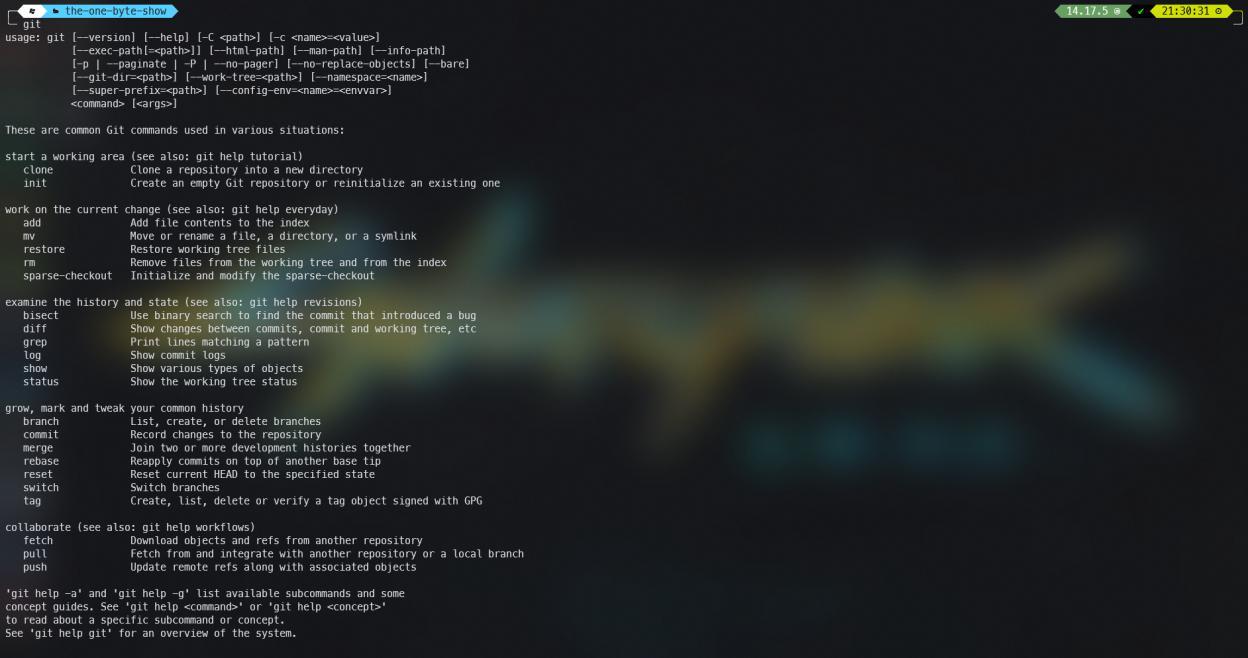
Undoes a committed snapshot.

When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the codebase.

1. ***git status***

Displays the state of the working directory and the staged snapshot.

You’ll want to run this in conjunction with git add and git commit to seeing exactly what’s being included in the next snapshot.



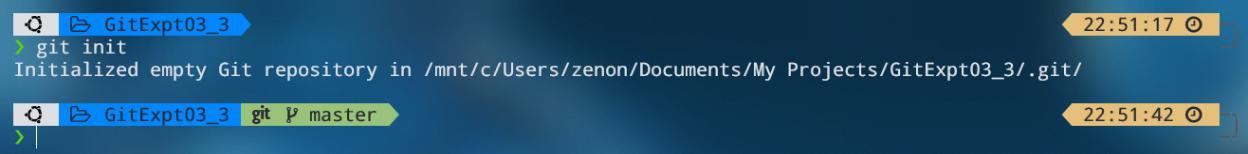
**Implementation:**

1. Setup user name
   1. Open git bash cli
   2. Use the following command to configure your username
      1. *git config --global user.name "Your UserName"*



1. Setup user email
   1. Use the following command to configure your email address
      1. *git config --global user.email "Your EmailID"*
2. Create your local repository named GitExpt03\_grno

To initialize a repository use the following command *i. 'git init’*



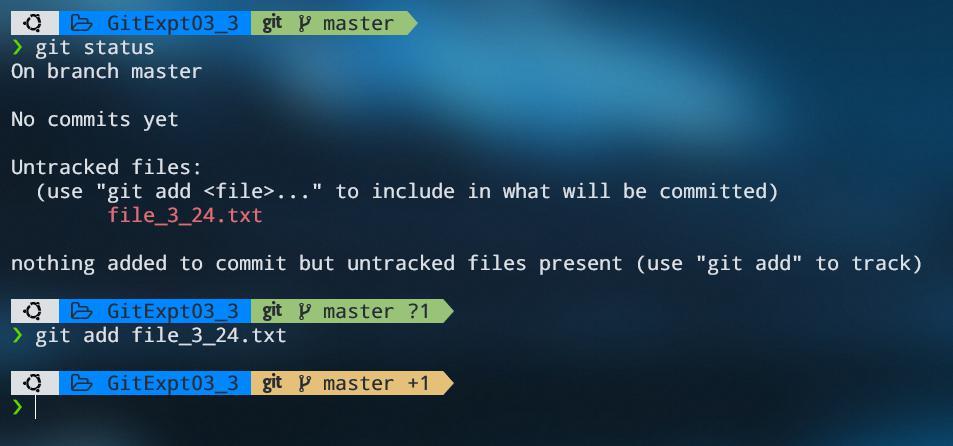
To rename repository, cd into the .git directory and put in the required repository name in the 'description' file



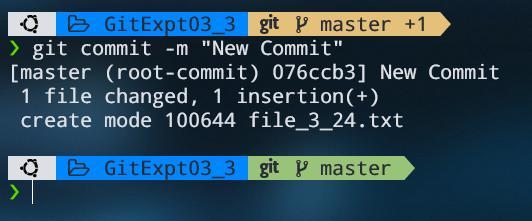
1. Add a file in above local repository named file\_grno\_rollno and add some content to the file Add it to the staging state
   1. Create a file normally with a text editor, and add contents to the file

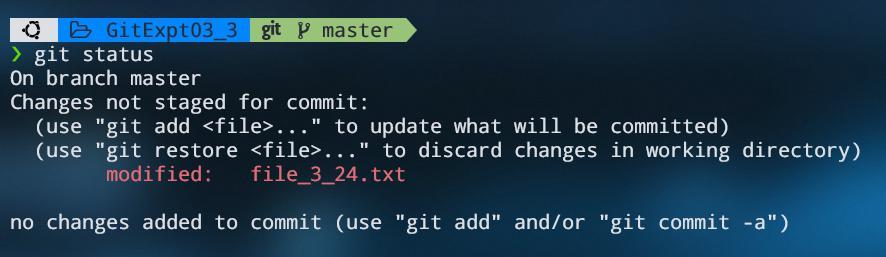


* 1. Use the following command to check if some change is made
     1. *'git status'*
  2. Use the following command to add the changes
     1. *'git add <filename>'*



5. Commit the files

1. Use the following command to commit the changes *i. 'git commit -m “<your message>”'*
2. Create a scenario for modifications/additions of the multiple files. Show the advantage of staging state.
   1. If we make some changes at this point in the process, git will show them as an untracked file.
   2. A demonstration is shown below



1. Display the log
   1. Git log is a utility tool to review and read a history of everything that happens to a repository
   2. Use the following command to view logs *i. 'git log'*

**Conclusion:**

In this experiment, we learned how to use Git Commands using the Git CLI and we learned how to create repositories, set up teams and collaborate with people on GitHub.