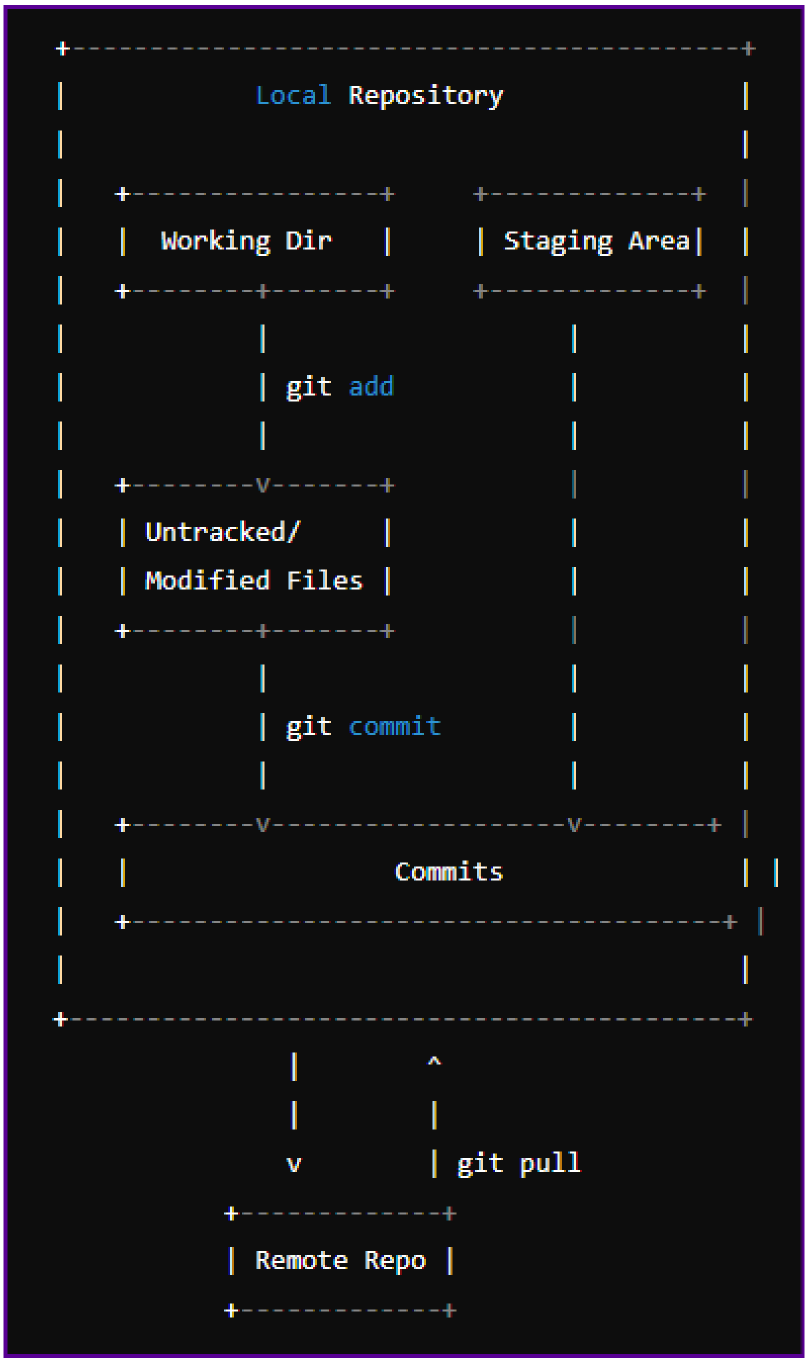
DevOps Shack

Git Comprehensive Guide

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]



```
+-------------------------------------------+
|             Local Repository              |
|                                           |
|   +----------------+     +-------------+  |
|   |  Working Dir   |     | Staging Area|  |
|   +-------+--------+     +-------------+  |
|           |                   |           |
|           | git add           |           |
|           |                   |           |
|   +-------v--------+          |           |
|   | Untracked/     |          |           |
|   | Modified Files |          |           |
|   +-------+--------+          |           |
|           |                   |           |
|           | git commit        |           |
|           |                   |           |
|   +-------v-------------------v--------+  |
|   |                Commits             | |
|   +------------------------------------+ |
|                                           |
+-------------------------------------------+
              |        ^
              |        |
              v        | git pull
           +-------------+
           | Remote Repo |
           +-------------+
```

Introduction to Git

Git is a distributed version control system designed to handle projects with speed and efficiency. It allows multiple developers to work on a project simultaneously without overwriting each other's changes.

Key Concepts in Git

1. **Repository (Repo)**: A storage space where your project resides. It can be local or remote.
2. **Working Directory**: The local files and directories that are part of your Git project.
3. **Staging Area (Index)**: A place where you can group changes to prepare them for a commit.
4. **Commit**: A snapshot of the changes in the repository.
5. **Branch**: A separate line of development.
6. **Master/Main**: The default primary branch in Git.
7. **HEAD**: A pointer to the current branch's latest commit.
8. **Clone**: A copy of a repository.
9. **Fork**: A personal copy of someone else's repository.
10. **Pull Request (PR)**: A method of submitting contributions to a project.
11. **Merge**: Combining changes from different branches.
12. **Rebase**: Reapplying commits on top of another base tip.
13. **Conflict**: Occurs when changes from different commits interfere with each other.
14. **Remote**: A version of your project that is hosted on the internet or network.
15. **Fetch**: Downloads commits, files, and refs from a remote repository.
16. **Pull**: Fetches and merges changes from a remote repository to your working directory.
17. **Push**: Uploads local changes to a remote repository.
18. **Tag**: A reference to a specific point in the Git history.

Git Commands

1. Configuration

`git config`
Used to set Git configuration values on a global or local project level.

- Set username:

  ```
  git config --global user.name "Your Name"
  ```
- Set email:

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

- Check configuration:

```
git config --list
```

2. Initializing a Repository

`git init`
Initializes a new Git repository.

- Initialize a repository:

```
git init
```

3. Cloning a Repository

`git clone`
Clones an existing repository.

- Clone a repository:

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

4. Working with the Working Directory

`git status`
Shows the working directory status.

- Check status:

```
git status
```

`git add`
Adds files to the staging area.

- Add a specific file:

```
git add filename
```

- Add all files:

```
git add .
```

`git commit`
Records changes to the repository.

- Commit with a message:

```
git commit -m "Commit message"
```

- Commit with all changes:

```
git commit -a -m "Commit message"
```

`git diff`
Shows changes between commits, commit and working directory, etc.

- Show changes in the working directory:

  ```
  git diff
  ```

5. Branching and Merging

`git branch`
Lists, creates, or deletes branches.

- List branches:

  ```
  git branch
  ```
- Create a new branch:

  ```
  git branch new-branch
  ```
- Delete a branch:

  ```
  git branch -d branch-name
  ```

`git checkout`

Switches branches or restores working directory files.

- Switch to a branch:

  ```
  git checkout branch-name
  ```
- Create and switch to a new branch:

  ```
  git checkout -b new-branch
  ```

`git merge`
Joins two or more development histories together.

- Merge a branch into the current branch:

  ```
  git merge branch-name
  ```

`git rebase`
Reapplies commits on top of another base tip.

- Rebase the current branch onto another branch:

  ```
  git rebase branch-name
  ```

6. Remote Repositories

`git remote`
Manages set of tracked repositories.

- Add a remote repository:

  ```
  git remote add origin https://github.com/user/repo.git
  ```
- List remote repositories:

  ```
  git remote -v
  ```
- Remove a remote repository:

  ```
  git remote remove origin
  ```

`git fetch`
Downloads objects and refs from another repository.

- Fetch changes from the remote repository:

  ```
  git fetch origin
  ```

`git pull`
Fetches from and integrates with another repository or a local branch.

- Pull changes from a remote repository:

  ```
  git pull origin main
  ```

`git push`
Updates remote refs along with associated objects.

- Push changes to a remote repository:

  ```
  git push origin main
  ```

7. Stashing and Cleaning

`git stash`
Temporarily shelves changes you've made to your working directory.

- Stash changes:

  ```
  git stash
  ```
- Apply stashed changes:

  ```
  git stash apply
  ```

`git clean`
Removes untracked files from the working directory.

- Clean untracked files:

  ```
  git clean -f
  ```

8. Viewing History

`git log`
Shows the commit logs.

- View commit history:

  ```
  git log
  ```
- View a specific number of commits:

  ```
  git log -n 10
  ```

`git show`
Displays information about objects.

- Show details of a commit:

  ```
  git show commit-id
  ```

`git blame`
Shows what revision and author last modified each line of a file.

- Blame a file:

  ```
  git blame filename
  ```

9. Undoing Changes

`git reset`
Resets current HEAD to the specified state.

- Unstage a file:

  ```
  git reset filename
  ```
- Reset to a specific commit:

  ```
  git reset --hard commit-id
  ```

`git revert`
Reverts a commit by creating a new commit.

- Revert a commit:

```
git revert commit-id
```

10. Tagging

`git tag`
Creates, lists, or deletes tags.

- Create a tag:

  ```
  git tag v1.0
  ```
- List tags:

  ```
  git tag
  ```
- Delete a tag:

  ```
  git tag -d v1.0
  ```

11. Collaborating with Others

`git cherry-pick`
Apply the changes introduced by some existing commits.

- Cherry-pick a commit:

  ```
  git cherry-pick commit-id
  ```

`git submodule`
Initialize, update, or inspect submodules.

- Add a submodule:

  ```
  git submodule add https://github.com/user/repo.git
  ```
- Update submodules:

  ```
  git submodule update --remote
  ```

12. Advanced Commands

`git bisect`
Use binary search to find the commit that introduced a bug.

- Start a bisect session:

  ```
  git bisect start
  ```
- Mark current commit as bad:

  ```
  git bisect bad
  ```
- Mark a known good commit:

```
git bisect good commit-id
```

`git reflog`
Record when branches and tips were updated.

- View the reference log:

  ```
  git reflog
  ```

13. Configuration and Optimization

`git gc`
Cleanup unnecessary files and optimize the local repository.

- Run garbage collection:

  ```
  git gc
  ```

`git fsck`
Verifies the connectivity and validity of the objects in the database.

- Check the repository:

  ```
  git fsck
  ```

Use Cases and Workflows

1. **Basic Workflow**

   - Create a repository:

     ```
     git init
     ```
   - Add files:

     ```
     git add .
     ```
   - Commit changes:

     ```
     git commit -m "Initial commit"
     ```

2. **Collaborative Workflow**

   - Clone a repository:

     ```
     git clone https://github.com/user/repo.git
     ```
   - Create a branch:

     ```
     git checkout -b feature-branch
     ```
   - Make changes and commit:

   -
     ```
     git add .
     git commit -m "Added new feature"
     ```

- Push changes:

  ```
  git push origin feature-branch
  ```
- Create a pull request on GitHub.

3. **Rebasing Workflow**

   - Rebase feature branch onto main:

   -
     ```
     git checkout feature-branch
     git rebase main
     ```
   - Resolve any conflicts and continue:
   - `git add .`
   - `git rebase --continue`
   -

4. **Handling Merge Conflicts**

   - Merge branch:

     ```
     git merge feature-branch
     ```
   - Resolve conflicts in files.
   - Mark conflicts as resolved:

   -
     ```
     git add .
     git commit -m "Resolved merge conflicts"
     ```

5. **Using Stashes**

   - Save changes temporarily:

     ```
     git stash
     ```
   - Apply stashed changes:

     ```
     git stash apply
     ```

6. **Tagging Releases**

   - Create a tag for a release:

     ```
     git tag v1.0
     ```
   - Push tags to remote:

     ```
     git push --tags
     ```

Git Best Practices

1. **Commit Often**: Frequent commits with meaningful messages.
2. **Use Branches**: Keep main branch clean, use feature branches.
3. **Pull Before Push**: Always pull latest changes before pushing.
4. **Write Good Commit Messages**: Clearly describe what the commit does.

5. **Use .gitignore**: Keep unnecessary files out of the repository.
6. **Review Before Commit**: Check your changes before committing.

Advanced Git Commands and Concepts

14. Managing Remote Repositories

`git remote show`
Displays detailed information about a remote repository.

- Show remote details:

```
git remote show origin
```

`git remote rename`
Renames a remote.

- Rename a remote:

```
git remote rename origin new-origin
```

`git remote set-url`
Changes the URL of a remote repository.

- Set a new URL:

```
git remote set-url origin https://new-url.com/repo.git
```

15. Working with Patches

`git format-patch`
Creates a patch file for a commit.

- Create a patch for the last commit:

```
git format-patch -1
```

`git apply`
Applies a patch to the working directory.

- Apply a patch:

```
git apply patch-file.patch
```

16. Rebasing and Interactive Rebasing

`git rebase -i`
Allows you to interactively rebase your commits.

- Start an interactive rebase:

```
git rebase -i HEAD~3
```

- During an interactive rebase, you can:
  - `pick` a commit to include it.
  - `squash` (or `s`) to merge it into the previous commit.
  - `reword` (or `r`) to change the commit message.
  - `edit` (or `e`) to make changes before continuing.

17. Resetting Commits

`git reset`
Resets the current HEAD to the specified state.

- Soft reset (keeps changes in the working directory):

  ```
  git reset --soft HEAD~1
  ```

- Mixed reset (keeps changes but unstages them):

  ```
  git reset --mixed HEAD~1
  ```

- Hard reset (discards changes):

  ```
  git reset --hard HEAD~1
  ```

18. Checking Out Commits and Files

`git checkout`
Switch branches or restore files.

- Checkout a specific commit:

  ```
  git checkout commit-id
  ```

- Restore a specific file to its state in a specific commit:

  ```
  git checkout commit-id -- filename
  ```

19. Cherry-Picking Commits

`git cherry-pick`
Apply changes introduced by some existing commits.

- Cherry-pick a commit from another branch:

  ```
  git cherry-pick commit-id
  ```

20. Bisecting to Find Bugs

`git bisect`
Use binary search to find the commit that introduced a bug.

- Start bisecting:
  ```
  git bisect start
  ```
- Mark the current commit as bad:
  ```
  git bisect bad
  ```
- Mark a known good commit:
  ```
  git bisect good commit-id
  ```
- After finding the bad commit, end bisecting:
  ```
  git bisect reset
  ```

21. Using Submodules

`git submodule`
Manage submodules, which are external repositories within your repository.

- Add a submodule:
  ```
  git submodule add https://github.com/user/repo.git
  ```
- Initialize and update submodules:
  ```
  git submodule update --init --recursive
  ```
- Remove a submodule:

-
  ```
  git submodule deinit -f path/to/submodule
  git rm -f path/to/submodule
  ```

22. Aliases

`git config alias`
Create shortcuts for Git commands.

- Create an alias for `git status`:
  `git config --global alias.st status`
- Create an alias for a log graph:
  ```
  git config --global alias.lg "log --oneline --graph --decorate"
  ```

23. Managing Large Files

`git-lfs`
Git Large File Storage (LFS) replaces large files with text pointers inside Git while storing the file contents on a remote server.

- Install Git LFS:

  ```
  git lfs install
  ```

- Track a file type with LFS:

  ```
  git lfs track "*.psd"
  ```

- Add and commit LFS tracked files:

- `git add .gitattributes`
- `git add largefile.psd`
- `git commit -m "Add large file"`
  `git push origin main`

24. Logging and Blaming

`git log`
Show commit logs with various options.

- One line log:

  ```
  git log --oneline
  ```

- Log with graph:

  ```
  git log --graph --oneline
  ```

- Log with specific author:

  ```
  git log --author="Author Name"
  ```

`git blame`
Show what revision and author last modified each line of a file.

- Blame a file:

  ```
  git blame filename
  ```

25. Rewriting History

`git filter-branch`
Rewrite branches.

- Remove a file from history:

  ```
  git filter-branch --tree-filter 'rm -f filename' HEAD
  ```

`git rebase -i`

Interactively rebase to modify commit history.

- Start an interactive rebase:

```
git rebase -i HEAD~n
```

26. Working with Tags

`git tag`
Create, list, delete, and verify tags.

- Create an annotated tag:

```
git tag -a v1.0 -m "Version 1.0"
```

- List tags:

```
git tag
```

- Show details of a tag:

```
git show v1.0
```

- Push a tag to remote:

```
git push origin v1.0
```

- Delete a local tag:

```
git tag -d v1.0
```

- Delete a remote tag:

```
git push origin --delete tag v1.0
```

27. Searching and Finding

`git grep`
Search for patterns in the repository.

- Search for a term:

```
git grep "search-term"
```

`git fsck`
Verify the connectivity and validity of objects in the database.

- Check the repository:

```
git fsck
```

28. Git Flow

Git Flow

Git Flow is a branching model for Git, developed by Vincent Driessen. It defines a strict branching model designed around the project release.

- Initialize Git Flow:

  ```
  git flow init
  ```

- Start a feature:

  ```
  git flow feature start feature-name
  ```

- Finish a feature:

  ```
  git flow feature finish feature-name
  ```

- Start a release:

  ```
  git flow release start release-version
  ```

- Finish a release:

  ```
  git flow release finish release-version
  ```

- Start a hotfix:

  ```
  git flow hotfix start hotfix-description
  ```

- Finish a hotfix:

  ```
  git flow hotfix finish hotfix-description
  ```

29. Continuous Integration

Git Hooks

Git hooks are scripts that run automatically on specific Git events.

- Create a pre-commit hook:

-
  ```
  touch .git/hooks/pre-commit
  chmod +x .git/hooks/pre-commit
  ```
- Example pre-commit hook:

-
  ```
  #!/bin/sh
  echo "Running pre-commit hook"
  ```

30. Handling Large Repositories

`git sparse-checkout`
Allows checking out only a subset of the repository.

- Enable sparse-checkout:

  ```
  git sparse-checkout init
  ```

- Set sparse-checkout paths:

  ```
  git sparse-checkout set folder1 folder2
  ```

Additional Use Cases

1. **Rewriting Commit Messages**
   - Amend the last commit:

     ```
     git commit --amend -m "New commit message"
     ```

2. **Fixing Mistakes**
   - Undo a commit while keeping changes:

     ```
     git reset --soft HEAD~1
     ```

3. **Ignoring Files**
   - Add a `.gitignore` file to ignore specific files:

   -
     ```
     node_modules/
     *.log
     ```

4. **Managing Multiple Repositories**
   - Add a secondary remote:

     ```
     git remote add secondary https://github.com/user/another-
     repo.git
     ```

5. **Tracking Changes**
   - View changes over time:

     ```
     git log -p
     ```

6. **Maintaining a Clean History**
   - Squash commits:

     ```
     git rebase -i HEAD~n
     ```

7. **Advanced Branch Management**

    - Track remote branches:

      ```
      git checkout -t origin/branch-name
      ```