GIT CHEAT SHEET

- INSTALLATION & GUIS
- With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.
- GitHub for Windows
- htps://windows.github.com
- GitHub for Mac
- htps://mac.github.com
- For Linux and Solaris platforms, the latest release is available on the official Git web site.
- Git for All Platforms
- htp://git-scm.com

- SETUP
- Configuring user information used across all local repositories
- git config --global user.name “[akhil chaithanya]”
- set a name that is identifiable for credit when review version history
- git config --global user.email “[akhil10.gandla@gmail.com]”
- set an email address that will be associated with each history marker
- git config --global color.ui auto
- set automatic command line coloring for Git for easy reviewing
- git config –list
- It will list all the Information

- SETUP & INIT
- Configuring user information, initializing and cloning repositories
- git Init
- initialize an existing directory as a Git repository
- git clone [url]
- retrieve an entire repository from a hosted location via UR

- STAGE & SNAPSHOT
- Working with snapshots and the Git staging area
- git status
- show modified files in working directory, staged for your next commit
- git add [file]
- add a file as it looks now to your next commit (stage)
- git reset [file]
- unstage a file while retaining the changes in working directory
- git diff
- diff of what is changed but not staged
- git diff –staged
- diff of what is staged but not yet committed
- git commit -m “[descriptive message]”
- commit your staged content as a new commit snapshot

- BRANCH & MERGE
- Isolating work in branches, changing context, and integrating changes
- git branch
- list your branches. a * will appear next to the currently active branch
- git branch [branch-name]
- create a new branch at the current commit
- git checkout
- switch to another branch and check it out into your working directory
- git merge [branch]
- merge the specified branch’s history into the current one
- git log
- show all commits in the current branch’s history

- INSPECT & COMPARE
- Examining logs, diffs and object information
- git log
- show the commit history for the currently active branch
- git log branchB..branchA
- show the commits on branchA that are not on branchB
- git log --follow [file]
- show the commits that changed file, even across renames
- git diff branchB...branchA
- show the diff of what is in branchA that is not in branchB
- git show [SHA]
- show any object in Git in human-readable forma

- SHARE & UPDATE
- Retrieving updates from another repository and updating local repos
- git remote add [alias] [url]
- add a git URL as an alias
- git fetch [alias]
- fetch down all the branches from that Git remote
- git merge [alias]/[branch]
- merge a remote branch into your current branch to bring it up to date
- git push [alias] [branch]
- Transmit local branch commits to the remote repository branch
- git pull
- fetch and merge any commits from the tracking remote branch

- TRACKING PATH CHANGES
- Versioning file removes and path changes
- git rm [file]
- delete the file from project and stage the removal for commit
- git mv [existing-path] [new-path]
- change an existing file path and stage the move
- git log --stat -M
- show all commit logs with indication of any paths that moved

- REWRITE HISTORY
- Rewriting branches, updating commits and clearing history
- git rebase [branch]
- apply any commits of current branch ahead of specified one
- git reset --hard [commit]
- clear staging area, rewrite working tree from specified commit
- git commit –amend
- Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit’s message

- IGNORING PATTERNS
- Preventing unintentional staging or committing of files
- Logs/
- *.notes
- Pattern*/
- git config --global core.excludesfile [file]
- system wide ignore patern for all local repositories

- TEMPORARY COMMITS
- Temporarily store modified, tracked files in order to change branches
- git stash
- Save modified and staged changes
- git stash list
- list stack-order of stashed file changes
- git stash pop
- write working from top of stash stack
- git stash drop
- discard the changes from top of stash stack

- Undoing changes
- **git revert <commit>**
- Create new commit that undoes all of the changes made in <commit> then apply it into the current branch
- **git reset <file>**
- Remove <file> from the staging area but leave the working directory unchanged this unstages without overwriting any changes
- **git clean –n**
- Shows which file would be removed from working directory use the –f flag in place of –n flag to execute the clean

- Git pull updates
- git pull –rebase <remote>
- Fetch the remote’s copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

- git push updates
- git push <remote>--force
- Forces the git push even if it results in a non-fast-forward merge. Do not use the --force flag unless you’re absolutely sure you know what you’re doing.
- git push <remote> --all
- Push all of your local branches to the specified remote.
- git push <remote> --tags
- Tags aren’t automatically pushed when you push a branch or use the --all flag. The --tags flag sends all of your local tags to the remote repo.

- Create a new project
- mkdir <file name>
- cd <file name>
- git Init
- git touch file.txt <for text documentation adding>
- git status
- git add *
- git commit –m “write according to your project”
- create a new repo
- copy the url link and paste in gitbash
- git push origin master