TheAutomationEngineer

What is Git :-

- Git is a version control system.
- Git helps you keep track of code changes.
- Git is used to collaborate on code.
- Git and GitHub are different things.

Why Git :-

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

Features of Git :-

- When a file is changed, added or deleted, it is considered modified
- You select the modified files you want to Stage
- The Staged files are Committed, which prompts Git to store a permanent snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

What is GitHub :-

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

By Achal Singh

TheAutomationEngineer

Git-GitHub Cheatsheet

Basic Setup

- **git config --globaluser.name "Your Name"**

Set your Git username.

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

Set your Git email.

- **git config --list**

List all Git configurations.

Initializing and Cloning

- **gitinit**

Initialize a new Git repository in your project.

- **git clone <repo-url>**

Clone an existing repository.

Working with Changes

- **git add <file>**

Stage a specific file for commit.

- **git add .**

Stage all changes in the current directory.

- **git commit -m "Commit message"**

Commit changes with a message.

- **git commit -am "Message"**

Add and commit tracked files in one step.

- **git commit --amend**

Edit the last commit message or add changes to it.

Handling Merge Conflicts

- **git diff**

Compare working directory changes.

- **git diff <branch1> <branch2>**

Compare two branches.

Resolve conflicts: Open the files, fix conflicts, then add and commit.

Status & Logs

- **git status**

Show the current status of changes in the working directory.

- **git log**

View commit history.

- **git log --oneline**

Show concise commit history.

Branching & Merging

- **git branch <branch-name>**

Create a new branch.

- **git checkout <branch-name>**

Switch to a specific branch.

- **git checkout -b <branch-name>**

Create and switch to a new branch.

- **git merge <branch-name>**

Merge specified branch into the current branch.

- **git rebase <branch-name>**

Reapply commits on top of another base.

- **git rebase -i HEAD~<n>**

Interactive rebase to edit commit history, rearrange commits, modify commit messages, or squash the last n commits

- **git branch -d <branch-name>**

Delete a local branch (use -D to force delete).

Spoorti Shetty

By Achal Singh

TheAutomationEngineer

Configuring git for the first time :-

```
$ git config --global user.name “<Enter your username here>”
```

```
$ git config --global user.email “<Enter your email here>”
```

General Git Features :-

Initializing Git :-

```
$ git init
```

Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder to keep track of changes.

Staging files/Adding files to Git repo :-

Staged files are files that are readyto becommitted tothe repository you are working on.

When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

```
$ git add <filename with extension>
```

Staging all files in a folder :-

```
$ git add --all
```

OR

```
$ git add -A
```

By Achal Singh

TheAutomationEngineer

Making a Commit :-

Adding commitskeep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should always include a message.

```
$ git commit -m “<Enter your message here>”
```

Git Commit without Stage :-

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment.

```
$ git commit -a -m “<Enter your message here>”
```

Status of files and log :-

```
$ git status
```

File status in a more compact way :-

```
$ git status --short
```

Log of a file :-

Log is used to view the history of commits for a repo.

```
$ git log
```

```
$ git log --oneline
```

By Achal Singh

TheAutomationEngineer

Git Help :-

Ifyouare having trouble remembering commands or options for commands, you can use Git help.

See all the available options for the specific command -

```
$ git <command> -help
```

See all possible commands -

```
$ git help --all
```

If you find yourself stuck in the list view, SHIFT + G to jump the end of the list, then q to exit the view.

Git Branching :-

InGit,a branch is anew/separate version of the main repository. Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project.

We can even switch between branches and work on different projects without them interfering with each other.

Making a new Git Branch :-

```
$ git branch <name of branch>
```

Checking all available Branches :-

```
$ git branch
```

By Achal Singh

TheAutomationEngineer

Switching to other Branches :-

```
$ git checkout <branch name>
```

Making a new branch and directly switching to it :-

```
$ git checkout -b <branch name>
```

Deleting a Branch :-

```
$ git branch -d <branch name>
```

Merging two Branches :-

It's preferred to change/switchtomaster branch before any branch needs to be merged with it.

```
$ git merge <branch name>
```

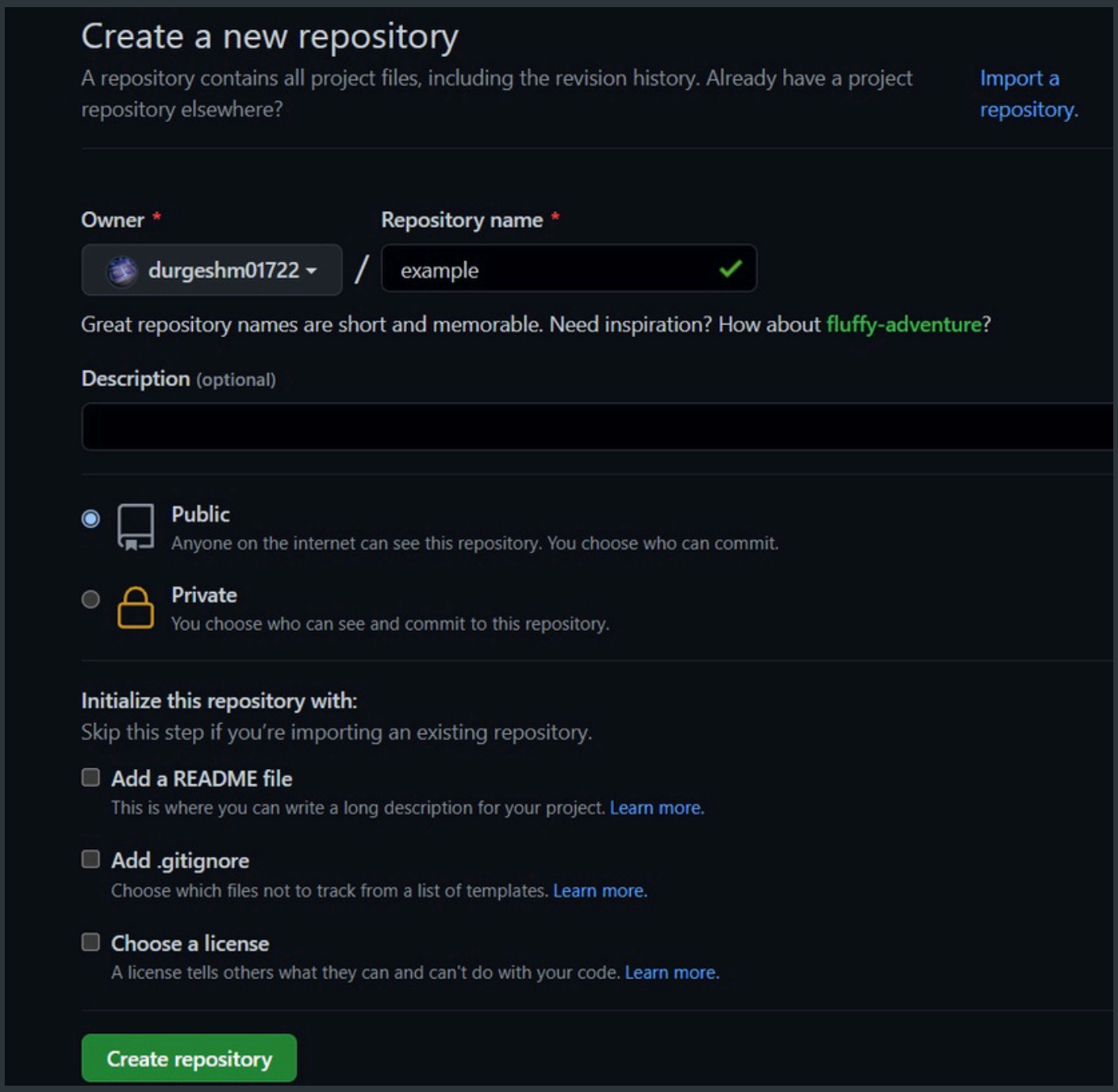
This will merge the specified branch with our master branch.

By Achal Singh

TheAutomationEngineer

Working with Github :-

Create a githubaccount to createyour remote repositories. Now, create a new repo where we will be uploading our files from local repo.



Note - Local repository (repo.) means the repo. which is on our system whereas, remote repo. means the one which is on other remote system/server, for eg. - GitHub, GitLab, Bitbucket, etc.

By Achal Singh

TheAutomationEngineer

Push local repo to GitHub :-

Copy theurl orthe link of therepothatwejust created. As an example, it should look like this - https://github.com/durgeshm01722/example.git

Paste the copied url in the below git command.

```
$ git remote add origin <paste copied URL here>
```

'git remote add origin <URL>' specifies that we are adding a remote repository, with the specified URL, as an origin to our local Git repo.

Finally, pushing our master branch to the origin URL (remote repo) and set it as the default remote branch.

```
$ git push --set-upstream origin master
```

Go back into GitHub and see that the repository has been updated.

Pushing local repo to github after doing the above process at least once :-

First commit all the changes. Then push all the changes to our remote origin i.e. remote repo on github.

```
$ git push origin
```

Pull local repo from GitHub :-

Git pullis used to pull all changesfroma remote repository into the branch we are working on. It is a combination of fetch and merge. Use it to update your local Git.

```
$ git pull origin
```

By Achal Singh

TheAutomationEngineer

Pull branch from GitHub :-

First, check which branches we haveandwhere are we working at the moment by ‘git branch’ command. Since we do not have the new branch on out local Git which is to be pulled from the Github. So, to see all local and remote branches, use -

```
$ git branch -a
```

For viewing only remote branches :-

```
$ git branch -r
```

Now, the new branch is seen in the console but it is not available on our local repo. So, let’s check it out using ‘git checkout <branch name>’. Now run ‘git pull’ to pull that branch on our local repo. We can now check the available branches using ‘git branch’.

Push branch to GitHub :-

First, let’s create a new local branch which we will be pushing to Github. Enter the command as ‘git checkout -b <branch name>’. You can check the status of the files in this current branch using ‘git status’. Commit all the uncommitted changes for all the files in this branch using ‘git commit -a -m “<Message>” ’. Now push this branch from our local repo to Github using ‘git push origin <branch name>’.

Git clone from GitHub :-

We can clone aforked repofrom Github on our local repo. A clone is a full copy of a repository, including all logging and versions of files. Move back to the original repository, and click the green "Code" button to get the URL to clone. Copy the URL.

By Achal Singh

TheAutomationEngineer

Now in the git bash, enter the following command to clone the copied repo onto your local machine -

```
$ git clone <copied URL>
```

To specify a specific folder to clone to, add the name of the folder after the repository URL, like this -

```
$ git clone <copied URL> <folder name>
```

By Achal Singh

TheAutomationEngineer

Git Undo :-

Git Revert :-

'revert' is the command we use when we want to take a previous commit and add it as a new commit, keeping the log intact. First thing, we need to find the point we want to return to. To do that, we need to go through the log. To avoid the very long log list, use the --oneline option which gives just one line per commit showing –

i. The first seven characters of the commit hash

ii. The commit message

Git Revert HEAD :-

We revert the latest commit using 'git revert HEAD' (revert the latest change, and then commit). By adding the option --no-edit, we can skip the commit message editor (getting the default revert message).

```
$ git revert HEAD --no-edit
```

Git Revert to any commit :-

To revert to earlier commits, use 'git revert HEAD~x' (x being a number. 1 going back one more, 2 going back two more, etc.)

Git Reset :-

'reset' is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit. First, get the seven characters of the commit hash from the log for the commit that you want to go back for. Then we reset our repository back to that specific commit using 'git reset commithash' (commithash being the first 7 characters of the commit hash we found in the log).

By Achal Singh

```
$ git reset <commithash>
```

Git Undo Reset :-

Even though thecommits are no longer showing up in the log, it is not removed from Git. If we know the commit hash, we can reset to it using 'git reset <commithash>'.

Git Amend :-

commit --amend is used to modify the most recent commit. It combines changes in the staging environment with the latest commit, and creates a new commit. This new commit replaces the latest commit entirely.

One of the simplest things you can do with --amend is to change a commit message.

```
$ git commit --amend -m "<Commit Message>"
```

Using this, the previous commit is replaced with our amended one.

Git Amend Files :-

Adding fileswith--amend works the same way as above. Just add them to the staging environment before committing.

TheAutomationEngineer

Undoing Changes

- **gitreset <file>**
Unstage a file.

- **git reset --soft HEAD~1**
Undo last commit but keep changes staged.

- **git reset --mixed HEAD~1**
Undo last commit, keep changes in the working directory (unstaged).

- **git reset --hard HEAD~1**
Completely remove the last commit.

- **git revert <commit-id>**
Create a new commit that undoes the specified commit.

Stashing Changes

- **gitstash**
Temporarily save changes.

- **git stash list**
View stashed changes.

- **git stash pop**
Reapply stashed changes and remove them from the stash list.

- **git stash apply**
Reapply stashed changes without removing them.

- **git stash clear**
Remove all stashed entries.

Collaborating & Pull Requests

- **git branch-a**
List all branches, including remote.

- **git push origin :<branch-name>**
Delete a remote branch.

Creating a Pull Request: Go to your GitHub repository, select your branch, and click "New Pull Request."

Remote Repositories

- **git remoteadd origin<url>**
Link your local repository to a remote one.

- **git remote -v**
List the remote repository URLs.

- **git remote set-url origin <new-url>**
Update the remote URL for the repository.

- **git remote rename <old-name> <new-name>**
Rename a remote.

- **git push -u origin <branch-name>**
Push changes to the remote repository.

- **git pull origin <branch-name>**
Pull changes from the remote branch.

- **git fetch**
Download updates from the remote without merging.

- **git fetch <remote>**
Fetch updates from a specific remote.

Advanced Operations

- **gitcherry-pick <commit-id>**
Apply a specific commit from another branch.

- **git cherry-pick <start-commit-id>^..<end-commit-id>**
Cherry-pick a range of commits.

- **git tag <tag-name>**
Add a tag to a commit.

- **git tag -d <tag-name>**
Remove a local tag.

- **git reflog**
View history of all changes (even uncommitted).

- **git reflog show <branch-name>**
Show reflog for a specific branch.

- **git show <commit-id>**
Show detailed info for a specific commit.

- **git bisect start**
Start bisecting to locate a bug.

Spoorti Shetty

By Achal Singh

TheAutomationEngineer

Reviewing Changes

- **git show <file>**
Display changes made to a specific file.

- **git diff <commit-id1> <commit-id2>**
Compare changes between two commits.

Help Command

- **git help <command>**
Get detailed help for a specific command.

GitHub Commands (Optional with GitHub CLI)

- **gh repo create**
Create a new GitHub repo from the command line.

- **gh repo clone <repo-url>**
Clone a GitHub repository.

- **gh pr create**
Create a pull request from the command line.

- **gh pr list**
List open pull requests in the repository.

- **gh issue create**
Create a GitHub issue from the command line.

GitHub API (using curl)

- **curl -H "Authorization: token YOUR_TOKEN" https://api.github.com/repos/USERNAME/REPO_NAME/issues**
List issues in a repository.

Submodules & Worktrees

- **git submodule add <repo-url> <path>**
Add a submodule.

- **git submodule init**
Initialize submodules.

- **git submodule update**
Update submodules.

- **git worktree add <path> <branch>**
Create a new working tree for a branch.

Cleaning Up

- **git clean -f**
Remove untracked files.

- **git clean -fd**
Remove untracked files and directories.

- **git gc --prune=now**
Clean up unnecessary files and optimize the local repository.

Repository Management and Information

- **git shortlog -s -n**
Summarize commits by author.

- **git describe --tags**
Get a readable name for a commit.

- **git blame <file>**
Show who last modified each line of a file.

- **git grep "search-term"**
Search for a term in the repository.

- **git revert <commit-id1>..<commit-id2>**
Revert a range of commits.

- **git archive --format=zip HEAD -o latest.zip**
Archive the latest commit as a ZIP file.

- **git fsck**
Check the object database for integrity.

Spoorti Shetty

By Achal Singh

TheAutomationEngineer

Best Practices and Common Workflows

- **Commit Often:** Make frequent commits with descriptive messages to maintain a clear project history.
- **Branch for Features:** Create a new branch for each feature or bug fix to keep changes organized and separate from the main codebase.
- **Use Meaningful Commit Messages:** Write clear and concise commit messages that explain the purpose of the changes.
- **Pull Regularly:** Regularly pull changes from the remote repository to stay updated with the latest changes and minimize merge conflicts.
- **Resolve Conflicts Promptly:** Address merge conflicts as soon as they arise to avoid complicating the integration process.
- **Review Pull Requests Thoroughly:** Ensure thorough review of pull requests to maintain code quality and facilitate knowledge sharing.
- **Tag Releases:** Use tags to mark important milestones or releases in the project for easy reference in the future.
- **Keep Your Branches Clean:** Delete branches that are no longer needed after merging them into the main branch to keep the repository organized.
- **Use Git Hooks for Automation:** Utilize Git hooks to automate tasks, like running tests before committing (pre-commit) or checking commit message formats. Hooks can help ensure code quality and consistency.
- **Squash Commits Before Merging:** Squash commits to combine related work into a single commit before merging, especially for feature branches. This keeps the project history clean and manageable.
- **Avoid Large Commits:** Try to keep commits small and focused on a single change or fix. This makes it easier to understand the history and isolate issues if something goes wrong.
- **Create Descriptive Branch Names:** Use branch naming conventions that describe the purpose, such as feature/login-form or fix/user-authentication-bug. This improves readability and collaboration.
- **Keep the Main Branch Deployable:** Always ensure that the main or production branch is stable and deployable. This allows the project to be released or updated at any time.

Spoorti Shetty

By Achal Singh