**Git Command Guide**

Welcome to the Git Command Guide. This document is intended to help you understand the basic Git commands with explanations and examples. It is a useful resource for new employees to learn how to effectively use Git in their workflow.

**Basic Commands**

**1. git clone**

**Explanation:**  
Clone a remote repository to your local machine.

**Example:**

bash

Copy code

git clone https://github.com/msnav/gitdemo.git

**2. git add**

**Explanation:**  
Stage files to prepare them for commit.

**Common arguments:**

* git add -A: Stage all files.
* git add -u: Stage all tracked files.
* git add <file>: Stage a specific file.

**Example:**

bash

Copy code

git add ReadMe.md

**3. git status**

**Explanation:**  
Display the state of the working directory and the staging area.

**Example:**

bash

Copy code

git status

**4. git reset**

**Explanation:**  
Undo changes or unstage files.

**Common arguments:**

* git reset: Unstage all files but preserve changes.
* git reset --hard: Unstage all files and delete changes (CAUTION!).

**Example:**

bash

Copy code

git reset

git reset --hard 5b331f3

**5. git restore**

**Explanation:**  
Unstage files or discard changes.

**Common arguments:**

* git restore --staged <file>: Unstage a file.
* git restore <file>: Discard changes since last commit.

**Example:**

bash

Copy code

git restore test.txt

**History and Comparison**

**6. git log**

**Explanation:**  
Show commit history with optional filters.

**Common arguments:**

* --after=<date>: Show commits after a specific date.
* --author=<name>: Show commits by a specific author.

**Example:**

bash

Copy code

git log --after="2022-01-01"

**7. git diff**

**Explanation:**  
Show changes between commits, working directory, and staging area.

**Common arguments:**

* git diff: Show unstaged changes.
* git diff --cached: Show staged changes.

**Example:**

bash

Copy code

git diff --cached test.txt

**Committing and Pushing**

**8. git commit**

**Explanation:**  
Record changes to the repository.

**Common arguments:**

* -m <message>: Add a commit message.
* --amend: Edit the previous commit message.

**Example:**

bash

Copy code

git commit -m "Updated ReadMe.md"

**9. git push**

**Explanation:**  
Upload local commits to the remote repository.

**Common arguments:**

* git push <remote>: Push to a specific remote.
* --force: Force push changes.

**Example:**

bash

Copy code

git push origin main

**Branching and Merging**

**10. git branch**

**Explanation:**  
Create, list, or delete branches.

**Common arguments:**

* git branch: List branches.
* git branch <name>: Create a new branch.
* git branch --delete <name>: Delete a branch.

**Example:**

bash

Copy code

git branch development

**11. git switch**

**Explanation:**  
Switch between branches.

**Common arguments:**

* git switch <name>: Switch to a branch.
* git switch -c <name>: Create and switch to a new branch.

**Example:**

bash

Copy code

git switch development

**12. git checkout**

**Explanation:**  
Switch branches or restore working tree files.

**Common arguments:**

* git checkout <branch>: Switch to a branch.
* git checkout <commit>: Check out a specific commit.

**Example:**

bash

Copy code

git checkout main

**13. git merge**

**Explanation:**  
Combine changes from different branches.

**Example:**

bash

Copy code

git merge devel

**14. git tag**

**Explanation:**  
Create, list, delete, or verify a tag.

**Common arguments:**

* git tag: List all tags.
* git tag <name>: Create a tag.
* git tag --delete <name>: Delete a tag.

**Example:**

bash

Copy code

git tag v1.0

**Remote Repositories**

**15. git fetch**

**Explanation:**  
Download objects and refs from another repository.

**Example:**

bash

Copy code

git fetch origin main

**16. git pull**

**Explanation:**  
Fetch from and integrate with another repository or a local branch.

**Example:**

bash

Copy code

git pull origin main

**Advanced Commands**

**17. git rebase**

**Explanation:**  
Reapply commits on top of another base tip.

**Example:**

bash

Copy code

git rebase -i HEAD~5

**18. git revert**

**Explanation:**  
Create a new commit by reverting changes from a previous commit.

**Example:**

bash

Copy code

git revert 2fc0df

**19. git stash**

**Explanation:**  
Stash changes in a dirty working directory away.

**Common arguments:**

* git stash list: List stash entries.
* git stash pop: Apply and remove the latest stash.

**Example:**

bash

Copy code

git stash

git stash pop

**20. git clean**

**Explanation:**  
Remove untracked files from the working directory.

**Common arguments:**

* git clean -n: Show files to be removed.
* git clean -x: Remove ignored files as well.

**Example:**

bash

Copy code

git clean -n

**Conflict Resolution**

**Resolving Merge Conflicts**

When two branches have conflicting changes, Git will stop and allow you to resolve the conflicts manually. Here is how you can handle merge conflicts:

1. **Identify the conflicts:**  
   Run git status to see which files have conflicts.
2. **Edit the conflicted files:**  
   Open the conflicted files and look for the conflict markers <<<<<<<, =======, and >>>>>>>. Edit the files to resolve the conflicts and remove the markers.
3. **Mark files as resolved:**  
   After resolving the conflicts, use git add <file> to mark the conflicts as resolved.
4. **Commit the merge:**  
   Once all conflicts are resolved, commit the merge using:

bash

Copy code

git commit

**Example of Resolving a Conflict**

Suppose you have a conflict in a file called example.txt. You would:

1. Open example.txt and manually resolve the conflicts.
2. Run:

bash

Copy code

git add example.txt

git commit -m "Resolved conflict in example.txt"

**Squashing Commits**

Squashing commits allows you to combine multiple commits into one, which is useful for simplifying commit history before merging.

**How to Squash Commits**

1. Start an interactive rebase for the last 5 commits:

bash

Copy code

git rebase -i HEAD~5

1. In the editor, change the word pick to squash (or s) for the commits you want to squash into the previous one.
2. Save and close the editor. Git will combine the commits and allow you to edit the commit message.
3. Save the commit message to finish the rebase.

**Example**

If you have five commits you want to squash into one:

bash

Copy code

git rebase -i HEAD~5

# Change all but the first commit from "pick" to "squash"

# Edit the commit message as needed

**Merging into Main Branch**

After reviewing and finalizing your changes on a feature branch, you will want to merge them into the main branch.

**Steps to Merge into Main**

1. **Switch to the main branch:**

bash

Copy code

git checkout main

1. **Pull the latest changes:**

bash

Copy code

git pull origin main

1. **Merge your feature branch:**

bash

Copy code

git merge feature-branch

1. **Push the updated main branch:**

bash

Copy code

git push origin main

**Example of Merging into Main**

Suppose your feature branch is called new-feature:

bash

Copy code

git checkout main

git pull origin main

git merge new-feature

git push origin main

**Updating After Review**

After receiving feedback on your pull request, you might need to make changes and update your branch.

**Steps to Update**

1. **Make the necessary changes.**
2. **Commit your changes:**

bash

Copy code

git add .

git commit -m "Incorporated review feedback"

1. **Push the changes to your branch:**

bash

Copy code

git push origin feature-branch

**Example of Updating**

After updating the code, you would:

bash

Copy code

git add .

git commit -m "Fixed issue #123 based on review"

git push origin feature-branch

This guide should provide a comprehensive overview of the most commonly used Git commands, along with their explanations and examples, including advanced topics such as conflict resolution, squashing commits, and merging into the main branch. Feel free to explore and practice these commands to get comfortable with Git!