## **9. Git Tagging (Versioning)**

Git **tags** are used to mark important milestones, such as releases (v1.0, v2.0). They are like commit references but don’t change over time.

### **9.1 Creating a Tag**

To create a lightweight tag:

git tag v1.0

This marks the current commit as v1.0.

To create an annotated tag with a message:

git tag -a v1.0 -m "Release version 1.0"

### **9.2 Listing Tags**

git tag

**Example Output:**

v1.0

v1.1

v2.0

### **9.3 Pushing Tags to Remote**

By default, tags are not pushed with git push. To push them:

git push origin v1.0

To push all tags:

git push --tags

### **9.4 Deleting a Tag**

Locally:

git tag -d v1.0

Remotely:

git push origin --delete v1.0

### **Example Use Case**

1. After finishing development for version 1.0:
2. git commit -m "Final commit for version 1.0"
3. git tag -a v1.0 -m "Release version 1.0"
4. git push origin v1.0
5. Later, when v1.1 is ready:
6. git tag -a v1.1 -m "Minor updates"
7. git push origin v1.1

### **Summary**

## **10. Git Ignore (.gitignore File)**

.gitignore is used to exclude files from being tracked in Git.

### **10.1 Creating a .gitignore File**

touch .gitignore

Then open it and add files or folders to ignore.

### **10.2 Common .gitignore Rules**

# Ignore node\_modules folder

node\_modules/

# Ignore log files

\*.log

# Ignore environment files

.env

# Ignore compiled files

\*.class

\*.pyc

# Ignore IDE and editor settings

.vscode/

.idea/

### **10.3 Applying .gitignore**

After adding files to .gitignore, commit the changes:

git add .gitignore

git commit -m "Added .gitignore"

### **10.4 Ignore an Already Tracked File**

If a file is already tracked but needs to be ignored:

git rm --cached file.txt

git commit -m "Removed file.txt from tracking"

### **Summary**

## **11. GitHub Workflow**

### **11.1 Creating a Repository on GitHub**

1. Go to [GitHub](https://github.com/)
2. Click on "New Repository."
3. Enter a name, choose "Public" or "Private," and click "Create repository."

### **11.2 Connecting Local Repo to GitHub**

After creating a local repository:

git remote add origin https://github.com/kundan/my-project.git

git branch -M main

git push -u origin main

### **11.3 Cloning a GitHub Repository**

To copy an existing GitHub repository:

git clone https://github.com/kundan/my-project.git

### **11.4 Pulling Latest Changes from GitHub**

git pull origin main

### **11.5 Pushing Local Changes to GitHub**

git add .

git commit -m "Updated project"

git push origin main

### **11.6 Creating a Pull Request (PR)**

1. Push a new branch:
2. git checkout -b feature-branch
3. git push origin feature-branch
4. Go to GitHub, open the repository, and create a **Pull Request** (PR).
5. Wait for review and merge it.

### **11.7 Forking a Repository**

1. Go to any public repository on GitHub.
2. Click "Fork" (creates a copy in your account).
3. Clone it locally:
4. git clone https://github.com/yourusername/forked-repo.git

### **Summary**

Would you like a detailed example for GitHub collaboration?

## **8. Git Branching and Merging**

Branches allow you to work on new features or fixes without affecting the main codebase. Merging brings these changes back into the main branch.

### **8.1 Creating a Branch**

To create a new branch:

git branch feature-branch

To switch to the new branch:

git checkout feature-branch

Or, create and switch in one step:

git checkout -b feature-branch

### **8.2 Listing Branches**

To see all branches:

git branch

**Example Output:**

\* main

feature-branch

The \* indicates the current branch.

### **8.3 Merging a Branch**

First, switch to the branch you want to merge into (e.g., main):

git checkout main

git merge feature-branch

### **8.4 Resolving Merge Conflicts**

If a merge conflict occurs:

<<<<<<< HEAD

Existing code in main branch

=======

New changes from feature-branch

>>>>>>> feature-branch

Manually edit the file, keeping the desired changes. After resolving:

git add conflicted-file.txt

git commit -m "Resolved merge conflict"

### **8.5 Deleting a Branch**

After merging, you can delete the branch:

git branch -d feature-branch

If the branch is not merged yet and you want to force delete:

git branch -D feature-branch

### **8.6 Pushing a Branch to Remote**

To push a branch to GitHub:

git push origin feature-branch

To delete it from remote:

git push origin --delete feature-branch

### **Summary**

## **12. Git Rebase (Alternative to Merging)**

Git **rebase** moves your branch commits on top of another branch, keeping a cleaner history.

### **12.1 Rebasing a Branch**

git checkout feature-branch

git rebase main

This applies feature-branch commits on top of main.

### **12.2 Handling Rebase Conflicts**

If a conflict happens, Git stops and asks you to resolve it manually. After fixing:

git add conflicted-file.txt

git rebase --continue

If you want to cancel the rebase:

git rebase --abort

### **12.3 Difference Between Merge and Rebase**

* **Merge:** Creates a new commit combining changes from two branches.
* **Rebase:** Moves commits to a new base, making history linear.

### **Summary**

## **13. Git Reset and Revert**

Used to undo changes.

### **13.1 Undo Last Commit (Soft Reset)**

Keeps changes but removes the commit:

git reset --soft HEAD~1

### **13.2 Undo Last Commit (Hard Reset)**

Deletes commit and changes:

git reset --hard HEAD~1

### **13.3 Reverting a Commit**

If you want to undo a commit without losing history:

git revert <commit-hash>

This creates a new commit that undoes changes.

### **Summary**

## **14. Git Cherry-Pick (Apply Specific Commit)**

Use **cherry-pick** to apply specific commits from another branch.

### **14.1 Cherry-Picking a Commit**

git cherry-pick <commit-hash>

This applies that commit to the current branch.

### **Example Use Case**

If you accidentally committed to feature-branch but need it in main:

git checkout main

git cherry-pick <commit-hash>

### **Summary**

## **15. Git Bisect (Find Bug in Commits)**

Helps find which commit introduced a bug.

### **15.1 Start Bisecting**

git bisect start

git bisect bad # Mark current commit as bad

git bisect good <commit-hash> # Mark an old commit as good

Git will now check commits between them.

### **15.2 After Finding the Bad Commit**

git bisect reset

### **Summary**

## **16. Git Hooks (Automate Actions)**

Hooks allow you to execute scripts before or after Git events.

### **16.1 Using a Pre-Commit Hook**

1. Navigate to .git/hooks/
2. Create a new script:
3. touch pre-commit
4. chmod +x pre-commit
5. Add a simple script:
6. #!/bin/sh
7. echo "Pre-commit hook running"
8. exit 1 # Stops commit if needed

### **Summary**

## **17. Git Submodules (Include Other Repositories)**

Submodules allow you to include another Git repository inside your project.

### **17.1 Adding a Submodule**

git submodule add <repo-url> path/

### **17.2 Cloning a Repo with Submodules**

git clone --recurse-submodules <repo-url>

### **17.3 Updating Submodules**

git submodule update --init --recursive

### **Summary**

## **Final Git Cheatsheet**

Would you like any practical examples or code demos?