DevOps Shack

Some Best Practices for Using Git: A Comprehensive Guide with Examples

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

1. Use Meaningful Commit Messages

A good commit message should accurately describe the changes made and the reason for the change.

Example:

```
git commit -m "Fix issue with user login by adding validation checks"
```

2. Commit Often with Small Changes

Frequent commits with small changes make it easier to track progress and find specific changes that introduced issues.

Example:

```
# Make a small change
git add .
git commit -m "Update README with installation instructions"

# Make another small change
git add .
git commit -m "Fix typo in documentation"
```

3. Use Branches for New Features or Fixes

Branches allow you to develop features and fixes independently of the main codebase.

Example:

```
# Create a new branch for a feature
git checkout -b feature/new-login-system

# Make changes and commit them
git add .
git commit -m "Add new login system"

# Push the branch to the remote repository
git push origin feature/new-login-system
```

4. Regularly Merge or Rebase

Keep your branches up-to-date with the main branch by regularly merging or rebasing.

Example:

```
# Merge main branch into your feature branch
git checkout feature/new-login-system
git merge main

# Or rebase your feature branch onto main
git checkout feature/new-login-system
git rebase main
```

5. Avoid Committing Sensitive Data

Use `.gitignore` to exclude sensitive or unnecessary files from being committed.
Example:

```
# Add sensitive files to .gitignore
echo "config.yml" >> .gitignore
echo ".env" >> .gitignore

# Commit the .gitignore file
git add .gitignore
git commit -m "Add .gitignore to exclude sensitive files"
```

6. Review Changes Before Committing

Use `git status` and `git diff` to review your changes before committing them.
Example:

```
# Check the status of your working directory
git status

# View the changes you made
git diff

# Add and commit changes after review
```

```
git add .
git commit -m "Refactor code for better readability"
```

7. Use Pull Requests for Code Reviews

Use pull requests to review and discuss code changes before merging them into the main branch.

Example:

```
# Create a new branch and push it to remote
git checkout -b feature/add-logout-button
git push origin feature/add-logout-button

# Open a pull request on GitHub (or another platform)
```

8. Keep Your Repository Clean

Regularly clean up unnecessary branches and tags.

Example:

```
# Delete a merged branch
git branch -d feature/old-feature

# Delete a remote branch
git push origin --delete feature/old-feature

# Prune remote-tracking branches that no longer exist on the remote
git fetch --prune
```

9. Use Tags for Releases

Tag important points in your history, such as releases.

Example:

```
# Create a tag for version 1.0.0
git tag -a v1.0.0 -m "Release version 1.0.0"

# Push tags to remote
git push origin --tags
```

10. Squash Commits When Necessary

Squash commits to combine multiple commits into a single, more meaningful commit.

Example:

```
# Rebase and interactively squash commits
git rebase -i HEAD~3

# Follow the prompts to squash commits
```

11. Handle Merge Conflicts Properly

Resolve merge conflicts carefully and test the changes.

Example:

```
# Merge branch and encounter conflict
git merge main

# Edit conflicting files to resolve conflicts
# Mark conflicts as resolved
git add .

# Commit the merge
git commit -m "Resolve merge conflict"
```

12. Use Descriptive Branch Names

Descriptive branch names help you and your team understand the purpose of the branch.

Example:

```
# Create a new branch with a descriptive name
git checkout -b bugfix/fix-login-error
```

13. Use Git Hooks for Automation

Automate tasks like running tests or linting code with Git hooks.

Example:

```
# Create a pre-commit hook to run tests
echo "#!/bin/sh
make test" > .git/hooks/pre-commit

# Make the hook executable
chmod +x .git/hooks/pre-commit
```

14. Stash Changes When Necessary

Use `git stash` to save changes temporarily without committing them.
Example:

```
# Stash your changes
```

```
git stash

# Apply stashed changes later
git stash apply
```

15. Document Your Workflow

Document your Git workflow in a CONTRIBUTING.md file to help new contributors.

Example:

```
# CONTRIBUTING.md

## Branch Naming
- Use feature/ for new features
- Use bugfix/ for bug fixes
- Use hotfix/ for critical hotfixes

## Commit Messages
- Use present tense ("Add feature" not "Added feature")
- Keep messages brief but descriptive
```

16. Use Git Aliases for Common Commands

Create Git aliases to save time on frequently used commands.

Example:

```
# Add alias for git status
git config --global alias.st status

# Use the alias
git st
```

17. Backup Your Repositories

Regularly backup your repositories to avoid data loss.

Example:

```
# Clone your repository as a backup
git clone --mirror https://github.com/username/repository.git backup-
repository.git
```

18. Use `git bisect` to Find Bugs

Use `git bisect` to identify the commit that introduced a bug.

Example:

```
# Start bisecting
git bisect start

# Mark the current commit as bad
```

```
git bisect bad

# Mark an earlier commit as good
git bisect good v1.0.0

# Follow the prompts to test each commit
```

19. Keep Commit History Clean

Use interactive rebase to edit commit history before sharing it.

Example:

```
# Rebase interactively
git rebase -i HEAD~4

# Follow the prompts to edit commit messages or squash commits
```

20. Learn and Use Advanced Git Features

Take the time to learn advanced Git features like `cherry-pick`, `reflog`, and `submodules`.

Example:

```
# Cherry-pick a specific commit
git cherry-pick abc1234

# View the reflog to recover lost commits
git reflog

# Add a submodule
git submodule add https://github.com/username/repository.git
path/to/submodule
```

21. Use `.gitkeep` for Empty Directories

Git doesn't track empty directories by default. Use a `.gitkeep` file to keep these directories.

Example:

```
# Create an empty directory
mkdir empty-directory

# Add a .gitkeep file
touch empty-directory/.gitkeep

# Add and commit the directory
git add empty-directory/.gitkeep
git commit -m "Add empty directory with .gitkeep file"
```

22. Use `git blame` to Find the Author of Changes

`git blame` shows the author of each line of a file.

Example:

```
# Find who last modified each line of a file
git blame filename.txt
```

23. Use `git log` for Detailed History

`git log` provides a detailed history of commits.
Example:

```
# Show commit history
git log

# Show a brief history
git log --oneline

# Show a graphical history
git log --graph --oneline --all
```

24. Use `git clean` to Remove Untracked Files

`git clean` removes untracked files from your working directory.
Example:

```
# Preview what would be removed
git clean -n

# Remove untracked files
git clean -f

# Remove untracked directories
git clean -fd
```

25. Use Git GUI Clients

Sometimes a GUI client can be more convenient for complex tasks.

Example:

- SourceTree
- GitKraken
- GitHub Desktop

26. Revert Commits When Necessary

Use `git revert` to undo a commit by creating a new commit.
Example:

```
# Revert a commit
git revert abc1234

# Push the revert commit
git push origin main
```

27. Use Git's Built-in Help

Use Git's built-in help system to learn about commands.

Example:

```
# Get help for a command
git help commit

# Search for a command
git help -g
```

28. Use Signed Commits

Sign your commits for added security and authenticity.

Example:

```
# Configure GPG signing
git config --global user.signingkey YOUR_KEY_ID

# Sign a commit
git commit -S -m "Signed commit"
```

29. Use `git stash` for Temporary Work

`git stash` is useful for temporarily saving changes you don't want to commit yet.

Example:

```
#

 Stash changes
git stash

# List stashed changes
git stash list

# Apply stashed changes
git stash apply stash@{0}

# Drop a stash
git stash drop stash@{0}
```

30. Use `git tag` to Mark Releases

Tags are useful for marking important points in your project's history.

Example:

```
# Create a lightweight tag
git tag v1.0.0

# Create an annotated tag
git tag -a v1.0.0 -m "Release version 1.0.0"

# Push tags to remote
git push origin --tags
```

31. Use `git remote` for Managing Remotes

Manage multiple remote repositories easily with `git remote`.
Example:

```
# Add a remote
git remote add upstream https://github.com/other/repo.git

# Fetch from a remote
git fetch upstream

# View remotes
git remote -v

# Remove a remote
git remote remove upstream
```

32. Use `git fetch` and `git pull` Correctly

Understand the difference between `git fetch` and `git pull`.
Example:

```
# Fetch updates from the remote repository
git fetch origin

# Merge updates into your local branch
git merge origin/main

# Fetch and merge in one step
git pull origin main
```

33. Understand Git's Object Model

Understanding Git's object model (commits, trees, blobs) can help you use it more effectively.

Example:

```
# View a commit's tree object
git cat-file -p HEAD^{tree}

# View a blob object
git cat-file -p HEAD:README.md
```

34. Use `git cherry-pick` to Apply Specific Commits

`git cherry-pick` allows you to apply specific commits from one branch to another.
Example:

```
# Cherry-pick a commit
git cherry-pick abc1234

# Resolve conflicts if necessary
# Continue cherry-pick
git cherry-pick --continue
```

35. Use `git reflog` to Recover Lost Commits

`git reflog` can help you recover commits that are no longer reachable.

Example:

```
# View reflog
git reflog

# Checkout a commit from reflog
git checkout HEAD@{2}
```

36. Use Git Submodules for Nested Repositories

Submodules allow you to keep a Git repository as a subdirectory of another Git repository.

Example:

```
# Add a submodule
git submodule add https://github.com/other/repo.git path/to/submodule

# Initialize submodules
git submodule update --init --recursive
```

37. Use `git diff` for Comparing Changes

`git diff` is a powerful tool for comparing changes.

Example:

```
# Compare working directory changes
git diff

# Compare staged changes
git diff --staged

# Compare changes between commits
git diff commit1 commit2
```

38. Use `git reset` to Undo Changes

`git reset` can be used to undo changes in your working directory or staging area.

Example:

```
# Unstage files
git reset HEAD filename.txt

# Reset to a previous commit
git reset --hard abc1234
```

39. Use `git archive` to Create Archives

`git archive` allows you to create a tar or zip archive of your repository.

Example:

```
# Create a tar archive
git archive --format=tar --output=archive.tar HEAD

# Create a zip archive
git archive --format=zip --output=archive.zip HEAD
```

40. Use `git rm` to Remove Files

`git rm` removes files from your working directory and staging area.

Example:

```
# Remove a file
git rm filename.txt

# Commit the removal
git commit -m "Remove filename.txt"

# Remove a file but keep it in working directory
git rm --cached filename.txt
```

41. Use `git mv` to Rename or Move Files

`git mv` renames or moves files and stages the changes.

Example:

```
# Rename a file
git mv oldname.txt newname.txt

# Commit the change
git commit -m "Rename oldname.txt to newname.txt"
```

42. Use `git shortlog` for Summarizing Commits

`git shortlog` summarizes commit history by author.

Example:

```
# Summarize commits
git shortlog

# Summarize commits for a specific branch
git shortlog branch-name
```

43. Use `git describe` to Describe Commits

`git describe` gives a human-readable name to a commit.

Example:

```
# Describe a commit
git describe

# Describe a specific commit
git describe abc1234
```

44. Use `git ls-tree` to List Tree Objects

`git ls-tree` lists the contents of a tree object.

Example:

```
# List the contents of a commit
git ls-tree HEAD

# List the contents of a directory
git ls-tree HEAD:path/to/directory
```

45. Use `git ls-files` to List Tracked Files

`git ls-files` lists all tracked files in the working directory.

Example:

```
# List all tracked files
git ls-files

# List files ignored by .gitignore
git ls-files --ignored
```

46. Use `git update-index` to Manage Index

`git update-index` updates the Git index (staging area).

Example:

```
# Mark a file as assume unchanged
git update-index --assume-unchanged filename.txt

# Unmark a file as assume unchanged
git update-index --no-assume-unchanged filename.txt
```

47. Use `git show` to View Object Details

`git show` displays details about objects (commits, trees, blobs).

Example:

```
# Show details of a commit
git show abc1234

# Show details of a tag
git show v1.0.0
```

48. Use `git filter-branch` for History Rewriting

`git filter-branch` allows for rewriting Git history.

Example:

```
# Rewrite history to remove a file
git filter-branch --tree-filter 'rm -f filename.txt' HEAD
```

49. Use `git apply` to Apply Patches

`git apply` applies a patch file to your working directory.

Example:

```
# Apply a patch
git apply patchfile.patch
```

50. Use `git am` to Apply Commit Patches

`git am` applies patches created by `git format-patch`.

Example:

```
# Apply a patch file
git am patchfile.patch
```

51. Use `git format-patch` to Create Patches

`git format-patch` creates patch files from commits.

Example:

```
# Create patches for the last 3 commits
git format-patch -3
```

52. Use `git send-email` to Send Patches via Email

`git send-email` sends patch files via email.

Example:

```
# Send a patch file via email
git send-email patchfile.patch
```

53. Use `git rerere` to Reuse Recorded Resolutions

`git rerere` remembers how you resolved conflicts.

Example:

```
# Enable rerere
git config --global rerere.enabled true

# Resolve a conflict and let rerere remember it
git rerere
```

54. Use `git maintenance` for Optimizing

`git maintenance` performs maintenance tasks to optimize repository performance.

Example:

```
# Run maintenance tasks
git maintenance run
```

55. Use `git gc` for Garbage Collection

`git gc` cleans up unnecessary files and optimizes the local repository.

Example:

```
# Run garbage collection
git gc
```

56. Use `git fsck` for Repository Integrity

`git fsck` checks the integrity of the repository.

Example:

```
# Check repository integrity
git fsck
```

57. Use `git verify-commit` to Verify Signed Commits

`git verify-commit` verifies GPG-signed commits.

Example:

```
# Verify a signed commit
git verify-commit abc1234
```

58. Use `git verify-tag` to Verify Signed Tags

`git verify-tag` verifies GPG-signed tags.

Example:

```
# Verify a signed tag
git verify-tag v1.0.0
```

59. Use `git replace` to Replace Objects

`git replace` allows you to replace objects in Git history.
Example:

```
# Replace a commit with another commit
git replace old_commit new_commit
```

60. Use `git notes` to Annotate Commits

`git notes` adds notes to commits without changing the commit itself.
Example:

```
# Add a note to a commit
git notes add -m "This commit needs further review" abc1234

# Show notes
git notes show abc1234
```

61. Use `git whatchanged` for Detailed

Logs `git whatchanged` shows the commit log with file changes.
Example:

```
# Show detailed log with file changes
git whatchanged
```

62. Use `git daemon` to Share Repositories

`git daemon` shares repositories over the network.
Example:

```
# Start a Git daemon
git daemon --reuseaddr --base-path=/path/to/repositories --export-all
```

63. Use `git instaweb` to Browse Repositories

`git instaweb` sets up a web interface for browsing repositories.
Example:

```
# Start the web interface
git instaweb
```

64. Use `git help` for Built-in Documentation

Git has extensive built-in documentation accessible via `git help`.
Example:

```
# Get help for a specific command
git help commit
```

65. Use `git config` to Customize Git

`git config` customizes Git settings.
Example:

```
# Set global username
git config --global user.name "Your Name"

# Set global email
git config --global user.email "your.email@example.com"
```

66. Use `git grep` to Search in Repositories

`git grep` searches for text in your repository.

Example:

```
# Search for a string in the repository
git grep "search string"
```

67. Use `git show-ref` to List References

`git show-ref` lists references in the repository.

Example:

```
# Show all references
git show-ref
```

68. Use `git symbolic-ref` for Symbolic References

`git symbolic-ref` reads and changes symbolic references.

Example:

```
# Show the current HEAD reference
git symbolic-ref HEAD

# Change HEAD to a new branch
git symbolic-ref HEAD refs/heads/new-branch
```

69. Use `git verify-pack` to Verify Packfiles

`git verify-pack` verifies packfiles in the repository.

Example:

```
# Verify packfiles
git verify-pack -v .git/objects/pack/pack-*.idx
```

70. Use `git fast-import` for Fast Import

`git fast-import` imports data into Git quickly.

Example:

```
# Use fast-import to import data
git fast-import < datafile
```

71. Use `git apply` with Patches

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

Example:

```
# Apply a patch
git apply patchfile.patch
```

72. Use `git am` with Email Patches

`git am` applies patches from email files.

Example:

```
# Apply patches from an mbox file
git am < mboxfile
```

73. Use `git archive` for Exporting

`git archive` creates an archive of files in the repository.
Example:

```
# Create a tar archive
git archive --format=tar HEAD > archive.tar

# Create a zip archive
git archive --format=zip HEAD > archive.zip
```

74. Use `git branch` for Managing Branches

`git branch` lists, creates, and deletes branches.
Example:

```
# List branches
git branch

# Create a new branch
git branch new-branch

# Delete a branch
git branch -d old-branch
```

75. Use `git bundle` for Backup and Transfer

`git bundle` creates a single file bundle of a repository.
Example:

```
# Create a bundle
git bundle create repository.bundle --all

# Clone from a bundle
git clone repository.bundle
```

76. Use `git cat-file` to View Object Content

`git cat-file` displays the content of repository objects.
Example:

```
# Show the content of a commit
git cat-file -p HEAD

# Show the content of a blob
git cat-file -p HEAD:README.md
```

77. Use `git checkout` for Switching Branches

`git checkout` switches branches or restores working directory files.
Example:

```
# Switch to a branch
git checkout main

# Restore a file
git checkout HEAD -- filename.txt
```

78. Use `git cherry` to Find Applied Changes

`git cherry` compares two branches to see what changes have been applied.
Example:

```
# Compare changes between two branches
git cherry main feature-branch
```

79. Use `git clone` for Repository Copying

`git clone` creates a copy of an existing repository.

Example:

```
# Clone a repository
git clone https://github.com/username/repository.git
```

80. Use `git commit` for Saving Changes

`git commit` records changes to the repository.

Example:

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

# Amend the last commit
git commit --amend
```

81. Use `git describe` for Tagging

`git describe` gives a human-readable name to a commit.

Example:

```
# Describe the current commit
git describe
```

82. Use `git fetch` for Remote Updates

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

Example:

```
# Fetch updates from a remote repository
git fetch origin
```

83. Use `git format-patch` for Patches

`git format-patch` prepares patches for emailing.

Example:

```
# Create patches for the last 3 commits
git format-patch -3
```

84. Use `git fsck` for Integrity Checking

`git fsck` verifies the connectivity and validity of objects.

Example:

```
# Check the repository
git fsck
```

85. Use `git gc` for Cleanup

`git gc` performs garbage collection.

Example:

```
# Run garbage collection
git gc
```

86. Use `git grep` for Searching

`git grep` searches for strings in the repository.
Example:

```
# Search for a string
git grep "search string"
```

87. Use `git init` for New Repositories

`git init` creates an empty Git repository.
Example:

```
# Initialize a new repository
git init
```

88. Use `git log` for History

`git log` shows the commit history.
Example:

```
# Show commit history
git log
```

89. Use `git merge` for Combining Branches

`git merge` joins two or more development histories together.
Example:

```
# Merge a branch into the current branch
git merge feature-branch
```

90. Use `git mv` for Renaming

`git mv` moves or renames a file, directory, or symlink.
Example:

```
# Rename a file
git mv oldname.txt newname.txt
```

91. Use `git pull` for Fetching and Merging

`git pull` fetches and integrates changes from a remote repository.
Example:

```
# Pull changes from a remote repository
git pull origin main
```

92. Use `git push` for Updating Remotes

`git push` updates remote refs along with associated objects.
Example:

```
# Push changes to a remote repository
git push origin main
```

93. Use `git rebase` for Reapplying Commits

`git rebase` moves or combines a sequence of commits.
Example:

```
# Rebase the current branch onto another branch
git rebase main
```

94. Use `git reflog` for Reflog Entries

`git reflog` manages the reflog of the repository.

Example:

```
# Show the reflog
git reflog
```

95. Use `git remote` for Remote Repositories

`git remote` manages tracked repositories.

Example:

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

96. Use `git reset` for Undoing Changes

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

Example:

```
# Unstage a file
git reset HEAD filename.txt
```

97. Use `git revert` for Reverting Commits

`git revert` creates a new commit that undoes changes from a previous commit.

Example:

```
# Revert a commit
git revert abc1234
```

98. Use `git rm` for Removing Files

`git rm` removes files from the working directory and index.

Example:

```
# Remove a file
git rm filename.txt
```

99. Use `git shortlog` for Summarized Logs

`git shortlog` summarizes git log output.

Example:

```
# Show a summarized log
git shortlog
```

100. Use `git status` for Status

`git status` shows the working tree status.

Example:

```
# Show the working tree status
git status
```