Here are some commonly used Git commands with examples to illustrate their usage:

### Basic Commands

**1. \*\*`git init`\*\***

Initializes a new Git repository in the current directory.

```bash

**git init**

```

\*Example:\* Creating a new Git repository in your current project folder.

2**. \*\*`git clone <repository-url>`\*\***

Clones an existing Git repository from a URL to your local machine.

```bash

**git clone https://github.com/user/repo.git**

```

\*Example:\* Cloning a repository from GitHub to your local machine.

3. \*\*`git status`\*\*

Displays the status of the working directory and staging area.

```bash

git status

```

\*Example:\* Checking the status of your repository to see which files have changes that are not yet committed.

4. \*\*`git add <file>`\*\*

Adds a specific file to the staging area.

```bash

git add file.txt

```

\*Example:\* Staging the file `file.txt` for the next commit.

To stage all changes, you can use:

```bash

git add .

```

\*Example:\* Staging all modified and new files in the current directory.

5. \*\*`git commit -m "commit message"`\*\*

Commits the staged changes to the repository with a descriptive message.

```bash

git commit -m "Add new feature to the application"

```

\*Example:\* Committing staged changes with a message describing what was done.

6. \*\*`git push`\*\*

Pushes the committed changes from the local repository to a remote repository.

```bash

git push origin main

```

\*Example:\* Pushing changes to the `main` branch of the remote repository named `origin`.

7. \*\*`git pull`\*\*

Fetches and merges changes from the remote repository to your local repository.

```bash

git pull origin main

```

\*Example:\* Pulling the latest changes from the `main` branch of the remote repository named `origin`.

### Branching and Merging

1. \*\*`git branch`\*\*

Lists all branches in the repository or creates a new branch.

```bash

git branch

```

\*Example:\* Listing all branches in the repository.

```bash

git branch feature-branch

```

\*Example:\* Creating a new branch named `feature-branch`.

2. \*\*`git checkout <branch-name>`\*\*

Switches to the specified branch.

```bash

git checkout feature-branch

```

\*Example:\* Switching to the `feature-branch` branch.

To create and switch to a new branch:

```bash

git checkout -b new-feature

```

\*Example:\* Creating a new branch named `new-feature` and switching to it.

3. \*\*`git merge <branch-name>`\*\*

Merges the specified branch into the current branch.

```bash

git merge feature-branch

```

\*Example:\* Merging the changes from `feature-branch` into the current branch.

4. \*\*`git rebase <branch-name>`\*\*

Reapplies commits from the current branch on top of the specified branch.

```bash

git rebase main

```

\*Example:\* Reapplying the commits of the current branch on top of the `main` branch.

### Advanced Commands

1. \*\*`git reset <commit>`\*\*

Resets the current branch to a specific commit.

```bash

git reset --hard a1b2c3d

```

\*Example:\* Resetting the current branch to the commit with hash `a1b2c3d` and discarding all changes after it.

2. \*\*`git revert <commit>`\*\*

Creates a new commit that undoes the changes of a specified commit.

```bash

git revert a1b2c3d

```

\*Example:\* Reverting the changes made by the commit with hash `a1b2c3d`.

3. \*\*`git log`\*\*

Shows the commit history for the repository.

```bash

git log

```

\*Example:\* Viewing the commit history of the repository.

For a more concise output:

```bash

git log --oneline

```

\*Example:\* Viewing the commit history in a single line per commit format.

4. \*\*`git stash`\*\*

Temporarily saves changes in the working directory that are not ready to be committed.

```bash

git stash

```

\*Example:\* Stashing uncommitted changes to work on something else.

5. \*\*`git stash pop`\*\*

Applies the most recently stashed changes and removes them from the stash list.

```bash

git stash pop

```

\*Example:\* Applying the most recent stashed changes back to your working directory.

6. \*\*`git remote`\*\*

Manages remote repository connections.

```bash

git remote -v

```

\*Example:\* Listing all configured remotes and their URLs.

7. \*\*`git fetch`\*\*

Downloads objects and refs from another repository but does not merge them into your working directory.

```bash

git fetch origin

```

\*Example:\* Fetching the latest changes from the remote repository `origin`.

8. \*\*`git diff`\*\*

Shows the differences between the working directory and the staging area, or between commits.

```bash

git diff

```

\*Example:\* Viewing changes that have not yet been staged.

To see changes between commits:

```bash

git diff a1b2c3d..d4e5f6g

```

\*Example:\* Viewing differences between two commits identified by their hashes.

### Git Config Commands

1. \*\*`git config --global user.name "Your Name"`\*\*

Sets the username for Git commits.

```bash

git config --global user.name "John Doe"

```

\*Example:\* Setting your name in Git configuration for all repositories.

2. \*\*`git config --global user.email "you@example.com"`\*\*

Sets the email address for Git commits.

```bash

git config --global user.email "john@example.com"

```

\*Example:\* Setting your email in Git configuration for all repositories.

3. \*\*`git config --list`\*\*

Lists all Git configurations for the current repository or user.

```bash

git config --list

```

\*Example:\* Viewing all current Git configuration settings.