**Git CheatSheet for Developers**

**Git Configuration**

| **Command** | **Description** |
| --- | --- |
| **git config** | Check all configuration options |
| **git config --list** | Check all configuration options with name and email |
| **git clone [https://url]** | Clone source code from a remote repository |
| **git config --global user.name "Your name"** | Configure username |
| **git config --global user.email "Your email"** | Configure email |
| **git config --global core.editor vim** | Configure editor |
| **git config user.name** | List username |
| **git config user.email** | List email |

**Getting & Creating Projects**

| **Command** | **Description** |
| --- | --- |
| **git init** | Initialize a local Git repository |

**Basic Commands**

| **Command** | **Description** |
| --- | --- |
| **git status** | Check status |
| **git diff** | Compare and show updated codes |
| **git add [file name]** | Add file changes in next commit |
| **git add .** | Add all unstaged changes in next commit |
| **git reset [file name]** | Used to unstage the staged files |
| **git clean -f** | To delete or remove unstaged files forcefully |
| **git commit -m "message about updates"** | Commit changes to current branch |
| **git commit -amend** | Amend with last commit but use the previous commit log message |
| **git rm --cached [file]** | Removes the file from the staging area (Unstage) |
| **git rm [file]** | Deletes the file from your working directory and stages the deletion |
| **git pull** | Fetches and merges changes on the remote server to your working directory |
| **git pull --allow-unrelated-histories** | Pull changes form remote branch with unrelated histories |
| **git fetch** | Gathers remote commits but does not merge them unlike pull |
| **git remote add origin [url]** | Adding a remote repository |
| **git show** | Shows information about any git object |
| **gitk** | Shows graphical interface for a local repository |

**Branching & Merging**

| **Command** | **Description** |
| --- | --- |
| **git branch** | List branches |
| **git branch [branch-name]** | Create a local branch |
| **git branch -m [old branch name] [new branch name]** | Rename a local branch |
| **git branch -d [branch-name]** | Delete a branch |
| **git branch -D [branch name]** | Delete a branch forcefully |
| **git branch -a** | See all branches (local and remote) |
| **git checkout [branch-name]** | Switch to another branch |
| **git checkout -b "branch name"** | Create a new branch and switch to that branch |
| **git checkout -b [branch name] origin/[branch name]** | Clone a remote branch and switch to it |
| **git switch [branch-name]** | Switch to another branch |
| **git merge [branch-name]** | Merge branchs |
| **git merge [source branch] [target branch]** | Merge a branch into a target branch |
| **git merge --allow-unrelated-histories** | Merge unrelated histories |
| **git cherry-pick [commit-ID]** | Bring in changes from one (or more) particular commit to the current branch. |

**Discard Changes**

| **Command** | **Description** |
| --- | --- |
| **git checkout -** | Switch to the branch last checked out |
| **git checkout -- [file-name.txt]** | Discard changes to a file |
| **git checkout [file]** | Matches the file with last commit |
| **git restore .** | To restore all files in the current directory |
| **git revert [commit-ID]** | Create new commit, reverting the changes from a specified commit in remote branch |
| **git help -a** | Shows the list of all available Git commands |

**Set Upstream Branch**

| **Command** | **Description** |
| --- | --- |
| **git push --set-upstream origin current-branch-name or git push -u origin current-branch-name** | - To push the current branch and set the remote as upstream - git push --set-upstream origin development - git push -u origin development |
| **git pull origin main or git pull origin development** | - Update the ***current branch*** from main branch - Update the ***current branch*** from development branch - Before updating, ***push*** you all modification to ***remote brach*** - Otherwise every ***modification will be removed*** |

**Sharing & Updating Projects**

| **Command** | **Description** |
| --- | --- |
| **git push origin** [branch name] | Push a branch to your remote repository |
| **git pull origin** [branch name] | Pull a branch from your remote repository |
| **git remote origin** [branch name] | Connect repository to local server |
| **git push <remote>** -**-force** | Forces the push even if it results in a non-fast-forward merge. Be sure that nobody has pulled the commits before using the --force option. |

**Inspection & Comparison**

| **Command** | **Description** |
| --- | --- |
| **git log** | View changes |
| **git log --summary** | View changes (detailed) |
| **git log --follow [file]** | Show the commits that changed file, even across renames |
| **git log --oneline** | View changes (briefly) |
| **git log branchB..branchA** | View changes on branchA that are not on branchB |
| **git log --graph** | Enables you to view your git log as a graph |
| **git log --decorate** | Makes git log display all of the references (e.g., branches, tags, etc) that point to each commit |
| **git log --author="name\_of\_author"** | Search for specific author |
| **git shortlog** | It groups each commit by author and displays the first line of each commit message |
| **git reflog** | Git keeps track of updates to the tip of branches using a mechanism called reflog. This allows you to go back to changesets even though they are not referenced by any branch or tag |
| **git diff branchB...branchA** | View the differences of what is in branchA that are not on branchB |
| **git blame [file name]** | Display the modification on each line of a file |
| **git diff --name-only** | Show only names of changed files |
| **git bisect start** | Starts the bisection search process to find that bad commit which introduced the bug we're facing right now |
| **git bisect good git bisect good [Commit ID]** | Takes up the good commit, which is that one where the bug was not there |
| **git bisect bad [Commit ID]** | Takes up the bad commit, which is that one where the bug was there. If commit ID is not provided, then it takes up the current commit as the bad commit |
| **git grep "hello"** | A text search on all files in the directory |
| **git bugreport** | Created new report at git-bugreport-2022-10-25-1228.txt |

**Tracking Path Changes**

| **Command** | **Description** |
| --- | --- |
| **git rm** [file] | Delete the file from project and stage the removal for commit |
| **git mv** [existing-path] [new-path] | Change an existing file path and stage the move |
| **git log --stat -M** | Show all commit logs with indication of any paths that moved |

**Setting up Alias**

| **Command** | **Description** |
| --- | --- |
| **git config --global alias.[short name for command] [actual command]** | Alias make the commands short and handy |

**Rewrite History**

| **Command** | **Description** |
| --- | --- |
| git rebase [branch] | Apply any commits of current branch ahead of specified one |
| git reset --hard [commit] | Clear staging area, rewrite working tree from specified commit |
| git reset HEAD "file-name" | Go back to the pointer I'm already at, remove from staged |
| git reset --soft "commit-hash" | Take my changes and go back/kill the commit I made but the file will be staged with the change ready to be committed again |
| git reset --mixed "commit-hash" | Return/Kill the commit too but it returns the files to before staged i.e. modified but it will still return |
| git reset --hard "commit-hash" | It will simply ignore the existence of this commit and undo everything that was done in this commit. It is a very rough reset and is normally used before pushing your commit to the remote repository. |

**Deletion**

| **Command** | **Description** |
| --- | --- |
| **git gc** | Cleans unnecessary files and optimizes the local repository |
| **git prune** | Deletes objects that don’t have any incoming pointers |

**Temporary Commits**

| **command** | **Description** |
| --- | --- |
| **git stash** | Save modified and staged changes |
| **git stash list** | list stack-order of stashed file changes |
| **git stash pop** | write working from top of stash stack |
| **git stash drop** | discard the changes from top of stash stack |