

## Experiment 3: To Perform various Git operations on local and remote repositories using Git cheat sheet.

### THEORY:

#### Introduction to Git

Git is a distributed version control system used for tracking changes in source code. It allows multiple developers to work on a project simultaneously while keeping track of changes and enabling collaboration through remote repositories like GitHub, GitLab, and Bitbucket.

#### Configuring Git

Before using Git for the first time, it is necessary to configure the user's identity. The following commands set up the user's name and email, which will be associated with all commits:

```
git config --global user.name "Your Name" git config --  
global user.email "your.email@example.com"
```

The `--global` flag ensures that the configuration applies to all repositories on the system.

#### Initializing a Git Repository

A Git repository must be initialized before tracking changes. This is done using the `git init` command:

```
git init
```

Executing this command creates a hidden `.git` directory within the project folder, which stores all version control information.

## Checking the Status of a Repository

To check the current state of the repository, including untracked and modified files, use:

```
git status
```

This command provides an overview of changes that need to be staged, committed, or pushed.

## Adding Files to the Staging Area

Before committing changes, files must be added to the staging area. This can be done using:

```
git add <file_name> # Adds a specific file
```

```
git add . # Adds all modified and new files
```

The staging area acts as an intermediate step before committing changes.

## Committing Changes

A commit captures the current state of the repository and saves it locally. Each commit requires a message that describes the changes made:

```
git commit -m "Descriptive commit message"
```

Commits are local and do not affect the remote repository until they are pushed.

## Connecting to a Remote Repository

To link the local repository with a remote repository (e.g., GitHub), use:

```
git remote add origin <repository_URL>
```

For example:

```
git remote add origin https://github.com/username/repository.git
```

To verify that the remote repository has been added, use: git

```
remote -v
```

## **Pushing Changes to a Remote Repository**

To upload commits to a remote repository: git

```
push origin main
```

- `origin` refers to the remote repository.
- `main` refers to the branch being pushed.

For the first push, use: git

```
push -u origin main
```

The `-u` flag sets `origin main` as the default upstream branch, allowing future pushes to be done with `git push` alone.

## **Pulling Changes from a Remote Repository**

To retrieve and merge updates from the remote repository:

```
git pull origin main
```

This command ensures the local repository is up-to-date with the remote repository.

## **Cloning an Existing Repository**

To create a local copy of an existing remote repository:

```
git clone <repository_URL>
```

For example:

```
git clone https://github.com/username/repository.git
```

This command downloads the repository and sets up a connection to the remote repository.

## **Branching and Merging**

Git allows working with multiple branches to develop new features without affecting the main codebase. Creating a new branch: `git branch new-branch`

Switching to the new branch:

```
git checkout new-branch
```

Merging a branch into the main branch:

```
git merge new-branch
```

Deleting a branch:

```
git branch -d new-branch
```

Branches help in parallel development and version control management.

## Implementation:

[illegible]

```
MINGW64 ~/Users/soham/Downloads/top/gt-deploy-demo-project
~/Downloads (1) MINGW64 ~/Downloads/top/gt-deploy-demo-project
$ cd git-deploy

~/Downloads (1) MINGW64 ~/Downloads/top/gt-deploy-demo-project
$ git config --global
usage: git config [--options]

Config file location
  --global      use global config file
  --system      use system config file
  --local       use repository config file
  --per-user    use per-user config file
  -f, --file <file> use given config file
  --[no]blob <blob-id> read config from given blob object

action
  --get          get value: name [value-pattern]
  --get-all     get all values: key [value-pattern]
  --get-regexp   get values for regexp: name-regexp [value-pattern]
  --get-or-show  get value specific for the old section[.var] old
  --replace-all replace all matching variables: name value [value-pattern]

add
  --add          add a new variable: name value
  --unset        remove a variable: name [value-pattern]
  --unset-all   remove all matches: name [value-pattern]
  --rename-section rename section: old-name new-name
  --remove-section remove a section: name
  -l, --[no]list list all
  --[no]fixup-value use string equality when comparing values to 'value-as

type
  -e, --[no]edit open an editor
  --get-color     find the color configured: slot [default]
  --get-colorbool find the color setting: slot [show-is-ty]

Type
  -t, --[no]type <type> value is given this type
  --bool          value is "true" or "false"
  --int           value is decimal number
  --bool-or-int   value is --bool or --int
  --bool-or-str   value is --bool or string
  --path          value is a path (file or directory name)
  --expiry-date   value is an expiry date

Other
  -s, --[no]null terminate values with NUL byte
  --name-only     show variable names only
  --includes      respect include directives on lookup
  --show-origin   show origin of config (file, standard input, blob, env

read (flag)
  --[no]show-scope show scope of config (worktree, local, global, system,
  --[no]default-value with --get, use default value when missing entry

~/Downloads (1) MINGW64 ~/Downloads/top/gt-deploy-demo-project
$ git config --global user.name "Mharsa@soham"

MINGW64 ~/Users/soham/Downloads/top/gt-deploy-demo-project
~/Downloads (1) MINGW64 ~/Downloads/top/gt-deploy-demo-project
$ cd git-deploy

~/Downloads (1) MINGW64 ~/Downloads/top/gt-deploy-demo-project
$ git config --global
usage: git config [--options]

Config file location
  --global      use global config file
  --system      use system config file
  --local       use repository config file
  --per-user    use per-user config file
  -f, --file <file> use given config file
  --[no]blob <blob-id> read config from given blob object

action
  --get          get value: name [value-pattern]
  --get-all     get all values: key [value-pattern]
  --get-regexp   get values for regexp: name-regexp [value-pattern]
  --get-or-show  get value specific for the old section[.var] old
  --replace-all replace all matching variables: name value [value-pattern]

add
  --add          add a new variable: name value
  --unset        remove a variable: name [value-pattern]
  --unset-all   remove all matches: name [value-pattern]
  --rename-section rename section: old-name new-name
  --remove-section remove a section: name
  -l, --[no]list list all
  --[no]fixup-value use string equality when comparing values to 'value-as

type
  -e, --[no]edit open an editor
  --get-color     find the color configured: slot [default]
  --get-colorbool find the color setting: slot [show-is-ty]

Type
  -t, --[no]type <type> value is given this type
  --bool          value is "true" or "false"
  --int           value is decimal number
  --bool-or-int   value is --bool or --int
  --bool-or-str   value is --bool or string
  --path          value is a path (file or directory name)
  --expiry-date   value is an expiry date

Other
  -s, --[no]null terminate values with NUL byte
  --name-only     show variable names only
  --includes      respect include directives on lookup
  --show-origin   show origin of config (file, standard input, blob, env

read (flag)
  --[no]show-scope show scope of config (worktree, local, global, system,
  --[no]default-value with --get, use default value when missing entry
```

```
MINGW64 ~/Downloads/esp/gh-demo-project
1 cd git-demo-project
2 git push -u origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

3 git push -u origin main
error: src refspec main does not match any
fatal: Path 'main' does not exist in 'origin'

4 git status
On branch master
nothing to commit, working tree clean

5 git push -u origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

6 git remote add origin https://github.com/AttharvAttharv/ESP-Lab.git
git branch -M main
git push -u origin main

remote: Permission to AttharvAttharv/ESP-Lab.git denied to jettik00.
fatal: unable to access 'https://github.com/AttharvAttharv/ESP-Lab.git/': The requested URL returned error: 401

7 git remote add origin https://github.com/AttharvAttharv/ESP-Lab.git
error: remote origin already exists.

8 git push -u origin master
error: src refspec master does not match any
fatal: Path 'master' does not exist in 'https://github.com/AttharvAttharv/ESP-Lab.git'

9 git push -u origin main
Counting objects: 4, done.
Compressing objects: 100% (4/4), 287 bytes | 187.00 KiB/s, done.
Writing objects: 100% (4/4), 287 bytes | 187.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pushed 4 (from 0)
To https://github.com/AttharvAttharv/ESP-Lab.git
 * [new branch] main -> main
branch 'main' set up to track 'origin/main'.

10 git push -u origin main
```

## Conclusion:

Successfully implemented various Git operations on local and remote repositories using the Git cheat sheet.