AUTHOR: IMTIAZ ADAR

EMAIL: IMTIAZADAROFFICIAL@GMAIL.COM

GIT COMMANDS – PRO LEVEL

1. ESSENTIAL DAILY COMMANDS

git init

Purpose: Initialize a new Git repository

```
# Initialize in current directory
git init

# Initialize with specific directory name
git init project-name

# Initialize bare repository (for servers)
git init –bare
```

git clone

Purpose: Clone an existing repository

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

# Clone with specific name
git clone https://github.com/user/repo.git my-project

# Clone specific branch only
git clone -b develop --single-branch https://github.com/user/repo.git

# Clone with depth (shallow clone)
git clone --depth 1 https://github.com/user/repo.git
```

git status

Purpose: Show working tree status

```
# Basic status
git status

# Short format
git status -s
```

```
# Show branch information
git status -b

# Ignore submodules
git status --ignore-submodules
```

git add

Purpose: Add files to staging area

```
# Add specific file
git add filename.js

# Add all changes
git add .

# Add interactively
git add -p

# Add all tracked files
git add -u
```

```
# Add with glob pattern
git add src/*.js
```

git commit

Purpose: Record changes to repository

```
# Commit with message
git commit -m "Add feature X"

# Amend previous commit
git commit --amend -m "Updated commit message"

# Commit all tracked files without separate add
git commit -am "Quick commit"

# Sign commit with GPG
git commit -S -m "Signed commit"

# Commit with specific date
git commit --date="2024-01-01" -m "Backdated commit"
```

git push

Purpose: Update remote refs with local commits

```
# Push to current branch
git push

# Push to specific remote branch
git push origin main

# Push with tags
git push --tags

# Force push (use with caution!)
git push -f

# Push to different branch name
git push origin local-branch:remote-branch

# Push and set upstream
git push -u origin branch-name
```

git pull

Purpose: Fetch from and integrate with another repository

```
# Pull from remote
git pull

# Pull with rebase instead of merge
git pull --rebase

# Pull specific branch
git pull origin develop

# Pull with auto-stash
git pull –autostash
```

git fetch

Purpose: Download objects and refs from another repository

```
# Fetch all remotes
git fetch

# Fetch specific remote
```

git fetch origin

Fetch all branches and tags

git fetch --all

Fetch and prune deleted branches

git fetch -p

Fetch specific branch

git fetch origin main

2. BRANCHING & MERGING

git branch

Purpose: List, create, or delete branches

List local branches

git branch

List all branches (remote included)

git branch -a

Create new branch

```
git branch feature-x

# Delete branch
git branch -d feature-x

# Force delete unmerged branch
git branch -D feature-x

# Move/rename branch
git branch -m old-name new-name

# Set upstream branch
git branch -u origin/main
```

git checkout

Purpose: Switch branches or restore working tree files

```
# Switch to branch
git checkout develop

# Create and switch to new branch
git checkout -b feature-y
```

```
# Switch to previous branch
git checkout -

# Discard changes in file
git checkout -- filename.js

# Switch to specific commit
git checkout a1b2c3d
```

git switch (Git 2.23+)

Purpose: Switch branches (modern alternative to checkout)

```
# Switch to existing branch
git switch develop

# Create and switch to new branch
git switch -c feature-z

# Switch to previous branch
git switch –
```

git merge

Purpose: Join two or more development histories together

```
# Merge branch into current
git merge feature-x

# Merge with no-fast-forward (always create merge commit)
git merge --no-ff feature-x

# Merge with squash (combine all commits into one)
git merge --squash feature-x

# Abort merge in progress
git merge --abort

# Merge with specific strategy
git merge -s recursive -X theirs feature-x
```

git rebase

Purpose: Reapply commits on top of another base tip

```
# Rebase current branch onto main
```

```
git rebase main

# Interactive rebase (last 5 commits)
git rebase -i HEAD~5

# Continue after resolving conflicts
git rebase --continue

# Skip problematic commit
git rebase --skip

# Abort rebase
git rebase --abort

# Autosquash during interactive rebase
git rebase -i –autosquash
```

git cherry-pick

Purpose: Apply changes from existing commits

```
# Cherry-pick specific commit
git cherry-pick abc123
```

Cherry-pick range

git cherry-pick abc123..def456

Cherry-pick with no commit

git cherry-pick -n abc123

Continue after conflict resolution

git cherry-pick –continue

3. HISTORY & INSPECTION

git log

Purpose: Show commit logs

Basic log

git log

One line per commit

git log --oneline

Graph view

git log --graph --oneline

```
# Show changes in files
git log --stat

# Show patch (diff)
git log -p

# Filter by author
git log --author="John"

# Filter by date
git log --since="2024-01-01"

# Filter by content
git log -S "function_name"

# Show branch information
git log --all --decorate --oneline --graph (alias: git adog)
```

git diff

Purpose: Show changes between commits, commit and working tree, etc.

```
# Unstaged changes
git diff

# Staged changes
git diff --staged

# Compare branches
git diff main..develop

# Compare specific files
git diff HEAD~2 HEAD -- filename.js

# Word diff instead of line diff
git diff --word-diff

# Show only names of changed files
git diff --name-only
```

git show

Purpose: Show various types of objects

```
# Show commit details
git show abc123

# Show file at specific commit
git show abc123:filename.js

# Show diff of commit
git show --stat abc123
```

git blame

Purpose: Show what revision and author last modified each line

```
# Blame file
git blame filename.js

# Blame with line numbers
git blame -L 10,20 filename.js

# Blame with commit hash
git blame -s filename.js

# Ignore whitespace changes
```

```
git blame -w filename.js
```

git bisect

Purpose: Use binary search to find commit that introduced a bug

```
# Start bisect session
git bisect start

# Mark current commit as bad
git bisect bad

# Mark known good commit
git bisect good abc123

# Automatically run test script
git bisect run npm test

# Reset bisect
git bisect reset
```

4. REWRITING HISTORY

git reset

Purpose: Reset current HEAD to specified state

```
# Soft reset (keep changes in staging)
git reset --soft HEAD~1

# Mixed reset (keep changes in working directory)
git reset --mixed HEAD~1

# Hard reset (discard all changes)
git reset --hard HEAD~1

# Reset specific file
git reset HEAD filename.js

# Reset to specific commit
git reset --hard abc123
```

git revert

Purpose: Create new commit that undoes specified commit

```
# Revert specific commit
git revert abc123

# Revert without committing
git revert -n abc123

# Revert merge commit
git revert -m 1 merge-commit-hash
```

git clean

Purpose: Remove untracked files from working tree

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

# Remove untracked files
git clean -f

# Remove untracked directories
git clean -fd
```

Interactive clean

git clean -i

5. REMOTE OPERATIONS

git remote

Purpose: Manage set of tracked repositories

List remotes

git remote -v

Add remote

git remote add origin https://github.com/user/repo.git

Remove remote

git remote remove origin

Rename remote

git remote rename origin upstream

Show remote details

```
git remote show origin

# Update remote URL
git remote set-url origin new-url
```

git push (Advanced)

```
# Push and delete remote branch
git push origin --delete branch-name

# Push all branches
git push --all

# Push with specific refspec
git push origin HEAD:refs/heads/main
```

git submodule

Purpose: Manage nested repositories

```
# Add submodule
git submodule add https://github.com/user/lib.git

# Initialize submodules
git submodule init

# Update submodules
git submodule update

# Update recursively
git submodule update --init --recursive

# Foreach command
git submodule foreach git pull origin main
```

6. STASHING

git stash

Purpose: Stash changes in dirty working directory

```
# Stash changes
git stash

# Stash with message
```

```
git stash push -m "Work in progress"

# List stashes
git stash list

# Apply stash
git stash apply

# Apply specific stash
git stash apply stash@{2}

# Pop stash (apply and remove)
git stash pop

# Create branch from stash
git stash branch new-branch stash@{1}

# Show stash diff
git stash show -p

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

```
# Clear all stashes
git stash clear

# Stash specific files
git stash push filename.js

# Stash untracked files
git stash -u
```

7. TAGS & RELEASES

git tag

Purpose: Create, list, delete tags

```
# List tags
git tag

# Create lightweight tag
git tag v1.0.0

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

# Create signed tag
```

```
git tag -s v1.0.0 -m "Signed release"

# Show tag details
git show v1.0.0

# Delete tag
git tag -d v1.0.0

# Delete remote tag
git push origin --delete v1.0.0

# Push all tags
git push --tags

# Push specific tag
git push origin v1.0.0

# Checkout tag
git checkout v1.0.0
```

8. ADVANCED OPERATIONS

git reflog

Purpose: Record when tips of branches were updated

Show reflog

git reflog

Show reflog for specific branch

git reflog show main

Expire old reflog entries

git reflog expire --expire=90.days

git worktree

Purpose: Manage multiple working trees

Add new worktree

git worktree add ../hotfix hotfix-branch

List worktrees

git worktree list

Remove worktree

```
git worktree remove hotfix-branch

# Prune worktrees
git worktree prune
```

git filter-branch / git filter-repo

Purpose: Rewrite branch history (dangerous but powerful)

```
# Remove file from history
git filter-branch --tree-filter 'rm -f password.txt' HEAD

# Change email in commits
git filter-branch --commit-filter '
   if [ "$GIT_AUTHOR_EMAIL" = "old@email.com" ];
   then
      GIT_AUTHOR_EMAIL="new@email.com";
      git commit-tree "$@";
   else
      git commit-tree "$@";
   fi' HEAD
```

git gc

Purpose: Cleanup unnecessary files and optimize repository

```
# Garbage collect
git gc

# Aggressive cleanup
git gc --aggressive

# Prune loose objects
git gc --prune=now
```

9. WORKFLOWS & BEST PRACTICES

Commit Message Convention

<type>(<scope>): <subject>

<body>

<footer>

Types: feat, fix, docs, style, refactor, test, chore

Example:

feat(auth): add OAuth2 integration

- Implement Google OAuth2 provider
- Add token refresh mechanism
- Update documentation

Closes #123

Common Aliases

```
# Add to ~/.gitconfig
[alias]
    co = checkout
    br = branch
    ci = commit
    st = status
    unstage = reset HEAD --
    last = log -1 HEAD
    lol = log --graph --oneline --decorate
    adog = log --all --decorate --oneline --graph
    dc = diff --cached
    undo = reset --soft HEAD~1
    amend = commit --amend --no-edit
    wip = commit -am "WIP"
```

Git Hooks

Common hooks in .git/hooks/:

- pre-commit: Run tests/linters before commit
- commit-msg: Validate commit message format
- pre-push: Run integration tests before push

Git Configuration

 # Set global configuration

git config --global user.name "Your Name"

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

git config --global core.editor "code --wait"

git config --global pull.rebase true

 # Set local configuration

git config user.name "Project Specific Name"

 # Show configuration

git config –list

Troubleshooting Commands

 # Fix line endings

git config --global core.autocrlf true *# Windows*

```
git config --global core.autocrlf input # Mac/Linux

# Fix permissions
git config --global core.fileMode false

# Debug SSL issues
git config --global http.sslVerify false

# Increase buffer size
git config --global http.postBuffer 524288000
```

Recovery Commands

```
# Recover deleted branch
git reflog
git checkout -b recovered-branch abc123

# Undo git add
git reset HEAD filename.js

# Recover uncommitted changes
git fsck --lost-found
```

 # Fix detached HEAD

git checkout -b new-branch

PRO TIPS

1. **Always review before pushing**:

git log --oneline origin/main..HEAD

2. **Use interactive add for selective staging**:

git add -p

3. **Keep commits small and focused**
4. **Never force push to shared branches**
5. **Use feature branches for all changes**
6. **Regularly fetch and rebase**:

git fetch && git rebase origin/main

7. **Clean up merged branches**:

git branch --merged | grep -v "*" | xargs -n 1 git branch -d

8. **Use git bisect for bug hunting**
9. **Sign important commits and tags**
10. **Backup before major operations**