**Git Command Reference (Powershell)**

**Example: Creating a Github Repository**

1. Login to Github
2. Click on “Create a repository”
3. Copy the “HTTPS clone URL”
4. Open “Git Shell” command line tool
5. Navigate to code directory
6. git init
7. git remote add origin <https://github.com/>....
8. Add & Commit Files
9. git push -u origin --all (Make sure the remote master is not checked out)

**Example Using Remote Clone (Atlassian style of Git usage)**

Do NOT need to git init when cloning

Create local directory where Git repository will reside and navigate there

!!! Rembmer the period !!!

git clone file:///”C:\...” . -> Period says clone to current directory

git clone file:////LB047409/TestGitMasterVS2012 . -> Remember “.”

git clone “C:\....” . -> specify “.” to clone to current directory

git clone https://... -> If clone from dir, may need to cd to new dir created

git remote set-url origin file:///”C:\..” -> Sets the remote push/pull

git remote add origin https://... -> Sets the remote push/pull

git remote -v -> displays the actual (push) and (pull) locations

git remote show origin -> Displays what local branches merge with what remote branches

!!! REPEAT STEPS FROM HERE DOWN !!!

git checkout master (If not already checked out)

!!! Shouldn’t need to do this step directly after a clone !!!

git pull -> pull from remote origin AND merge into local master

git checkout master

git fetch -> will pull from remote origin (will not merge, points to origin/master)

git merge origin/master ->

git lga -> display current remote branch (v1.0)

git branch v2.0 -> create a new branch before beginning work

!!! DO WORK !!!

git commit -am “v2.0 – c1”

git checkout master

git pull -> fetch latest updates from the server and merge into local master

git fetch

git merge origin/master -> merge local-remote-master with local-master

git merge v2.0 -> merge v2.0 into local-master

git checkout master -> (local master)

git merge v2.0 -> merge local v2.0 with local master

git merge origin/master -> merge local-remote-master with local-master

Delete Branches before pushing otherwise branches will be tracked

git branch -d branch\_name -> deletes local branch

Delete remote branches if the local branches weren’t deleted before the push

git push origin :branch\_name -> deletes the remote branch

!!! make sure remote master is NOT checked out before this step !!!

git push --dry-run -> shows what will be pushed

git push -u origin --all -> push all of your branches to the server (-u == --set-upstream)

git push -u origin --tags -> push all of your tags to the server (-u == --set-upstream)

git push --set-upstream origin -> use this to push local-master to remote-master

git push --set-upstream origin v2.0 -> use this to set branch where the push will go

git remote show origin -> will display the remote branches

!!! make sure to create a new branch from the master before doing any new work !!!

Help

git config --help -> gives help information related to “git reset”

git reset --help -> gives help information related to “git reset”

Git Version

git --version

Create Repository

Navigate to directory in Powershell

git init

Clone a Remote Repository

git clone https://github.com/jquery/jquery.git

Add File to Staged

git add README.TXT

git add –A (Adds ALL files including untracked ones)

git add –u (Adds files to Staging)

Delete File to Repository/Staged

del Readme.txt

git status –> shows file delete

git add –u -> adds file to staged

git status –> shows staged files

Rename File

ren file3.txt file3a.txt

git add -A

Checkout

git checkout Readme.txt -> specific file name

git checkout feature1 -> checkout the feature1 branch

git checkout master -> checkout the master

git checkout –b feature2 -> creates a branch while checking-out

Undo Checkout

git checkout Readme.txt

git reset--hard -> Reset the working copy back to the HEAD

Undo Checkin (Undo Last Commit and take out of repository)

git reset --soft HEAD~1 -> Removes Last Commit and moves those changes back to repository

git reset --hard HEAD~1 -> Moves HEAD back one and discards all of the changes

Exclude Files from Git (Example)

(create a logs directory and add files)

add .gitignore file to the root and add the following:

/logs -> will exclude all files in the logs dir

/logs/\*.txt -> will exclude all txt files in the logs dir

git add .gitignore

git commit –m “Added .gitignore”

Clean

git clean -n -> Displays what actions will take place if a clean is executed

git clean -f -> Performs the clean operation

Status - View Status of Items to Commit

git status

Difference between two(2) Commits

List Files with Differences between two commits

git diff SHA1 SHA2

Diff by HEAD (HEAD is latest commit. HEAD~1 go back one commit. All below are equal)

git diff HEAD~1

git diff HEAD~1..

git diff HEAD~1..HEAD

Diff by Hash (First part of SHA1 hash)

git diff f6d4..7642

Add Updated Files to Commit List (Staged)

git add -u

git add --all -> adds all files to the staging area

git reset -> This will remove all file from the staged list

git reset <filename> -> This will remove a single file from the staged list

Commit (Check-In)

git commit -m “Enter Some Message Here” -> Add message with commit

git commit -am “Message Here” -> Add new files and message with commit

git commit --amend -> allows you to add to the previous commit

Add a Commit Message via Text Editor

git commit (First Call – enter commit message and save)

git commit (Second Call – commits files)

git log – Show the files committed

Show Commit Details

git show HEAD

Show Files in Repository

git ls-tree -r master --name-only

git ls-tree -r master --name-only > info.txt -> Dump into a text file for viewing

Log (Show Repository Log)

git log

git log --oneline -> Shows commit descriptions and SHA1 hash codes

git log --oneline | wc -l -> Shows how many commits exist

git log --oneline --graph -> Shows different branches and merges

git shortlog -> Show individual contributor commits

git shortlog –sne -> s-Summary, n-Ordered by number of commits decreasing, e-Email

git reflog -> shows all the places where HEAD has pointed. Can use the reflog to find information that may have been cleaned up by Git’s garbage collection.

Add Remote & Fetch

git remote -v -> displays the actual (push) and (pull) locations

git remote add origin http://... -> allows you to define a remote

git fetch -> will pull from the repository and create a new branch

git fetch origin -> specifies which remote to pull from

git log origin/master -> name of remote branch not yet in local working copy

git merge origin/master -> merge from origin/master into current branch

git branch -r -> will show the branch just merged from

*git pull -> combines git fetch; git merge origin/master*

git fetch origin master -> pulls from remote repository branch

Branches – Set Upstream Branch and Perform Pull

git branch --set-upstream master origin/master -> set remote branch that mirrors local branch

git pull -> pull any changes down

Branches (If a branch is the HEAD, every commit will be add to the branch)

git branch -> show branches

git branch -r -> show remote branches

git log --graph --oneline --all --decorate -> shows log with tags (See Aliases)

git branch feature1 -> creates a branch

git checkout feature1 -> checks out the feature1 branch

git branch fix1 974b56a -> creates a branch from a specific commit SHA1

git branch –m fix1 bug1234 -> renames/moves the fix1 branch to branch bug1234

git branch –d bug1234 -> delete branch bug1234 (not physically, still see SHA1)

git branch –D bug1234 -> force deletion

git checkout –b feature2 -> creates a branch while checking-out

git branch bug1234 234e42e -> This can be used to UNDELETE a branch that was removed

git branch feature3 v1.0 -> create branch feature3 from tag “v1.0”

git checkout feature3 -> checks out the newly created branch

git fetch origin master -> get remote branch

Deleting Remote Branches (Example… quirky)

git push origin :v1.0\_fixes\_remote\_branch\_name -> Delete a remote branch

git push origin v1.0\_fixes:v1.0\_fixes\_remote\_branch\_name -> pushes updates from the local v1.0\_fixes branch to the remote v1.0\_fixes\_remote\_branch\_name on the server

git push origin :v1.0\_fixes\_remote\_branch\_name

git push origin :v1.0\_fixes

Merging Branches

git checkout master -> merges the feature1 branch with the HEAD

git merge feature1 -> merges the feature1 branch with the HEAD

git branch -d feature1 -> deletes the feature1 branch

git mergetool -> displays merge tool (kdiff3 – 3 way merge tool)

Remote

Display

git remote -> Will display (fetch url) and (push url)

git remote -v ->Display remote

Add

git remote add origin http://github.com/JamesKovacs/GitFundamentals.git

git remote add origin git@github.com:UserName/GitRepo.git -> auto apply SSH creds

Remove Origin

git remote rm origin -> Removes origin

git remote -v -> Shows that the origin no longer exists

Push

git push -> May be prompted for username and password unless use SSH creds

git push -- tags -> Push with tags

git push origin master -> push commit to the master branch

Tags (Versions/Labels) – Attach to a single Commit SHA1. Friendly Name of SHA1 Hash

git tag -> show tags/versions/labels

git tag -n -> show tags with message annotations

git tag v1.0 -> sets tag/version/label

git tag -a v1.0\_with\_message -> allows a message to be added via editor

git tag -a v1.0\_with\_message -m “Stable Release” -> allows a message to be added inline

git tag -s v1.0\_signed -> adds a signed tag

git push --tags -> This is how you push tags with commits

Aliases

git config --global alias.lga “log --graph --oneline --all --decorate”

git lga -> Executes the alias

cat ~/.gitconfig -> Displays the configuration file

Stash (Temporary holding area for changes not ready to commit to a branch)

git stash -> will rollback pending changes in the working copy and create a holding area

git stash list -> will display the holding area that contains the pending changes

git stash apply -> will pull the changes back to the working copy

(git reset –hard HEAD -> will toss out those changes brought back into the working copy)

git stash pop -> pops(removes) the top item from the stash and applies it to the working copy

git stash drop -> drops the reference to the top stash

git stash branch feature2\_additional -> creates a new branch &checks it out from a stash

Merge Tool

git mergetool

Rebase (Use this to move the master to the head of a branch and then merge the code)

***Makes for a cleaner history*** ( <http://git-scm.com/book/ch3-6.html>)

git checkout feature3 -> make feature3 the current code base

git rebase master -> take current branch and put on top of master

git checkout master -> make master the current code base

git merge feature3 -> merge feature3 with master

git rebase --continue -> continue a previous merge (usually needed after using mergetool)

Cherry Pick Updates (Apply a single commit, like a patch, to the current HEAD)

git cherry-pick 45b45b0 -> select one(1) single commit and apply it to the HEAD

Configuration

List Configuration Contents

git config --global --list

cat ~/.gitconfig

cat .git/config

Configuration Options

git config --global user.name ”John Doe”

git config --global user.email “john.doe@fmr.com”

git config --global core.editor notepad++.exe

git config --global core.editor "'C:\Program Files (x86)\Notepad++\notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

git config --global help.autocorrect 1

git config --global color.ui auto

git config --global core.autocrlf true (t->CRLF to LF, f->As-Is)

Undo Config Changes

Revert Changes to Previous Settings

git config --unset core.autocrlf

git config --unset user.name

**Configuration Types**

System-Level Configuration (Not Normally Modified)

* git config --system
* C:\Program Files(x86)\Git\etc\gitconfig

User-Level Configuration

* git config --global
* C:\Users\aNNNNNN\.gitconfig

Repository-Level Configuation

* git config -> Repository-Level Configuration (Each Repository .git/config)

***Configuration Settings***

Diff

Use KDIff3 (Look in Docs to see how to accomplish this)

Line Feed Options

core.autocrlf true -> Converts CRLF to LF (cross platform projects - recommended)

core.autocrlf false -> Checkout as-is, Commit as-is

Core Editor

core.editor notepad++.exe

Git Workflow – Atlassian (Need a Bitbucket Account)

There is a Project Maintainer -> Owns the Primary Repository (Server)

Fork Repository -> Server side clone creates a Personal Repository (Server)

Clone the Fork -> (git clone) Private Development Repository

Commit Locally -> (git commit -am “…”)

Publish Commits

File a Pull Request with the Main Repository

Push to Personal Repository -> (git push) (Server)

Maintainer Then…

Pulls contributor’s changes into their local repository and verifies code

Merges changes into his local master branch

Pushes the master branch to the official repository

Git Workflow – Steps

1. Make a copy of the Master Repository
   1. git clone https://???
2. Point to the Master Repository (Read Only, Needed to Pull Latest Data)
   1. git remote add upstream https://???
3. Pull from the Master Repository
   1. git pull upstream master
4. Push to Personal Remote
   1. git push origin feature-branch
5. Once code is pushed to the Master Repository, Pull the latest code
   1. git pull upstream master

Git Workflow – Get From Server

Pull

Make Chages

Commit Locally (multiple times)

Branch & Merge Locally (optional)

Git Workflow – Push to Server

Pull (Needed to resolve any conflicts)

Push