GIT GUIDE (GIT Cheat Sheet)

    author:  AliBinary

    Email: AliGhanbariCs@gmail.com

    GitHub: https://github.com/AliBinary

    created: September 2023

Basics

Info

git -v # Version

git -h # Help

command prompt (in windows)

note: In Linux and Mac, the commands may be slightly different

mkdir Moon # Make directory named "Moon" (or u can use "md Moon")

cd Moon # Enter to "Moon" directory

cd.. # Come back in directory

cls # Clears the screen

dir #Shows files and folders in directory

dir /a #Shows hidden files and folders in directory

del .git # Remove .git subdirectory

rd Moon # Removes directory

echo hello > file1.txt # Write text to a file

echo world >> file1.txt # Append text in a file

ren file.txt FILE.js # Rename a file or files

code file.txt # Open with vscode (if such a file doesn’t exist, it will be created)

Configuration

git config --global user.name "Ali Gh” # Use “ if your name/email has space inside it

git config --global user.email @gmail.com

code # Open vscode

git config --global core.editor "code --wait" # Select vscode for default editor

git config --global -e # Edit our global configuration settings in our default editor

git config --global core.autocrlf true # "true" keyword for win, "input" for mac/linux

# These 2 lines botton is for set vscode for default difftool

git config --global diff.tool vscode

git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"

git config --global -e

**Git ignore File**

File creation and setting

echo bin/ > .gitignore # This file use for ignore list to prevent commit them (it works when commited and prevent only files that do’nt exist in index/staging area)

# U can custom .gitignore even for classes and functions of ur code! (see https://github.com/github/gitignore)

Useful Terms

{

**GIT OBJECTS:**

Commits

Blobs # Files

Trees # Directories

Tags

}

{

**select files for do something:**

. # All files in directory

\*.txt # Use pattern

file1.txt file2.txt # Select one or more than one files

}

**Creating Snapshots**

**Initializing a repository**

git init

**Staging files**

git add file.js # Stages a single file

git add file1.js file2.js # Stages multiple files

git add \*.js # Stages with a pattern

git add . # Stages the current directory and all its content

**Viewing the status**

git status # Full status

git status -s # Short status (left: staging area, right: working directory)

git ls-files

**Committing the staged files**

git commit -m “Message” # Commits with a one-line message

git commit # Opens the default editor to type a long message

**Skipping the staging area**

git commit -am “Message”

**Removing files**

git rm file.js # Removes from working directory and staging area

git rm --cached file.js # Removes from staging area only

**Renaming or moving files**

git mv file1.js file1.txt

**Viewing the staged/unstaged changes**

git diff # Shows unstaged changes

git diff --staged # Shows staged changes

git diff --cached # Same as the above

git diff file1.txt #see changes of a file

**Viewing the history**

git log # Full history (u can see commit ID or commit hash here)

git log --oneline # Summary

git log --reverse # Lists the commits from the oldest to the newest

**Viewing a commit**

git show 921a2ff # Shows the given commit (with commit ID)

git show HEAD # Shows the last commit

git show HEAD~2 # Two steps before the last commit

git show HEAD:file.js # Shows the version of file.js stored in the last commit

git ls-tree HEAD~1 #Shows all the files and directories in a commit

**Unstaging files (undoing git add)**

git restore --staged file.js # Copies the last version of file.js from repo to index

**Discarding local changes**

git restore file.js # Copies file.js from index to working directory

git restore file1.js file2.js # Restores multiple files in working directory

git restore . # Discards all local changes (except untracked files)

git clean -fd # Removes all untracked files

**Restoring an earlier version of a file**

git restore --source=HEAD~2 file.js

**Browsing History**

**Viewing the history**

git log --stat # Shows the list of modified files

git log --patch # Shows the actual changes (patches)

**Filtering the history**

git log -3 # Shows the last 3 entries

git log --author=“Mosh”

git log --before=“2020-08-17”

git log --after=“one week ago”

git log --grep=“GUI” # Commits with “GUI” in their message

git log -S“GUI” # Commits with “GUI” in their patches

git log hash1..hash2 # Range of commits

git log file.txt # Commits that touched file.txt

git log --oneline -- file.txt # Commits that have modified the file in oneline

git log –-oneline –-patch –- file.txt

**Formatting the log output**

git log --pretty=format:”%an committed %H”

git log –-pretty=format:”%Cgreen%an%Creset committed %h on %cd”

# for more options read https://git-scm.com/docs/pretty-formats

**Creating an alias**

git config --global alias.lg “log --oneline --all --graph" # Creates a Snippet for commands

git config –-global alias.prt “log --pretty=format:’%an committed %h’”

git config --global alias.unstage “restore --staged .”

git config -l # List of all git aliases

**Viewing a commit**

git show HEAD~2

git show HEAD~2:file1.txt # Shows the version of file stored in this commit

git show HEAD~2 --name-only # Shows the files have been modified in commit

git show HEAD~2 --name-status # Gives more information

**Comparing commits**

git diff HEAD~2 HEAD # Shows the changes between two commits

git diff HEAD~2 HEAD file.txt # Changes to file.txt only  
#similar to log command, here we have option like --name, -status, -only

**Checking out a commit**

git checkout dad47ed # Checks out the given commit

git checkout master # Checks out the master branch

git checkout . # restore all changes to past of last commit

git checkout file.txt #restore one file to past of last commit

git log --oneline # in ‘detached HEAD’ state, can’t see all the commits earlier

git log --oneline --all #to see all extra commits

**Finding a bad commit**

git bisect start

git bisect bad # Marks the current commit as a bad commit

git bisect good ca49180 # Marks the given commit as a good commit

git bisect reset # Terminates the bisect session

**Finding contributors**

git shortlog # Use -h to see options

git shortlog -n -s -e # Number of commits per author with their email

git shortlog -n -s -e --before=”” --after=”” # Contributors per given date range

**Viewing the history of a file**

git log file.txt # Shows the commits that touched file.txt

git log --stat file.txt # Shows statistics (the number of changes) for file.txt

git log --oneline --stat file.txt

git log --patch file.txt # Shows the patches (changes) applied to file.txt

Restoring a Deleted File

git rm file.txt

git commit -m “Remove file.txt”

git log --oneline -- file.txt # Shows all the commits that touched this file

git checkout a642e12 file.txt # ID is required, where the last version of the file was

git status -s # We added the past to out file. Let’s run a short status

git commit -m “Restore file.txt” # This is how we can restore a deleted file 😊

**Finding the author of lines**

git blame file.txt # Shows the author of each line in file.txt

git blame -e file.txt # Show Author’s email

git blame -L 1,3 file.txt # Filter by lines

**Tagging**

git tag v1.0 # Tags the last commit as v1.0 (lightweight tag)

git tag v1.0 5e7a828 # Tags an earlier commit (lightweight tag)

git tag # Lists all the tags

git tag -a v1.1 -m “My version 1.1” #Tags a commit (annotated tag)

# We can associate a message with a tag (information about tagger, email, etc.)

git tag -n # To see the message of tags

# in lightweight tags-> their commit message and in annotated tags-> custom message

git tag -d v1.0 # Deletes the given tag  
git checkout v1.0 # We can reference a commit using the tag  
git show v1.1

**Branching & Merging**

**Managing branches**

git branch bugfix # Creates a new branch called bugfix

git branch # Shows list of branches (asterisk shows current branch)

git status # another way to know the current branch

git checkout bugfix # Switches to the bugfix branch ( in the past used)

git switch bugfix # Same as the above ( I think this is better to use)

git branch -m bugfix bugfix/signup-form # Rename a branch

git log --oneline

git switch -C bugfix # Creates and switches

git branch -d bugfix # Deletes the bugfix branch

**Comparing branches**

git log master..bugfix # Lists the commits in the bugfix branch not in master

git log master..bugfix --oneline

git diff master..bugfix # Shows the summary of changes

git diff bugfix # Same as the above (because We’re currently on master)

git diff --name-only bugfix # List of files that are different in 2 branches

git diff --name-status bugfix # Same as the above (Gives more information)

**Stashing**

git stash push -m “New tax rules” # Creates a new stash (untrack files are not included)

git stash push -am “message” # Now untracked files is going to be included in the stash

git stash list # Lists all the stashes

git stash show stash@{1} # Shows the given stash

git stash show 1 # shortcut for stash@{1}

git stash apply 1 # Applies the given stash to the working dir

git stash drop 1 # Deletes the given stash

git stash clear # Deletes all the stashes

**Merging**

git log --oneline --all --graph # See the branches and how they diverged

git merge bugfix # Merges the bugfix branch into the current branch

git merge --no-ff bugfix # Creates a merge commit even if FF (Fast Forward) is possible

git config ff no # Disable FF only in the current repo

git config --global ff no # Same as the above (apply to all of your repos)

git merge --squash bugfix # Performs a squash merge

git merge --abort # Aborts the merge (when we don’t want to resolve conflicts now)

# go back to the state before we started the merge

git merge --quit # You can quit merging when there are conflicts

**Viewing the merged branches**

git branch --merged # Shows the merged branches

git branch -d bugfix # Delete a branch (it’s safe to delete a merged branch)

git branch --no-merged # Shows the unmerged branches

**Merge Conflicts** # Conflicts happen when a file has differences in 2 branches

{Conflicts happen when:

Change1, Change2

Change, Delete

Add1, Add2

}

git status # Look under unmerged paths, This is where u can find the conflict of files

code file.txt # See markers that represent the changes in the current and other branch

# There are different ways of resolving conflicts, we can manually edit the file

# Remember that Ideally you should not introduce new code here

git add file.txt

git status # We no longer have unmerged paths(means we no longer have conflicts)

git commit # We’re done with the merge

Visual Merge Tools # Kdiff, p4Merge or WinMerge

# Download and install P4Merge

git config --global merge.tool p4merge # Configure P4merge as our default merge tool

git config --global mergetool.p4merge.path “C:\Program Files\p4merge\”

git mergetool # Use our external merge tool

git commit # resolve conflicts, close p4merge and commit changes

**Rebasing**

git rebase master # Changes the base of the current branch

**Cherry picking**

git cherry-pick dad47ed # Applies the given commit on the current branch

**Collaboration**

**Cloning a repository**

git clone url

**Syncing with remotes**

git fetch origin master # Fetches master from origin

git fetch origin # Fetches all objects from origin

git fetch # Shortcut for “git fetch origin”

git pull # Fetch + merge

git push origin master # Pushes master to origin

git push # Shortcut for “git push origin master”

**Sharing tags**

git push origin v1.0 # Pushes tag v1.0 to origin

git push origin —delete v1.0

**Sharing branches**

git branch -r # Shows remote tracking branches

git branch -vv # Shows local & remote tracking branches

git push -u origin bugfix # Pushes bugfix to origin

git push -d origin bugfix # Removes bugfix from origin

**Managing remotes**

git remote # Shows remote repos

git remote add upstream url # Adds a new remote called upstream

git remote rm upstream # Remotes upstream

**Rewriting History**

**Undoing commits**

git reset --soft HEAD~1 # Removes the last commit, keeps changed staged

git reset --mixed HEAD~1 # Unstages the changes as well (default option)

git reset --hard HEAD~1 # Discards local changes

git reset --hard 67a2e25 # Recover any reset merge commit

**Reverting commits**

git revert 72856ea # Reverts the given commit

git revert HEAD~3.. # Reverts the last three commits

git revert --no-commit HEAD~3..

git revert -m 1 HEAD # We choose the First parent and its last commit for target

**Recovering lost commits**

git reflog # Shows the history of HEAD

git reflog show bugfix # Shows the history of bugfix pointer

**Amending the last commit**

git commit –amend # make changes to the previous commit

**Interactive rebasing**

git rebase -i HEAD~5

Add changes to GitHub Repository with Git

# first copy web URL from repository in github

git clone [URL]

cd [REPOSITORY NAME] # go to the directory for do changes

git add .

git comit -m 'adding name'

git push -u origin main # update (or send) changes to repository

# the option <u> is to use only the <git push> command next time

git pull #download (update) our directory from github repository

Storing Images and Demos in your Repo

Clone a fresh copy of your repo

git clone url

Create a new branch

git checkout --orphan assets # Create and switch to a new branch called “assets

# The --orphan flag creates a new branch but without any prior commits

Remove files from the working tree

git rm -rf # Delete all files that the working tree recognizes

# Any files that were not added to the tree will remain left behind in the folder

To make sure: See current branch

git branch # the branch with the \* next to it is the current branch

Add your images and screenshots and commit the change

git add screenshot.png demo.git logo.png

git commit -m “Added Assets”

Finally push your changes

git push origin assets

Use the images in your README

# You can now use ![Demo animation](../assets/demo.gif?raw=true) in your README to have the gif (or image or any file you want) display on your master’s readme.