Useful GIT Commands

COMMON COMMANDS:

Command Usage Example	Description
pwd	To get the current working directory
git push origin HEAD:refs/for/develop	Push the changes from the current branch to the branch mentioned after HEAD:refs/for/
HEAD	always refers to the last commit on the current branch
origin	In Git, "origin" is a shorthand name for the remote repository that a project was originally cloned from. More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier
git reset --soft HEAD~1	Soft resets the branch to the previous commit with an option to switch back to this commit at a later time
git reset --hard origin/<branch name>	Hard resets the branch to the specified branch name where in there is no option of returning to this commit
git status	returns the status of this branch
git checkout develop	to switch to the branch name mentioned here
git pull	to make the local repository up to date with the changes from the remote repository
git add <file_name>	to add the file name to the staging area. Please note that if the file is under subfolder then we need to specify the full path from the base path of the branch. For eg: /tests/python/data/abc.inc
git add .	to add all the files that are modified to the staging area
git commit -m "commit message"	to commit the changes to the staging area with an appropriate message. Please note that the firs word of the commit message should always be the TASK number
git commit --amend <escape> :w - save :q - exit	to amend the commit message
git commit -am "commit message"	to perform an express commit

CONFIG COMMANDS:

Command Usage Example	Description
git config --global merge.tool p4merge	to configure the default merge tool as p4merge

git config --global mergetool.p4merge.path "c:/Program Files/Perforce/p4merge.exe"	to provide the application path of the merge tool
git config --global mergetool.prompt false	to avoid the prompt to open the tool
git config --global diff.tool p4merge	to configure the default diff tool as p4merge
git config --global difftool.p4merge.path "c:/Program Files/Perforce/p4merge.exe"	to provide the application path of the diff tool
git config --global difftool.prompt false	to avoid the prompt to open the tool
git config --global --list	to list all the global configurations of git
git config --global -e	This opens the .gitconfig file
git config --global user.name “firstname lastname”	to add the user name to the list of global configurations
git config --global user.email firstname.lastname@fiserv.com	to add the user email address to the list of global configurations
git commit --amend --reset-author	to reset the author settings after changing the user name and email addresses

LOG COMMANDS:

Command Usage Example	Description
git log --oneline	to get the history of commits
git log --oneline --graph --decorate --all	to get the history of commits along with the mergers and feature branches

DIFF COMMANDS:

Command Usage Example	Description
git diff	To get the difference of contents of file between working directory and staging area, we can use the following command:
git difftool	To visually see in the difftool
git diff HEAD	difference between working directory and the last commit
git diff --staged HEAD	difference between staging area and the last commit
git diff -- <filename>	difference between two versions of a single file if there are multiple file differences between the versions
git diff <reference1 or git commit id> <reference2 or git commit id or HEAD>	to compare 2 different commits
git diff HEAD HEAD^	HEAD^ means HEAD-1. To compare the last and last but one commits
git diff <tagname1> <tagname2>	to see the difference between 2 tags
git difftool <tagname1> <tagname2>	to visually see the difference between 2 tags

BRANCH COMMANDS:

Command Usage Example	Description
git branch	to list all the local branches
git branch -a	to list all the branches including remote branches
git branch -m mynewbranch newbranch	to rename the branch from "mynewbranch" to "newbranch"
git branch -d <branchname>	to delete a branch. We need to be on a different branch before deleting the given branch
git checkout -b <branchname>	to create a branch and then checkout the branch

MERGE COMMANDS:

Command Usage Example	Description
git merge <branchname>	merges the mentioned branch name to the current branch. Fast forward merge can happen only when there are no commits to the branch to which the other branch merges
git merge --no-ff branch_to_be_merged	To disable fast forward merge
git merge <branchname> -m "commit message"	to provide the merge commit message

REBASE COMMANDS:

What is Rebase? - Rebase option is used to push the changes from the develop or master branches to the feature branch so that it will not cause any merge conflict when merging the feature branch with either the master or develop branches

Command Usage Example	Description
git rebase master	we need to be in the current feature branch to rebase from the master or develop branches
git rebase --abort	to abort the rebase
git rebase --continue	to continue the rebase after resolving the rebase conflicts

STASH COMMANDS:

Command Usage Example	Description
git stash	used to save the changes made to the files. GIT stash will work only on tracked files
git stash apply	to get the stashed files back to adding state so that we can add and commit to the staging area
git stash list	to get the list of stashes that we have saved
git stash drop	to drop the last stash we saved
git stash -u	to add any untracked files to the stash.
git stash pop	to apply the old stash and then drop the stash so that we can create a new one including the newly made changes

git stash save "stash text"	to save multiple stashed with meaningful pointer message
git stash show stash@{1}	to get the last but one stash based on the index number 1 that we have passed. GIT Stash starts with index 0 and the latest stash will be stored in index 0 where as the oldest stash will be saved in last index number. LIFO stack
git stash apply stash@{1}	to get the stash available in the index number 1
git stash drop stash@{1}	to drop the stash at index number 1 and the stash reorders accordingly
git stash clear	to clear the list of stash saved already
git stash branch <branch_name>	to save the stash in a new branch. Git then switched to the new branch. It creates a stash in the current branch, creates a new branch, moves the changes to the new branch, drops the stash from the old branch and then switches to the new branch

LIGHT WEIGHT TAG COMMANDS:

Command Usage Example	Description
git tag <tagname>	to give a user defined name to a particular branch
git tag --list	to list all the list of tags that is available
git show <tagname>	to show the details of the tag
git tag --delete <tagname>	to delete the tag

ANNOTATED TAG COMMANDS:

Command Usage Example	Description
git tag -a <tagname>	to create an annotated tag. After pressing enter on this command, it will prompt to provide a message that is meaningful to denote the tag. This is the difference between an annotated tag and a light weighted tag
git tag <tagname> -m "<message to be written>"	to add message to the tag
git tag -a <tagname> commit_id	to add tags to a historical commit
git tag -a <tagname> -f commit_id	to update tags to a point to a different historical commit
git push origin <tagname>	to push a specific tag
git push origin master --tags	to push all the tags at once
git push origin :<tagname>	to delete the tags that are already pushed

RESET & REFLOG COMMANDS:

3 types of resets -> soft, hard & mixed(default)

Command Usage Example	Description
git reset HEAD^1	to reset the head to the latest branch minus one
git reset --soft HEAD~1	Soft resets the branch to the previous commit with an option to switch back to this commit at a later time
git reset --hard origin/<branch name>	Hard resets the branch to the specified branch name where in there is no option of returning to this commit
git reflog	displays the list of commits & resets made to the current branch with their commit/reset reference numbers. it has all the commit changes for the last 60 days
git reset <git ref num>	to move back and forth in the list of commits/resets that are displayed in the reflog
git cherry-pick <commit reference number>	to cherry pick the changes made under a specific commit reference number