#_ Mastering Git: Important Notes [Tips & Tricks]

1. Combine add & commit

In Git, you can combine the git add and git commit commands using the -a flag with the commit command. This allows you to automatically stage all modified and deleted files before making a commit. Here's the command:

```
git commit -am "Your commit message"
```

2. Aliases

Git allows you to create aliases for frequently used commands, making your workflow more efficient. To set up an alias, use the git config command or directly edit the .gitconfig file. For example:

```
git config --global alias.co checkout
```

Now, you can use git co instead of git checkout.

3. Amend

The git commit --amend command lets you modify the most recent commit. If you forgot to add some changes or need to update the commit message, use this command:

```
git commit --amend
```

4. Force Push

Sometimes you may need to overwrite the remote repository's history with your local changes. Use the git push command with the -f or --force flag:

```
git push -f origin <branch_name>
```

Be cautious when using this command, as it can cause issues if others are also working on the same branch.

5. Revert

The `git revert` command is used to create a new commit that undoes the changes introduced by a specific commit. It is a safer way to undo changes compared to `git reset`. Example usage:

```
git revert <commit_hash>
```

6. Codespaces

GitHub Codespaces allows you to develop in an online, cloud-based environment. It's useful for quickly testing and collaborating on code without setting up a local development environment.

7. Stash

The `git stash` command helps you save changes that are not ready to be committed yet, allowing you to switch branches without committing everything. Later, you can apply the stash back to your working branch. Basic usage:

```
git stash
git stash apply
```

9. Pretty Logs

Improve the appearance of your Git log with pretty formatting:

```
git log --oneline --graph --decorate
```

You can customize the output further using different options for formatting.

10. Bisect

Git bisect is a helpful tool to find the commit that introduced a bug or issue. It performs a binary search through the commit history to find the faulty commit. The process involves marking good and bad commits until the problematic commit is identified:

```
git bisect start
git bisect bad
git bisect good <commit_hash>
```

11. Autosquash

When using `git rebase -i`, you can automatically squash or fixup commits using the `autosquash` feature. Set the `autosquash` option in your git config:

```
git config --global rebase.autosquash true
```

Now, when you run `git rebase -i`, the commits marked with "fixup!" or "squash!" in the message will be automatically squashed.

12. Hooks

Git hooks allow you to run custom scripts before or after specific Git actions (e.g., commit, push, merge). You can find hook scripts inside the `.git/hooks` directory of your repository.

14. Checkout to Last

If you want to quickly switch to the previously checked-out branch, you can use the following command:

```
git checkout -
```

This will switch to the branch you were on before the current branch. The hyphen acts as a reference to the previous branch.

15. Interactive Staging

The git add -p command allows you to interactively stage changes within a file. You can choose which chunks of changes you want to add to the staging area, making it easy to separate unrelated changes into separate commits.

```
git add -p
```

16. Reflog

The git reflog command shows a log of all reference updates, including branch checkouts and resets. It's helpful for recovering lost commits or branches.

```
git reflog
```

17. Cherry-pick

Cherry-picking allows you to apply a specific commit from one branch to another. This can be helpful when you want to include a single fix or feature from another branch without merging the entire branch.

```
git cherry-pick <commit_hash>
```

18. Git Worktrees

Git worktrees allow you to have multiple working directories for the same repository, enabling you to work on different branches simultaneously.

```
git worktree add <path> <branch_name>
```

19. Ignore Whitespace Changes

You can ignore whitespace changes when reviewing a diff using the `--ignore-space-change` flag.

```
git diff --ignore-space-change
```

20. Git Clean

The `git clean` command allows you to remove untracked files from your working directory.

```
git clean -n  # Dry-run, see what will be deleted
git clean -f  # Perform the actual cleanup
```

21. Interactive Rebase

During an interactive rebase, you can reorder, edit, squash, or drop commits to create a cleaner commit history.

```
git rebase -i <base_branch>
```

22. Git Bisect Skip

If a commit is too complex or ambiguous to determine if it's good or bad during a `git bisect`, you can use `git bisect skip` to skip the commit and continue the bisect process.

```
git bisect skip
```

23. Git Archive

The git archive command allows you to create a tar or zip archive of a specified branch or commit.

```
git archive -o <output_file.zip> <branch_or_commit>
```

24. Git Revert Range

You can use git revert with a range of commits to create a single revert commit for multiple commits.

```
git revert <start_commit>^..<end_commit>
```

25. Excluding Files from Commits

To exclude specific files from being committed, you can create a .gitignore file and add the filenames or patterns of files you want to ignore.

26. Git Bisect Reset

After completing a git bisect, you can reset back to the original HEAD with:

```
git bisect reset
```

27. Git Rebase Autostash

When running git rebase, you can use the --autostash option to automatically stash and apply your changes before performing the rebase.

```
git rebase --autostash <base_branch>
```

28. Git Bisect Reset

If you want to abort a `git bisect` without making any changes, you can use `git bisect reset`.

```
git bisect reset
```

29. Git Rerere

The `git rerere` (reuse recorded resolution) feature allows Git to remember how conflicts were resolved in the past, making it easier to resolve similar conflicts in the future.

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

30. Git Worktree Remove

When you no longer need a Git worktree, you can remove it using the following command:

```
git worktree remove <path>
```

31. Git Blame

The `git blame` command shows who last modified each line of a file, helping you trace changes back to their origin.

```
git blame <file_name>
```

32. Git Archive With Submodule

If your repository has submodules, you can use `git archive` to create an archive including the submodules.

```
git archive --format=zip --output=<output_file.zip> HEAD -- <submodule_path>
```

33. Git Clean Interactive

The git clean command can be run interactively, allowing you to choose which untracked files to delete.

```
git clean -i
```

34. Git Notes

Git notes allow you to attach additional information to commits without modifying the commit messages. It's useful for adding metadata or comments to commits.

```
git notes add -m "Additional information" <commit_hash>
```

35. Git Worktree List

To see a list of all linked worktrees, you can use the following command:

```
git worktree list
```

36. Git Archive Specific Paths

When creating an archive with git archive, you can specify specific files or directories to include.

```
git archive --format=zip --output=<output_file.zip> HEAD --
<file_or_directory>
```