1. Git Configuration

Configure your Git identity and default preferences to start using Git with your name, email, and UI settings.

```
git config --global user.name "Om Nanda"
    git config --global user.email "om.nanda@example.com"
    git config --list
    git config --global color.ui auto
```

2. Initializing & Cloning

Start a new Git project or copy an existing one from a remote server to your local machine.

```
git init
    git clone <repo-url>
    git clone --depth=1 <repo-url>  # shallow clone
```

3. Basic Workflow

Track file changes, stage them for commit, and save snapshots of your project history.

```
git status
    git add <file>
    git add .                   # stage all changes
    git commit -m "message"
    git commit -am "msg"        # add & commit tracked files
```

4. Viewing Changes

Compare file changes and review project history with diffs and logs.

```
git diff
    git diff --staged
    git log
    git log --oneline --graph --all
    git show <commit>
```

5. Branching and Merging

Create and manage isolated lines of development with branches. Merge them when work is complete.

```
git branch
    git branch <new-branch>
    git switch <branch>
    git switch -c <branch>      # create and switch
    git merge <branch>          # into current
    git branch -d <branch>      # delete
```

6. Working with Remotes

Connect your repository to others hosted online, fetch updates, and share your changes.

```
git remote -v
    git remote add origin <url>
    git fetch
    git pull origin <branch>
    git push -u origin <branch>
```

---

7. Tagging and Releases

Mark specific points in history as important, like a version release.

```
git tag
    git tag v1.0
    git tag -a v1.0 -m "Release 1"
    git push origin v1.0
```

---

8. Undoing Mistakes

Fix errors by discarding changes, reverting commits, or inspecting history.

```
git restore <file>
    git reset HEAD <file>
    git reset --hard
    git revert <commit>
    git reflog
```

---

9. Stashing Work

Temporarily store modified files to switch branches without committing.

```
git stash
    git stash save "message"
    git stash list
    git stash apply
    git stash pop
```

---

10. Comparing Branches

Identify changes between branches using diffs and logs.

```
git diff main..feature
    git diff --name-only
    git log main..feature
```

---

11. Ignoring Files

Prevent unnecessary files from being tracked using a .gitignore file.

```
Example .gitignore:

    *.log
    node_modules/
    .env
    dist/
```

---

12. Rewriting History

Reorder, squash, or clean up commits for a more readable history (use with caution).

```
git rebase main
    git rebase -i HEAD~3
    git cherry-pick <commit>
```

---

13. Cleaning Up

Free up space by removing untracked files and optimizing the repo.

```
git clean -fd
    git gc
```

---

14. Collaboration Tips

Best practices to follow when working on shared repositories with teams.

```
- Always pull before pushing
- Use branches for features
- Write meaningful commit messages
- Use rebase for clean history
- Keep main branch deployable
```

---

15. Git Hooks

Automate scripts at different points of the Git lifecycle like pre-commit checks or post-merge actions.

```
Hooks are stored in the `.git/hooks/` directory.

Examples:

    pre-commit     # Runs before a commit
    post-merge     # Runs after a merge
    pre-push       # Runs before pushing
```

```
To activate a hook, rename it (remove `.sample`) and make it executable.
```

16. Git Submodules

Track other repositories inside your project great for plugin or dependency management.

```
git submodule add <repo-url> <path>
    git submodule init
    git submodule update
    git submodule foreach git pull
```

17. Git Bisect

Find the commit that introduced a bug using binary search.

```
git bisect start
    git bisect bad
    git bisect good <commit>
    # Git checks out midpoints
    git bisect reset   # To end bisect session
```

18. Git Archive

Export a Git repository as a tar or zip archive without including the .git history.

```
git archive --format=zip --output=source.zip main
```

19. Git Blame

See who made changes to each line in a file useful for debugging or reviewing history.

```
git blame <file>
```

20. Git Shortlog

Summarize contributions by author, useful for release notes and team tracking.

```
git shortlog -sn
```

21. Git Notes

Attach notes to commits without changing history.

```
git notes add -m "Reviewed by team"
    git log --show-notes
```

22. Git Worktrees

Manage multiple working directories attached to the same repository.

```
git worktree add ../branch-folder <branch-name>
```

- Om Nanda