**GIT**

# GIT Intro

1. Why Source Control?
   1. Backup/Achieve
   2. Versioning/History
   3. Undo Changes
   4. Comparing
   5. Collaboration/Teamwork
   6. Isolation of changes
2. Source Control Options:
   1. Centralized
      1. Free: Subversion, CVS
      2. Commercial: ClearCase, Perforce, TFS
      3. Requires connection to central server for most operations
   2. Decentralized/Distributed
      1. Mercurial
      2. Git
      3. Most Operations are local
      4. Central server not required
3. Why Git?
   1. Distributed Source Control System
      1. Not Required to be decentralized
   2. Massively Scales
   3. Open Source
   4. Developed for Linux Project Requirements
   5. Most operations are local
   6. Very fast
   7. Active Community
4. Key GIT Terminology
   1. Repository contains files, history, config managed by Git
   2. Three States of Git
      1. Working Directory
      2. Staging Area - Index
      3. Commit – Git Local Repo

There is hidden folder called /. git, that contains actual git repo, manages the git commit history.

* 1. Remote Repository
  2. Master Branch – Default Branch

# GIT Quick Start

1. Create Account in GitHub
2. Create New Repository - **github-demo**
3. Open CMD; check *git –version*
4. Open Git bash
   * + - 1. pwd
         2. mkdir git-learning-projects
         3. cd git-learning-projects
5. Set up
   * + - 1. git config --global user.name “Gorla Kumar”
         2. git config --global user.email “Email”
         3. git config --global –list
         4. git clone <https://github.com/gorlakumarr/github_demo.git>
         5. ls
         6. cd github\_demo/
         7. git status
         8. echo "Test Git Quick Start" >> start.txt
         9. cat start.txt
         10. git add start.txt
         11. git commit -m "Adding start.txt file"
         12. git push origin master

# Text Editor Installation

1. Installation overview:
   1. Notepad++
      * + 1. Add Notepad C:\Program Files\Notepad++ to System Variables Path. They try **notepad++** on cmd the notepad++ will open.
   2. Bash alias
      * + 1. notepad++ .bash\_profile
          2. type: alias npp= ‘notepad++.exe -multiInst -nosession'
          3. npp 🡪 Notepad++ will open
   3. Git Configuration
      * + 1. cat ~/.gitconfig
          2. git config --global core.editor “notepad++.exe -multiInst -nosession”
          3. git config --global -e

# Basic GIT Commands

1. Git Basics Overview:
   1. Starting a Project

[Hipster Ipsum – Artisanal filler text for your project. (hipsum.co)](https://hipsum.co/)

* + 1. Fresh (No Source yet)

git init fresh-project

cd fresh-project

ls -al

add file

git commit

master (root-commit) 905d8c8] Adding hipster.txt file This was done by notepad++

rm -rf fresh-project

* + 1. Existing Source Locally: [Initializr - Start an HTML5 Boilerplate project in 15 seconds!](http://www.initializr.com/)
       - 1. Download project
         2. Navigate to user dir: unzip ~/Downloads/initializr-verekia-4.0.zip
         3. mv initializer web-project
         4. cd web-project
         5. git init
         6. git status
         7. git add .
         8. git commit -m “First Commit, inline”
         9. rm -rf .git
         10. cd ..
         11. rm -rf web-project
    2. GitHub Project (Fork and Clone) : [scm-ninja/starter-web: Simple starting point website project based upon Initializr (github.com)](https://github.com/scm-ninja/starter-web)

1. Basic workflow (add, commit, push & pull)
   * + - 1. npp hipster.txt
         2. git add hipster.txt
         3. git status
         4. git pull origin master **(fetch + merge)**
         5. git push origin master
2. Tracked Files: *A file is any file that Git is aware of and tracking actively.*
   * + - 1. Modify hipster.txt
         2. git commit -am “Adding more hispsum text” **(Add + Commit)**
         3. git ls-files (**git tracked files)**
         4. npp file.txt (will not be shown in tracked files)
         5. git add file.txt (now it will be shown under tracked files by GIT)
3. Editing files
   * + - 1. Even with in the same file, git can track the difference between changes that are to be committed that are staged, and changes that have not been staged yet.
4. Recursive Add:
   * + - 1. mkdir -p level1/level2/level3
         2. add file
         3. git add .
5. Renaming and Moving files
   * + - 1. git mv level3-file.txt level3.txt
         2. git add -A 🡪 recursively add any changes, but it will also update any files that have been renamed, moved or deleted.
         3. git branch -m <new\_name>
         4. git push origin -u <new\_name>
         5. git push origin --delete <old\_name>
6. Deleting files
   * + - 1. npp doomed.txt
         2. git rm doomed.txt: it will get error because git is not yet tracked.
         3. rm doomed.txt (to delete file if git is not tracked)
         4. git checkout -- doomed.txt: (Restore back file)
7. History
   * + - 1. git log help
         2. git log
         3. git log --abbrev-commit
         4. git log --oneline --graph –decorate
         5. git log --all --oneline --graph --decorate
         6. git log b1f885c 3aff140
         7. git log --since=”3days ago”
         8. git log -- file.txt
         9. git log --follow -- level1
         10. git show *commitID*
8. Aliases
   * + - 1. git config --global alias.hist "log --all --oneline --graph --decorate"
         2. npp ~/.gitconfig
9. Ignoring Unwanted files and Folders
   * + - 1. npp .gitignore 🡪 Add <**.DS\_Store**>
         2. git add .gitignore
         3. git commit
         4. git push
         5. git commit -am "Modified README.md file"
         6. git pull
         7. git push

# Visual Merge/Diff tool Installation

1. Visual Diff/Merge Tool Setup
   1. P4Merge
   2. Git Configuration
      * + 1. it config --global merge.tool p4merge
          2. git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"
          3. git config --global diff.tool p4merge
          4. git config --global difftool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

# Comparisons

1. Comparing Working Directory and Staging Area
   * + - 1. git diff 🡪 By comparing the differences between what’s in my local working directory that has been recently modified but not yet stages, versus what’s currently staged.
         2. git difftool
2. Comparing working directory and git repository (Last Commit)
   * + - 1. git diff HEAD 🡪 git will compare the difference between the working directory and the last commit
         2. git difftool HEAD
3. Comparing Staging Ares and git repository (Last Commit)
   * + - 1. git diff --staged HEAD
4. Limiting the Comparison to one File (One Path)
   * + - 1. git diff -- filename
         2. git difftool -- filename
5. Comparing between the commits
   * + - 1. git log --online
         2. git diff a313760 b1f885c
         3. git difftool a313760 b1f885c
         4. git difftool a313760 HEAD
         5. git difftool HEAD HEAD^ 🡪 Compares HEAD and HEAD-1
6. Comparing between local and Remote branches
   * + - 1. git diff master origin/master
         2. git difftool master origin/master

# Branching & Merging

1. Branching Basics
   * + - 1. git branch -a
         2. git branch newbranch
         3. git checkout newbranch
         4. git branch -m newbranch mynewbranch
         5. git branch -d mynewbranch
         6. git checkout -b newbranch
2. Happy Path / Fast Forward Merges
   * + - 1. git merge newbranch

git merge newBranch

Updating 263fa9e..27aa870

Fast-forward

simple.html | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

* + - * 1. git push --set-upstream origin newBranch

1. Happy Path / Disable Fast Forward Merges
   * + - 1. git merge copyright --no--ff
2. Automatic Merges
   * + - 1. git merge copyRight -m "Merging the Changes"
         2. git merge --abort
3. Conflicting Merges and Resolution

# Rebasing

1. Simple Rebase Example

git checkout -b rebasebranch

git rebase master

git rebase –abort

git rebase (--continue | --abort | --skip)

# Stashing

1. Basic Stash Example
   * + - 1. git stash
         2. git stash apply
         3. git stash list
         4. git stash apply 1
         5. git stash apply stash@{1}
         6. git stash drop 1
         7. git stash drop
         8. git stash -u
         9. git stash + git stash drop 🡪 git stash pop
         10. git stash save “Simple Change”
         11. git stash show stash@{1}
         12. git stash pop 1
         13. git stash clear
         14. git stash branch branchname

# Tagging

1. Basic Tagging Examples

git tag mytag

git tag –list

git show mytag

git tag –delete mytag

git tag -a <v-1.0>

git commit --amend

git tag V1.2 -m “Release 1.2”

git tag -a v.9\_beta <commit id>

git tag -a v.9\_beta -f <commit id>

git push origin v.9\_beta

git push origin master –tags

git push origin :v0.8\_beta