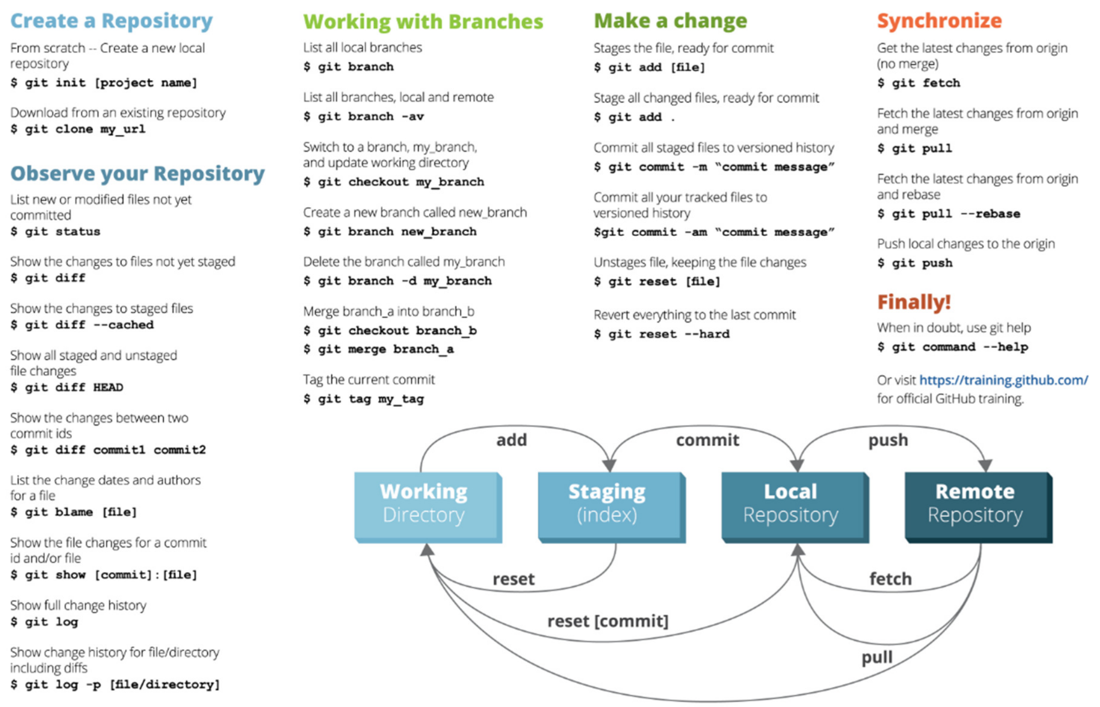
Git Cheat Sheet, our aim is to provide a handy reference tool for both beginners and experienced developers/ DevOps engineers. This **Git Cheat Sheet** not only makes it easier for newcomers to get started but also serves as a refresher for experienced professionals



Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```
Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```
Show the changes to files not yet staged
```
$ git diff
```
Show the changes to staged files
```
$ git diff --cached
```
Show all staged and unstaged file changes
```
$ git diff HEAD
```
Show the changes between two commit ids
```
$ git diff commit1 commit2
```
List the change dates and authors for a file
```
$ git blame [file]
```
Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```
Show full change history
```
$ git log
```
Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```
List all branches, local and remote
```
$ git branch -av
```
Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```
Create a new branch called new_branch
```
$ git branch new_branch
```
Delete the branch called my_branch
```
$ git branch -d my_branch
```
Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```
Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```
Stage all changed files, ready for commit
```
$ git add .
```
Commit all staged files to versioned history
```
$ git commit -m "commit message"
```
Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```
Unstages file, keeping the file changes
```
$ git reset [file]
```
Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```
Fetch the latest changes from origin and merge
```
$ git pull
```
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```
Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```
Or visit https://training.github.com/ for official GitHub training.

What is Git?

Git is the free and open-source distributed version control systems that's responsible for everything **GitHub** related that happens locally on your computer.

Understanding Version Control

Version control, also known as source control, is the technique of tracking and managing changes to codes and these are the systems that are software tools that enable software teams to manage modifications to source code as time passes.

What is GitHub?

GitHub is a widely-used Free-to-use cloud Storage platform with version control and many other essential features that specifically helps developers to manage and deploy their projects on GitHub.

Benefits of Using Git

History Tracking: Git allows you to track every change made in your project, including: who made the change and when it was made.

Collaboration: Multiple developers can be able work on the same project at the same time, and Git efficiently manages the merging of changes in code.

Branching and Merging: Git enables developers to create branches to work on new features or bug fixes and later merge them back into the main codebase.

Offline Work: Git works offline, which means you can commit changes and work on your project even without an internet connection.

Git –Commands

Here are the Git installation commands for different operating systems:

Commands	Description
Git for Windows stand-alone installer.	
$ brew install git	Install Git with **Homebrew** on Mac OS
$ sudo port selfupdate	Install Git with **MacPorts** on Mac OS
$ sudo apt-get install git	Install Command for Linux
$ git –version	Shows the current version of your Git

Git Configuration & Setup commands

Commands	Description
git config –global user.name "Your Name"	Set your username globally.
git config –global user.email "youremail@example.com"	Set your email globally.
git config –global color.ui auto –	Set to display colored output in the terminal
git help	Display the main help documentation, showing a list of commonly used Git commands.

Initializing a Repository Commands

Commands	Description
git init	Initializes a new Git repository in the current directory.
git init <directory>	Creates a new Git repository in the specified directory.
git clone <repository_url>	this Clones a repository from a remote server to your local machine.

Commands	Description
git clone –branch <branch_name> <repository_url>	Clones a specific branch from a repository.

Basic Git Commands

Commands	Description
git add <file>	Adds a specific file to the staging area.
git add . or git add –all	Adds all modified and new files to the staging area.
git status	Shows the current state of your repository, including tracked and untracked files, modified files, and branch information.
git status –ignored	Displays ignored files in addition to the regular status output.
git diff	Shows the changes between the working directory and the staging area (index).
git diff <commit1> <commit2>	Displays the differences between two commits.

Commands	Description
git diff –staged or git diff –cached	Displays the changes between the staging area (index) and the last commit.
git diff HEAD	Display the difference between the current directory and the last commit
git commit	Creates a new commit with the changes in the staging area and opens the default text editor for adding a commit message.
git commit -m “<message>” or git commit –message “<message>”	Creates a new commit with the changes in the staging area and specifies the commit message inline.
git commit -a or git commit –all	Commits all modified and deleted files in the repository without explicitly using git add to stage the changes.
git notes add	Creates a new note and associates it with an object (commit, tag, etc.).
git restore <file>	Restores the file in the working directory to its state in the last commit.

Commands	Description
git reset <commit>	Moves the branch pointer to a specified commit, resetting the staging area and the working directory to match the specified commit.
git reset –soft <commit>	Moves the branch pointer to a specified commit, preserving the changes in the staging area and the working directory.
git reset –hard <commit>	Moves the branch pointer to a specified commit, discarding all changes in the staging area and the working directory, effectively resetting the repository to the specified commit.
git rm <file>	Removes a file from both the working directory and the repository, staging the deletion.
git mv	Moves or renames a file or directory in your Git repository.

Basic Git Commands Git Commit (Updated Commands)

Commands	**Description**
git commit -m “feat: message”	Create a new commit in a Git repository with a specific message to indicate a new feature commit in the repository.
git commit -m “fix: message”	Create a new commit in a Git repository with a specific message to fix the bugs in codebases
git commit -m “chore: message”	Create a new commit in a Git repository with a specific message to show routine tasks or maintenance.
git commit -m “refactor: message”	Create a new commit in a Git repository with a specific message to change the code base and improve the structure.
git commit -m “docs: message”	Create a new commit in a Git repository with a specific message to change the documentation.
git commit -m “style: message”	Create a new commit in a Git repository with a specific message to change the styling and formatting of the codebase.
git commit -m “test: message”	Create a new commit in a Git repository with a specific message to indicate test-related changes.
git commit -m “perf: message”	Create a new commit in a Git repository with a specific message to indicate performance-related changes.

Commands	**Description**
git commit -m “ci: message”	Create a new commit in a Git repository with a specific message to indicate the continuous integration (CI) system-related changes.
git commit -m “build: message”	Create a new commit in a Git repository with a specific message to indicate the changes related to the build process.
git commit -m “revert: message”	Create a new commit in a Git repository with a specific message to indicate the changes related to revert a previous commit.

Branching and Merging Commands

Commands	Description
git branch	Lists all branches in the repository.
git branch <branch-name>	Creates a new branch with the specified name.
git branch -d <branch-name>	Deletes the specified branch.
git branch -a	Lists all local and remote branches.
git branch -r	Lists all remote branches.

Commands	Description
git checkout <branch-name>	Switches to the specified branch.
git checkout -b <new-branch-name>	Creates a new branch and switches to it.
git checkout — <file>	Discards changes made to the specified file and revert it to the version in the last commit.
git merge <branch>	Merges the specified branch into the current branch.
git log	Displays the commit history of the current branch.
git log <branch-d	Displays the commit history of the specified branch.
git log –follow <file>	Displays the commit history of a file, including its renames.
git log –all	Displays the commit history of all branches.
git stash	Stashes the changes in the working directory, allowing you to switch to a different branch or commit without committing the changes.
git stash list	Lists all stashes in the repository.

Commands	Description
git stash pop	Applies and removes the most recent stash from the stash list.
git stash drop	Removes the most recent stash from the stash list.
git tag	Lists all tags in the repository.
git tag <tag-name>	Creates a lightweight tag at the current commit.
git tag <tag-name> <commit>	Creates a lightweight tag at the specified commit.
git tag -a <tag-name> -m “<message>”	Creates an annotated tag at the current commit with a custom message.

Remote Repositories commands

Commands	Description
git fetch	Retrieves change from a remote repository, including new branches and commit.
git fetch <remote>	Retrieves change from the specified remote repository.

Commands	Description
git fetch –prune	Removes any remote-tracking branches that no longer exist on the remote repository.
git pull	Fetches changes from the remote repository and merges them into the current branch.
git pull <remote>	Fetches changes from the specified remote repository and merges them into the current branch.
git pull –rebase	Fetches changes from the remote repository and rebases the current branch onto the updated branch.
git push	Pushes local commits to the remote repository.
git push <remote>	Pushes local commits to the specified remote repository.
git push <remote> <branch>	Pushes local commits to the specified branch of the remote repository.
git push –all	Pushes all branches to the remote repository.
git remote	Lists all remote repositories.
git remote add <name> <url>	Adds a new remote repository with the specified name and URL.

Git Comparison commands

Commands	Description
git show	Shows the details of a specific commit, including its changes.
git show <commit>	Shows the details of the specified commit, including its changes.

Git Managing History commands

Commands	Description
git revert <commit>	Creates a new commit that undoes the changes introduced by the specified commit.
git revert –no-commit <commit>	Undoes the changes introduced by the specified commit, but does not create a new commit.
git rebase <branch>	Reapplies commits on the current branch onto the tip of the specified branch.

Why we use Git?

- Track changes to your code
- Collaborate on projects with others
- Maintain an organized code history
- Easily revert to previous versions when needed
- Release your code efficiently and manage versions
- Enhance productivity and code integrity in software development.

List of Top Useful Git Commands

Git Commands list that can be used frequently on Git.

1. git help

Take help from the Git help section for different commands and other errors.

```
git help
```

2. git config

To set the basic configurations on Git like your name and email.

```
git config
```

3. git config –-global user.name " "

Sets configuration values for your user name on git.

```
git config –-global user.name "Sachin tailor"
```

4. git config –-global user.email " "

Sets configuration values for your user email on git.

```
git config –-global user.email saachin.tailor@gmail.com
```

5. git config –-global color.ui

To see different colors on the command line for different outputs.

```
git config –-global color.ui true
```

6. mkdir

Create a directory if not created initially.

```
mkdir store
```

7. cd

To go inside the directory and work on its contents.

```
cd store
```

8. git init

To create a local git repository for us in our store folder. This will help to manage the git commands for that particular repository.

```
git init
```

9. git status

To see what's changed since the last commit. It shows all the files that have been added and modified and are ready to be committed and files that are untracked.

```
git status
```

10. git add Readme.txt

To add a file Readme.txt to the staging area to track its changes.

```
git add Readme.txt
```

11. git commit -m " "

To commit our changes(taking a snapshot) and provide a message to remember for future reference.

```
git commit -m "Created a Readme.txt"
```

12. git log

To check the history of commits for our reference.

```
git log
```

13. git add

To add a specific list of files to the staging area.

```
git add
```

14. git add –all

To add all files of the current directory to the staging area.

```
git add --all
```

15. git add *.txt

To add all text files of the current directory to the staging area.

```
git add *.txt
```

16. git add docs/*.txt

To add all text files of a particular directory(docs) to the staging area.

```
git add docs/*.txt
```

17. git add docs/

To add all files in a particular directory(docs) to the staging area.

```
git add docs/
```

18. git add “*.txt”

To add text files of the entire project to the staging area.

```
git add “*.txt”
```

19. git diff

To figure out what changes you made since the last commit.

```
git diff
```

20. git reset head license

To undo the staging of the file that was added in the staging area.

```
git reset head license
```

21. git checkout –license

To Blow away all changes since the last commit of the file.

```
git checkout –license
```

22. git commit -a -m “ ”

To add any of our tracked files to the staging area and commit them by providing a message to remember.

```
git commit -a -m “Readme.md”
```

23. git reset –soft HEAD^

To undo the last commit and bring the file to the staging area.

```
git reset –soft HEAD^
```

24. git reset –hard HEAD^

To undo the last commit and remove the file from the staging area as well(In case we went horribly wrong).

```
git reset –hard HEAD^
```

25. git reset –hard HEAD^^

To undo the last 2 commits and all changes.

```
git reset –hard HEAD^^
```

26. git remote add origin

These commands make a bookmark which signifies that this particular remote refers to this URL. This remote will be used to pull any content from the directory and push our local content to the global server.

```
git remote add origin
https://github.com/madaan123/MyAlgorithms.git
```

27. git remote add <address>

To add new remotes to our local repository for a particular git address.

```
git remote add <address>
```

28. git remove rm

To remove a remote from our local repository.

```
git remove rm
```

29. git push -u origin master

To push all the contents of our local repository that belong to the master branch to the server(Global repository).

```
git push -u origin master
```

```

```

31. git branch Testing

To create a new branch named Testing.

```
git branch Testing
```

32. git branch

To see all the branches present and current branches that we are working on.

```
git branch
```

33. git checkout Testing

To switch to branch Testing from the master branch.

```
git checkout Testing
```

34. ls

To see directories and files in the current directory.

```
ls
```

35. ls -la

To see hidden directories and files within the current directory.

```
ls -la
```

36. git merge Testing

To merge the Testing branch with the master branch.

```
git merge Testing
```

37. git branch -d Testing
To delete the Testing branch.

```
git branch -d Testing
```

38. git checkout -b admin
To create a new branch admin and set it as the current branch.

```
git checkout -b admin
```

39. git branch -r
To look at all the remote branches.

```
git branch -r
```

40. git branch -D Testing
To forcefully delete a branch without making commits.

```
git branch -D Testing
```

41. git tag
To see the list of available tags.

```
git tag
```

42. git checkout v0.0.1
To set the current tag to v0.0.1.

```
git checkout v0.0.1
```

43. git tag -a v0.0.3 -m “version 0.0.3”
To create a new tag.

```
git tag -a v0.0.3 -m “version 0.0.3”
```

44. git push –tags
To push the tags to the remote repository.

```
git push –tags
```

45. git fetch

To fetch down any changes from the global repository to the current repository.

```
git fetch
```

46. git stash

To move staged files to the stash area which is present in the staging area.

```
git stash
```

47. git stash pop

To get back the files that are present in the stash area.

```
git stash pop
```

48. git stash clear

To clear the stash folder.

```
git stash clear
```

49. git rebase

Three tasks are performed by git rebase

1. Move all changes to master which are not in origin/master to a temporary area.
2. Run all origin master commits.
3. Run all commits in the temporary area on top of our master one at a time, so it avoids merge commits.

```
git rebase
```

50. git –version

used to show the current version of Git

```
git –version
```