**Section 4**

**Diving Deeper into Commits**

Folder Name: git-deep-dive

1. **Understanding git stash**

Temp storage for unstaged and uncommitted changes

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git stash | * Stashes changes that have been added to index (staged changes) and changes made to files currently tracked by Git (unstaged changes). |  |
| git stash -u | * Stash untracked files |  |
| git stash list | * Listing multiple stashes. * Each stash entry has a name – stash@{1}, name of the branch when the entry was made, short description of the commit entry |  |
| git stash push –m “<msg>” |  |  |
| git stash pop | * Reapply previously stashed changes. * Last stash committed to be added to the project and removed from the stash list |  |
| git stash apply (#) | * Latest commit |  |
| git stash drop (#) | * Latest stashed data deleted |  |
| git stash clear | * Stash listed deleted |  |

1. **Git Reflog**

All project changes made

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git reflog | Overview of all changes applied in a branch listed  30 days storage   * Copy cache/commit code of the one you want to revert back to   Bring last commit into project |  |

1. **Combining Master and Feature Branches**

**Master**

Main project branch with latest deployment

**Feature branch**

Holds all previous commits

Based on master branch

* **Merge Types**

Folder Name:Branches

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git merge | * Combining commits from different branches by creating a new merge commit (recursive) or by moving the HEAD (fast-forward) |  |
| git merge (name of branch)  FFWD | * No additional commit in Master branch * Merge moves HEAD forward (i.e HEAD moved to latest feature branch) * New commit not created |  |
| git merge --squash (name of feature branch) | Squash all commits in feature branch into latest commit |  |
| **Non FFWD - Recursive** |  |  |
| git merge --no--ff (name of feature branch) | * Additional commits in both master and feature branch after feature was created * Additional merge created in master branch * Only undo latest merge commit to bring master to HEAD in one swoop |  |
| git rebase (name of branch you want to rebase) | * Add commits in feature branch to updated master branch * New base commit for commits in feature branch (rebase master to feature branch) * Avoid outside repo * Creates new commits (new ID created), doesn’t move commits * Bases latest changes in feature branch on changes in master branch * Know when you rebase – **not advisable** |  |

1. **Merge Conflicts**

Always compare changes prior to merging

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git status | Check conflicts |  |
| git merge --abort | Abort merge – back to state before applying git commit command |  |
| git log --merge | IDs two commits you want to merge |  |
| git diff | IDs problems within branches |  |
| git commit |  |  |

1. **Cherry Picking**

Copy commit including the changes made only in this commit as HEAD to other branch

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git cherry-pick (commit ID) | * Add specific commit to branch (HEAD) * Copies commit with new ID * New commit ID created |  |

1. **Git Tags**

Different milestones in project

* Tags are made to tag important stages in project history
* Lightweight tag – latest commit (HEAD commit)
* Annotated tag - Full object (contact details). Typically work with these kind of tags
* Keeps tags structured

|  |  |  |
| --- | --- | --- |
| **Command** | **Explanation** | **Image** |
| git tag 1.0 (input commit ID) | Lightweight tag (temp) |  |
| git show 1.0 | Display content/objects  Commits = Objects |  |
| git tag | Tags listed |  |
| git tag –d 1.0 | Deleted tag |  |
| git tag –a 2.0 –m “<msg>” | Annotated tag |  |
|  |  |  |