GIT Fundamentals

# About GIT

* **GIT is a distributed DVCS** (Distributed Version control System)
* DVCS topologies – centralized (ex. enterprise), hierarchical (ex. Linux) & distributed (ex. Github)
* Advantages of DVCS / GIT
  + Backup is simple ex clone the repository in case of failover.
  + Reliable branching / merging enables stricter version control and apply patches easily.
  + Maintaining Full history enables regression analysis ex finding a bug
  + Version control used for deployment ex Heroku
* Created by Linus Torvalds and written in Perl & C.
* GIT is fast, scalable, simple, distributed DVCS. It provides strong branch & merge support

# Installation

## Windows

* Run the installer **msysGit**
* For Windows explorer integration, look for GIT extensions. (Tortis GIT obsolete)
* Set the Path environment to windows & Unix (better Unix commands)
* Set Line ending conversion to “Checkout window style commit unix style line endings” (cross platform)
* Once Installation is complete, go to windows PowerShell
  + **git --version** //returns version of the GIT installed
  + Go to test directory & run **git init** // makes the directory a local repo & means successful installation

## Mac OSX

* For Brew installation , run the command **brew install git**
* Download the DMG package for installation

## Linux

* Run the command **apt -get install git -core** (Debian/Ubuntu distros)
* Run the command **yum install git -core** (Fedora distros)

# GIT Configuration

* GIT can be configured at the 3 levels:
  + System level – File is located @ git\etc\gitconfig. File location attribute is **--system**
  + User level – File is located @ C:\users\<name>\gitconfig. File location attribute is **--global**
  + Repository level – it is located @ .git/config in each repo. File location attribute is **--file**
* If no location attribute specified then it refers to config of local repo.
* Set the attributes in git config.:
  + **git config --global user.name “Smita Busar”**
  + **git config --global user.email “smitabusar@hotmail.com”**
  + **git config --global core.editor notepad|vim|notepad++**
  + **git config --global help.autocorrect 1** //does a fuzzy match and execute if match found
  + **git config --global color.ui auto** // easy to parse logs
  + **git config --global core.autocrlf true** (win & other os compaitable) | **false** (win only) | **input** (mac only)
  + **git config --global push.default matching**(sets upstream branch to a match)|**simple** (confirms that user sets the upstream branch)
  + **git config --global alias.lga “log –oneline –graph – all –decorate”** // generates alias for given command . Execute as **git lga**
* **git config --global --list** // view the list of the global attributes
* **git config --global --unset user.name** // to unset an attribute from global config. You can manually remove the section.
* Configuration precedence : system<global<repo

## Deploy SSH Keys

* Verify that GIT has SSH key or not (GIT Bash)
  + **cd ~/.ssh** //change the directory to .ssh
  + **ls** // lists the files in directory. Look for id\_rsa.pub or id\_dsa.pub
* Generate a SSH key (GIT Bash)
  + **ssh-keygen -t rsa -C “**[**smitabusar@hotmail.com**](mailto:smitabusar@hotmail.com)**”** // generates the SSH key and stores it in id\_rsa.pub. Passphrase is recommended for security
  + **ssh -add id\_rsa** //to add the SSH key
* **clip < ~/.ssh/id\_rsa.pub** // copies the SSH Key to the clipboard. (GIT Bash)
* Add SSH Key to GitHub (GitHub site)
  + Go to account settings.
  + Click “SSH Keys” on left sidebar. Click “Add SSH Key”.
  + Give Title & Paste the key in “key” field. Click “Add Key”.
* Test it out (GIT Bash)
  + ssh –T [git@github.com](mailto:git@github.com) // gives successful authentication method.
* For more details please see <https://help.github.com/articles/managing-deploy-keys>

# GIT Commands

## Common Commands

* **HEAD** – most recent commit. **HEAD~1** –commit just before most recent one.
* **git status** // tells the status (untracked |staged |committed) of the files in repo.
* **git log** // shows the history of commits (SHA, author, date,message)
  + **--oneline** //condensed version. For line count pass append “| wc –l”
  + **--graph** //used to show the branches & merges.
  + **<RefName>/master** // shows the logs of the master branch of the remote repo.
  + **--all** //logs of all the branches
  + **--decorate** //decorates the log with head, tags etc
* **git shortlog** // shows the commits messages grouped by user
  + **–sne** // shows only count of commits, username & emails
* **git diff <SHA1>..<SHA2>** // shows the difference in 2 commits
  + **git diff HEAD~1..HEAD=git diff HEAD~1** // shows difference between most recent commits
* **git show HEAD //** displays last changes done to the repo. Useful for remote repo last commits
  + **HEAD~10** // to see changes of 10th commit
  + Also SHA can be used to show the needed commit
* **git tag** // lists all the tags in a repo.
  + **<Tname>** // creates a tag generally the version number
  + **-a <Tname> -m “message”**//creates a tag with message
  + **-s <Tname>//**asks for a passphrase to create a signed tag. For official commits
  + **-v <Tname>** // verifies if tag is a signed tag. Uses public private verification technique

## Commands for Local Repoistory

* **git add <filename>** // stages the new file in local repo. Filename is case sensitive
  + **-u** // stages all modified files (shows updated as well as deleted files)
  + **-A**// stages all the files including the untracked files and renamed files.
* **git commit** // GIT identifies a commit by a SHA1 hash. Each SHA can be abbreviated by first 5-8 chars. Git opens a .git/COMMIT\_EDITMSG text file to enter the commit message.
  + **-m “message”** // commits with the message
  + **-am “message”** // stages new or updated files & commits with the message
* git add & git commit can be used alternatively to stage changes as multiple commits. Ex: A feature was changed in F1 & bug was fixed in B1 then add & commit F1 & then add & commit B1
* **git checkout <filename>** // Revert back file to last commit and abandon any current changes
  + **<BName>** // checkouts a given branch. Make it the HEAD
  + **-b <BName>** // creates & checkout the branch
* **git reset** 
  + **--hard** // Remove all the last changes to all files & no files to commit. Working copy is clean
  + **--soft HEAD~1 //** Moves the head to last commit with all changes reflected in staging area. After making the fixes this can be recommitted.
* **git clean** // clean up the working copy from stray files like temp files or build files etc.
  + **-n** // what would be deleted
  + **-f** // actually remove the files
* **.gitignore** // File that specifies all files not accounted for version control ex log files
  + **Logs/** //any log directory anywhere in my application
  + **/logs** or **/logs/\*.txt** or **/log/\*.log** //log directory in my local repo or text /log files in logs
* **git reflog** //shows all the SHAs where HEAD has pointed in the past. Can be used to recover a branch.
  + Cleans up the commits in 30 days
* **git stash** // saves the changes of current branch & makes working copy clean
  + **list** //shows the changes in the stored in stash
  + **apply** // pull the changes from stash to working copy. Stash still has the changes
  + **pop** // removesthe changes from stash & apply to working copy
  + **drop** // drop the last changes from stash
  + **branch <BName>** //creates the branch, checks it out and apply the stash to it. Stash is empty
* **git rebase <target branch>** // put the current branch on top of target branch
  + --continue //continue the merge
* **git cherry-pick <SHA>** // applies that commit to the current branch ex applying a patch
* Merging, Rebasing & Cherry picking does not apply same commit twice

## Command for Remote Repository

* **git clone <git url>** // Downloads the remote repo to local repo. Automatically sets upstream branch
* **git remote** //show the name of repo which is origin. For local repo , it shows nothing.
  + **-v** // verbose shows the fetch & pull url
  + **add <RefName> <url>** // adds the remote repo to your local repo to evaluate patches or pull requests. url can use the http(s) or ssh protocol.
  + **rm <RefName>** // removes the connection to remote repo.
* GIT follows the following 4 protocols :
  + http(s) – 80/443 - <https://github.com/smitabusar/CodeFellows.git> – Read Write , authorization=password, firewall friendly
  + git – 9418 - [git://github.com/smitabusar/CodeFellows.git](https://github.com/smitabusar/CodeFellows.git) –Read only, anonymous only
  + ssh – [22-git@github.com:smitabusar/CodeFellows.git](mailto:22-git@github.com:smitabusar/CodeFellows.git) -Read Write, SSH keys for auth
  + file – file/users/smita/gitcode/codefellows-read write, local only
* **git branch** //returns the branch “master” (separate main dev from bug fixes. Temp working copy)
  + **-r** // list down all the branches in the repo
  + **--set-upstream-to=<remote>/<branch> master** // sets branch master in local repo to track <remote> <branch>
  + **<BName>** // creates a local branch. Label on SHA1 but unlike tags it moves on with the commits.
  + **<BName> <SHA>**// creates a local branch on specific SHA.
  + **-m <Bname1>** <Bname2> //renames a branch name
  + **-d <Bname>** //delete a branch. GIT prevents deletion if not merged.
  + **-D <BName>** //delete a branch without confirmation
* **git tag** // Tag is a stable point in codebase often version number.
* **git fetch** // fetches from the repo if there exists with RefName=”origin”
  + **<RefNname>** // fetches from that remote repo
* **git merge <source> <target>** // It is fast forward merge i.e. source commits are appended to target commits.
  + **Use git merge tool (kdiff3)** // help to merge changes from branches
* **git pull** // shortcut for git fetch + git merge origin/master but asks to set upstream
  + **<RefName> master** //fetches & merges the data without setting the upstream
* **git push** //enables to push the data from local copy to remote copy.
  + For http(s) protocol, it needs username password. It is not ideal so use SSH protocol.
  + To verify SSH status , give **ssh -vT** [**git@github.com**](mailto:git@github.com) in GIT Bash // gives successful message
  + **--tags** //pushes the tags to the GitHub
  + **<RefName>** //pushes to that remote repo
  + **<RefName> <BName>:<NewName>** //pushes & creates branch in remote repo. If new name is not specified, remote branch name is same as local branch name
  + **<RefName> :<BName>** //deletes the remote branch