Git Commands Cheat Sheet

Introduction:
This cheat sheet provides a quick reference for essential Git commands and practices. Whether you're a beginner or an experienced user, this guide will help you navigate Git efficiently.

1. General Commands:
For checking status, viewing commit history
- `git status`: Check the status of changes
- `git log`: View commit history

2. Starting/Managing Repos:
For cloning and initializing a new Git repository
- `git clone [url]`: Clone a repository
- `git init`: Initialize a new Git repository
- `git remote add origin [url]`: Set remote origin URL

3. Staging & Committing Changes:
For staging and committing changes to Git
- `git add [file/directory]`: Stage supplied file/directory
- `git add .`: Stage all the changes for the commit
- `git commit -m "message"`: Commit staged changes with a message

4. Branching & Navigation:
Managing branches, switching between them, and creating new branches
- `git branch`: List branches in the repo
- `git checkout [branch_name]`: Switch to a branch or,
- `git switch [branch_name]`: Switch to a branch
- `git checkout -b [new_branch_name]`: Create and switch to a new branch or,
- `git switch -b [new_branch_name]`: Create and switch to a new branch
- `git branch -D [branch_name]`: To delete a branch

5. Pushing & Pulling Changes:
Pushing changes to a branch and pulling changes from a branch
- `git push origin [branch_name]`: Push changes to a branch
- `git pull origin [branch_name]`: Pull changes from a branch

6. Stashing Changes:

Storing and applying changes temporarily

- `git stash`: Temporarily store changes you don't want to commit yet
- `git stash –include-untracked`: Temporarily store changes you don't want to commit yet including the untracked files that are not committed yet
- `git stash pop`: Apply stored changes back to the working tree

  Notes:
  If you've made changes on the main branch (or any other branch) and want to switch branches without committing those changes, use ‘**git stash**’ to store the changes temporarily. Once you've switched to the desired branch, use **`git stash pop`** to retrieve and apply those changes.

7. Cleaning & File Removal:
Removing files and directories, and cleaning untracked files

- `git mv [source file path] [destination file path]`: Move or rename a file within a git repository
- `git clean . -f`: Remove untracked files

8. Undo/Revert Actions:
Reverting changes in the working directory and resetting commits

- `git checkout .`: Revert changes in the working directory
- `git reset HEAD~[number]`: Reset the last [number] of commits and put the commit changes in unstaged area
- `git reset –hard HEAD~[number]`: This would remove your uncommitted changes, even if you staged them
- `Git reset -soft HEAD~[number]`: Use this if you don't want your unstaged changes to be removed

  Error Handling:
  Handling common errors encountered during Git operations. If you encounter the error: "*Your local changes to the following files would be overwritten by checkout: Please commit your changes or stash them before you switch branches,*" use `git checkout .` to revert the changes.

10. Merging & Pull Requests:
Creating pull requests and merging changes from other branches. (On hosting platforms like GitHub/Bitbucket) Create a pull request to propose changes.

- `git merge [branch_name]`: Merge changes from another branch into the current one

11. Advanced/Utility Commands:
Utilizing advanced Git commands for remote management and branch deletion

- `git reflog`: Show the history of recent actions in the repo
- `git rm -r –cached .`: Remove all the files from git without deleting them

Abbreviations and Acronyms:

Here are some common abbreviations and acronyms that you can use for the Git Commands Cheat Sheet:

- URL: Uniform Resource Locator
- Git: Global Information Tracker
- Repo: Repository
- Cmd: Command
- Msg: Message
- Diff: Difference
- Stg: Staging
- WD: Working Directory
- PR: Pull Request
- Ref: Reference
- Log: Log
- Branch: Branch
- Init: Initialize
- Push: Push
- Pull: Pull
- Merge: Merge
- HEAD: Current Commit
- Stash: Stash

Tips and Tricks:

Here are some additional tips and tricks to enhance your Git experience:

Ignoring Files:
Create a ***.gitignore*** file to specify files or directories you want Git to ignore. This is useful for excluding files like build artifacts or sensitive information.

.gitignore example:

```
*.log
build/
Secret.txt
```

Viewing Diffs:
Use **git diff** to see the differences between the working directory, staging area, and the last commit

- `git diff`: Show differences in the working directory
- `git diff --staged`: Show differences in the staging area
- `git diff HEAD`: Show differences between the working directory and the last commit

Viewing the Remote Information:
Check information about remote repositories with git remote -v and fetch updates from the remote repository using git fetch

- `git remote -v`: View remote repositories
- `git fetch`: Fetch changes from the remote repository

Interactive Rebase:

Use interactive rebase (git rebase -i) to modify, combine, or delete commits before pushing them. This can be helpful to create cleaner commit histories.

- `git rebase -i HEAD~[number]`: Rebase the last [number] commits interactively

Blame:
Use git blame to see who last modified each line of a file. This can help track down when and by whom specific changes were made.

- `git blame [filename]`: Show file changes and author details

Cleaning & File Removal:
Removing files and directories, and cleaning untracked files

- `rm [file_name]`: Remove a file
- `rm -rf [directory_name]`: Remove a directory and its contents
- `pwd`: print working directory

References:

- https://git-scm.com/

By **Priyanka Madan Lal**
Sr. QA Engineer