Git Commands

Link to Git Cheat Sheet

Terms

- working directory : Where you are coding (can be same as git directory)
- staging area : Where you commit changes from working directory to git directory (but not pushed in)
- git directory : Where your local repository is hosted
- local : Your git repository on your computer
- remote : Git repository NOT on your computer, and you are pushing / pulling from
- origin : Where the repository was ORGINALLY cloned (i.e. master version)
- pull : Fetch from remote to local
- push : Push from local to remote
- commit hash : ID of commit
- tree : contains a list/history of git changes
- upstream : changes/commits that are in origin

Setting up

- `git version`
  - Checks your git version
- `git config --global user.name "Your Name"`
  - Your commit name
  - Leave "<>" blank to check current name
- `git config --global user.email "youremail@gmail.com`
  - Your commit email
  - Leave "<>" blank to check current email
- `git config --list`
  - Shows all config settings
- `git init`
  - Creates a git repository at current directory (creates .git directory)
  - remove .git directory to remove git status
- `git add <filename>`
  - `-A` add all files in current directory

    - Stages all files in current directory (adds to git repo)
- `git clone <URL>.git <cloning_directory>`
    - Leaving `<cloning_directory>` blank to clone at current directory
    - URL should end with a .git to specify its a git repository
    - Some git services like GitHub are automatically detected so doesn't need the .git
    - `--depth <no.>` determines amount of commits to clone
    - `--recurse-submodules` clones submodules
- `git remote -v`
    - Check if remote is the origin
- `git branch -a`
    - List all branches in the local and remote

Basic Commands

- `git help <command>`
    - Gets help for specific command
    - Alt: `git <command> --help`
- `git status`
    - Shows status, includes:
    - Untracked files (fresh repo), Uncommited/commited changes, Conflicts
    - Use before pushing and pulling to check for possible conflicts!
- `git add <filename>`
    - `-A` add all files in current directory
    - Stages files (adds to commit)
- `git commit -m "Commit Message"`
    - Commit changes
    - Use without `-m` option to check full commit status in a code editor
    - Use with `--amend` to amend last commit
- `git log`
    - Shows recent changes
    - `q` to quit, `Return` for next line
- `git diff`
    - Shows any differences
    - optional: `<commit-hash> <commit-hash>`; `<branch-name> / head` - latest -- `--stat` for a condensed summary (file changes)
    - `q` to quit, `Return` for next line
- `git fetch`
    - Fetches (download) from remote to local
- `git pull (origin) (branch)`
    - Pulls from remote to local
    - bracketed items are optional (remove brackets!):
    - `origin` - specifies from origin directory
    - `branch` - specifies branch name, current branch by default

- `git push (origin) (branch)`
  - Pushes from local to remote
  - bracketed items are optional (remove brackets!):
  - `origin` - specifies from origin directory
  - `branch` - specifies branch name, current branch by default
  - `-u` - set upstream (for pushing in a new branch from local to remote-origin)
- `git checkout <commit-hash>/<branch-name>`
  - Switch your current git repository to a specific commit OR branch
  - `git checkout main` to switch back to latest
  - `git checkout head` ADVANCED: Detaches current commit from working tree
  - Leave option empty to get the current status
- `git branch <branch-name>`
  - Creates a new branch based on current commit
  - Use checkout to specify specific commit / switch to new branch
  - Leave option empty to get the current branch you are on
- `git stash`
  - Stash current changes
- `git stash show`
  - Show stashed changes
- `git stash pop`
  - Pop stash changes (unstash)
- `git mv <file_location> <new_location/new_name>`
  - Rename / move existing files to another name / location
  - Useful for changing file name case (windows is non-case sensitive and won't detect file name case changes)
  - `git mv src/graphics.cpp src/GRAPHICS.cpp` - renaming to uppercase
  - `git mv src/graphics.cpp src/graphics/graphics_auto.cpp` - renaming and moving

Merging

- Squash and merge (makes your commitS into one large commit)
  - Merging from `bugfix` branch INTO `master`
    - `git checkout master`
    - `git merge --squash bugfix`
    - resolve conflicts....
    - `git rebase`
    - resolve conflicts....
    - `git commit`
- Updating your own branch with stuff from master
  - `git checkout <your_branch>`
  - `git rebase master`
  - resolve conflicts....
  - `git force -f`

    - Why force?
        - Since rebase rewrites commit history, the previous version of the branch is no longer valid.
        - A force push replaces the old local branch history with the rebased one, removing the old, now obsolete commits.
- Merge with rebase (retains all your commits and move them over)
    - Rebasing `bugfix` INTO `master`
        - `git checkout bugfix`
        - `git rebase master`
        - resolve conflicts....
        - `git checkout master`
        - `git merge bugfix`
        - push!

Git Large File Storage (LFS)

When adding files larger than 100mb, versioning for it will no longer be available, forcing the repository to be completely usable, use git LFS to manage these large files

- `git lfs install` : install LFS if you dont have it
- `git lfs track "filename"` OR `git lfs track "*.psd"` <- tracks all files of specified type
- `git add .gitattributes` - if git attributes doesn't exist! git attributes is used for tracking LFS files
- `git add -A`
- `git commit -m "Commit message"`

Advanced Commands

WARNING : Commands below will might result in an unstable repository or overwritten changes when used incorrectly! Use with care!

DANGER LEVEL

0. No danger (but access advanced data)
1. overwrites local changes (if no errors)
2. overwrites local & can affect remote changes
3. overwrites local & overwrites remote changes (poweruser!)

- `git rebase` [1]
    - Compare between local and remote and tries to fix tree
- `git rebase origin/<branch name>` [2]

    - i.e. `git rebase origin/master` rebases `master` branch to current branch, basically apply master branch changes into current branch
- `git reset` [1]
    - **REMOVES** unstaged / uncommitted changes
    - Resets repository to last state
    - `--hard` to force reset
    - `<commit-hash>` to reset to specific commit
- `git clean -fxd` [1]
    - Force **REMOVES** all unstaged / uncommited changes
- `git push -f` [3]
    - Does a force push, this **REWRITES** the remote tree with your local tree!
- `git reset --soft HEAD~3` [2]
    - Rewinds/undo the last 3 commits from current head, change 3 to any number
- `git add dir/dir2/ -f` [2]
    - Force adds/stages directory, **OVERWRITES** gitignore
    - Replace with `<filename>` for file
- `git checkout --orphan` [2]
    - Makes a new orphan branch with **NO commit history**
- `git rebase -i --root` [3]
    - Allows manual rebasing in an interactive editor FROM starting of repo (root)
    - Changes made here will **MESS UP** your local tree unless you know what you are doing
    - `git rebase` to reset rebased changes
    - `git rebase -i <commit-hash>` to specify starting commit
    - `--root` specifies starting commit to be from first
- `git rebase -i HEAD~3` [3]
    - Allows manual rebasing in an interactive editor FOR last 3 commits
    - Change the respective command in your text editor, i.e. `pick` to `drop`; to affect the commit
- `git fsck --unreachable` [0]
    - show list of unreachable (dangling) commits
- `git gc --prune=now` [3] - removes unused history!
    - remove all unreachable (dangling) commits

Cross-GUI Interations

- `github .`
    - open github GUI at current location
    - replace `.` with folder path for specified location
- `explorer .`
    - open file explorer at current location
    - replace `.` with folder path for specified location
- typing in file explorer address path

    - `cmd` - open command prompt
    - `powershell` - open powershell

Helpful Windows CMD Commands

- `cd ..`
    - Go to parent directory
- `cd <directory_name>`
    - Go to next specified directory
- `dir`
    - Display all files in current directory
    - `help dir` for list of sorting options
    - `dir /os` - file ordered by smallest size first
    - `dir /o-s` - file ordered by largest size first
- `rmdir /s /q <directory_name>`
    - Force removes a directory and all its contents
    - `/s` Removes all directories and files in specified directory
    - `/q` Quiet mode: Do not ask if "Are you sure"
- `mkdir <folder_name>`
    - Creates a new folder in current directory
- `echo your_text_here > filename.extension`
    - Create file with echo
- `code filename.extension`
    - Create file with vscode
- `notepad filename.extension`
    - Create file with notepad
- `type filename.extension`
    - Show file contents
- `del <file_name>`
    - Delete singular file
    - `/f` Force delete (read-only etc)
    - `/q` Quiet mode: Do not ask if "Are you sure"
    - `help del` for list of deleting options
- `set <variable_name>=<contents>`
    - Stores a variable, persistent in current cmd session
    - `<contents>` can be a command with `%command%`
    - i.e. `set curpath=%cd%` set variable curpath to current directory
- `echo %<variable_name>%`
    - Echo variable
    - Surround with `"<>"` to create a string
- `%<variable_name>%`
    - Utilise variable, use cases: for specifying directories etc
- `<filename>.exe/.bat`

    - Execute said file, .bat is a bash script
    - .sh (shell script) is linux equivalent
- `<any-cmd> > filename.extension`
    - Pipe results to specified file

Git Configurations

.gitignore

Excludes certain files from being pushed. Can be overriden with `git add <> -f` https://git-scm.com/docs/gitignore

- prefix `!`
    - Should be placed after all exclude criteria
    - Applied to any parameters, include said
- `*.extension`
    - Exclude file extension
- `<folder_name>/`
    - Exclude everything in folder
    - Use `[Dd]` for wildcard, specified allows D or d
    - i.e. `[Dd]ebug/`

.gitattributes

Various git config attributes Involves language statistics watcher https://git-scm.com/docs/gitattributes

- prefix `*.extension`
    - Exclude file extension
    - i.e. `*.tex linguist-vendored`
- prefix `<project-root>/<directory-1>/<directory-2>/**`
    - Exclude directory
    - i.e. `opengl-dev/lib/** linguist-vendored`
- prefix `<filename.extension> -`
    - Exclude specific file
    - i.e. `jquery.js -linguist-vendored`
- option `linguist-vendored`
    - Exclude as vendored code (from library)

- option `linguist-documentation`
  - Exclude as documentation
- option `linguist-generated`
  - Exclude as generated files
- option `linguist-language=<language>`
  - Reclassifies file to another language
  - i.e. `*.rb linguist-language=Java`

.gitkeep

Placeholder file for empty directories

Signing commits on github

- `git config commit.gpgsign true`
  - Enable signed commits for current repository, switch to `false` to disable
- `git config --global commit.gpgsign true`
  - Enable signed commits globally, switch to `false` to disable

Setting up (*Using GPG*)

1. Install GPG command line tools https://www.gnupg.org/download/ For windows, get Gpg4win
2. Locate gpg.exe If following default installation path, should be under: `C:\Program Files (x86)\GnuPG\bin` cd to said directory
3. `gpg.exe --full-generate-key`
4. kind of key: Enter to accept default
5. key size: Enter to accept default
6. length of time: Enter to accept default (doesn't expire)
7. Verify selections above
8. Enter your user ID information. For a private email address, use github provided `no-reply`! Your `no-reply` github email is viewable under github website settings > `Emails` (https://github.com/settings/emails) It will look something like this `ID+USERNAME@users.noreply.github.com` or this `USERNAME@users.noreply.github.com`
9. Type a secure passphrase
10. Get list of keys with: `gpg.exe --list-secret-keys --keyid-format=long`
11. copy the ID, ID should be under `sec` right after first `\` For example: `sec 4096R/3AA5C34371567BD2 2016-03-10 [expires: 2017-03-10]` `3AA5C34371567BD2` is the ID
12. `gpg.exe --armor --export <INSERT_ID_HERE>`
13. Copy the GPG key generated

14. Go to your github website settings > `SSH and GPG keys` (under `Access` section) (https://github.com/settings/keys)
15. Click `New GPG key`
16. Confirm and authenticate
17. `git config --global gpg.program "C:\Program Files (x86)\GnuPG\bin\gpg.exe"` replace path there to path to your `gpg.exe` if different
18. `git config --global gpg.program gpg`
19. `git config --global user.signingkey <INSERT_ID_HERE>`
20. Profit! Your key is stored and automatically applied as long as you are using the same device with Cpg4win installed.

Removing passphrase

1. `gpg --passwd <INSERT_ID_HERE>`
2. Enter your passphrase as normal
3. Keep the new passphrase empty and accept the warnings
4. Repeat step 3
5. Passphrase is now removed for this device

Revision #14
Created 15 January 2024 07:08:56 by neinwhal
Updated 5 March 2025 05:23:55 by neinwhal