Git Console Commands

Updated: January 12, 2020

Legend:

- '$' indicates a command in git bash. Do not include them when writing the command.
- Blue writing indicates a command in PowerShell.
- Words surrounded by <> indicate that something needs to be inserted into that spot. Do not include <> when writing the command.

Resources:

- Git Command Line Tutorial: https://www.youtube.com/watch?v=HVsySz-h9r
- There are a variety of step-by-step examples of common git use cases in the examples section.

Background:

Your local repository consists of 3 "trees" maintained by git:

1. Working Directory (holds the working files)
2. Index (staging area for files before a commit is published)
3. HEAD (most recent published commit)

These trees are only local and committed files must also be pushed if using remote repositories. It's generally not good practice to commit many files at the same time with a generic message for all files. Instead, staging specific files for commit with their own individual messages is recommended.

Cloning Remote Repositories:

```
$ git clone <git-link> <desired file path>      // Clones an existing remote repository to local machine
$ git clone <git-link> .                        // Clones an existing remote repository to current path
```

Setup:

```
$ git init                                      // Initializes local git repository
```

Staging:

```
$ git add <fileName>                            // Adds file(s) to index/staging area
$ git add .                                     // Adds all files to index/staging area
$ git add *.html                                // Adds all .html files to index/staging area
$ git reset <filename>                          // Removes file from index/staging area
$ git reset                                     // Removes all files from index/staging area
```

Commits:

```
$ git commit                                    // Prompts for detailed description and then commits
$ git commit -m '<comment>'                     // Commits with <comment> as commit title
$ git log                                       // Shows a list of commits w/ unique hash numbers
```

Status:

```
$ git status                                    // Check status of working tree
```

Updating Remote Repositories:

When working with others, **always pull** first before pushing your latest commit. When using branches, the $ git push -u command associates the local with the remote branch, so that in the future we can just use the $ git push and $ git pull commands without indicating the branch name.

```
$ git diff                                  // Shows changes from the last commit
$ git pull origin <branchName >             // Pulls latest from specific branch of remote repository
$ git pull                                  // Pulls from the current branch of remote repository
$ git push origin <branchName>              // Push to specific branch of remote repository
$ git push -u origin <branchName>           // The -u allows us to associate local w/ remote branches
$ git push                                  // Push to the current branch of remote repository
```

Branches:

```
$ git branch                                // Lists all the local branches, including current
$ git branch -a                             // Lists all the local and remote branches
$ git branch <branchName>                   // Creates new branch
$ git checkout <branchName>                 // Changes working branch
$ git branch -d <branchName>                // Deletes branch locally
$ git push origin --delete <branchName>     // Deletes branch on remote repository
```

Merges:

Before merging back to master, always remember to $ git checkout master, then $ git pull origin master to sync any changes that may have been made while we were working on the branch.

```
$ git branch --merged                       // Shows the branches that have been merged
$ git merge <branchName>                    // Merges another branch into active branch
$ git add <fileName>                        // After editing conflicted files, mark files as merged
$ git diff <sourceBranch> <targetBranch>    // Preview changes before merging
```

Cloning Remote Repositories:

```
$ git clone <git-link> <where to clone>     // Clones an existing remote repository to local machine
$ git clone <git-link> .                    // Clones an existing remote repository to current path
```

Configuring Remote Repositories:

```
$ git remote -v                             // Lists remote repositories
$ git remote add origin <git-link>          // Links remote repository with local repository
$ git remote remove origin                  // Removes the remote repository
```

Console Navigation:

```
$ ls                                        // Lists items in current directory
ls -Force                                   // PowerShell: Lists all items (including hidden)
$ cd                                        // Changes directory
```

Git Ignore:

Git ignore files are used to ensure that only necessary files are being uploaded to github. Without them, you are likely to be bloating your git project with local or temp files. Standard gitignore files for various contexts can be found on the internet. For example, Unity (game engine) has a standard gitignore file to ensure only essential files are being uploaded to github. If there is a standard, download a copy and include it in your project folder. Otherwise, you can make your own:

```
$ touch .gitignore                  // Creates a blank .gitignore file
new-item .gitignore                 // PowerShell: Creates a blank .gitignore file
```

Now, open the .gitignore file and name items to be ignored in the one of the following formats:

- filename.txt
- /foldername or foldername/
- *.txt (all text files)

Remove Git Tracking:

```
$ rm -rf git                        // Removes the .git folder (ending version control tracking)
                                    // You can also just delete the .git hidden folder for the same result
```

Other:

```
$ git --version                     // Indicates current version of git
$ touch <filename.filetype>         // Creates blank file of specified type
new-item <filename.filetype>        // PowerShell: Creates a blank file of specified type
$ git config --global user.name "<Name>"        // Stores username that will show in central repository
$ git config --global user.email "<Email>"      // Stores email that will show in central repository
$ git config --global credential.helper store   // Stores git login so you do not need to sign in for every commit
$ git config --list                 // Shows the git config details
$ git help <verb>                   // Shows help for the specified git action
```

Git Console Examples

Connect Project to Github Repository Example:

1. Have pre-existing files to be backed on github
2. Add a ".gitignore" file if necessary
3. Navigate to project folder containing all relevant files using git bash or equivalent terminal
4. $ git init
5. $ git remote add origin <git-link>
6. $ git add **.** // Note the period after "add" to indicate to add all files
7. $ git commit -m "Initial commit"
8. $ git push origin master

Clone Git Project From Github Repository Example:

1. Have clone link of github repository
2. Navigate to folder where files are to be cloned using git bash or equivalent terminal
3. $ git clone <git-link>

Clone Git Project From Github Repository Alternative Example:

1. Have clone link of github repository
2. Navigate to folder where files are to be cloned using git bash or equivalent terminal
3. $ git init // Initiate git project
4. $ git remote add origin <git-link> // Add remote to git project (connect local to online github repo)
5. $ git pull origin master // Pull content of online repo to local folder

Commit Substantial Changes to Project Example:

1. Complete desired changes in existing git project
2. Navigate to project folder using git bash or equivalent terminal
3. $ git add **.** // Add files to staging area (index)
4. $ git commit
5. If unable to write, press insert to enter "insert mode"
6. Write descriptive title for the commit on the first line
7. Two lines down from the title (leave a line of whitespace between), write a detailed description for the commit
8. Press escape and enter ":wq" to leave insert mode and accept changes
9. $ git push // Push local changes to remote

Append Minor Changes to Previous Commit Example:

1. Complete desired changes in existing git project
2. Navigate to project folder using git bash or equivalent terminal
3. $ git add **.** // Add files to staging area (index)
4. $ git commit --amend // Append changes to previous commit
5. Edit commit message if necessary (see steps 5-7 above)
6. Press escape and enter ":wq" to leave insert mode and accept changes
7. **If working with others, pull all recent changes before force pushing or else you will overwrite their work**

1. $ git push -f // Force push local changes to remote. Push must be forced because you
   // locally changed the content of a commit that currently exists in the
   // remote. The discontinuity prevents a push unless forced.

Create/Merge Branch Example:

Note:

- (b-name → branch name)
- Useful command: "$ git branch -a" // Lists all branches (* indicates the current working branch)
  // Can be used to double check working branch or if branch was
  // created/deleted successfully

1. Navigate to master branch.
2. $ git branch <b-name> // Creates new branch
3. $ git checkout <b-name> // Changes local working directory to named branch
4. Make changes to files as usual
5. Commit changes locally as usual
6. $ git push -u origin <b-name> // Pushes and associates local branch with remote branch
   // After this command is used for the first time, can use
   // "$ git push" and "$ git pull" for future changes to branch
7. $ git checkout master // Changes local working directory back to master branch
8. $ git pull origin master // Pulls changes from remote master branch (if any)
9. $ git merge <b-name> // Merges branch into local master
10. $ git push origin master // Pushes changes to master remote repository
11. $ git branch -d <b-name> // Deletes local branch
12. $ git push origin --delete <b-name> // Deletes remote branch

Git Interactive Rebase Example: (Change commit tree order)

2. $ git rebase -i <b-name>~4 // Interactive rebase going back 4 commits on the named branch
3. Manipulate tree (If unable to write, press insert to enter insert mode)
   a. Reorder lines around to reorder commits
   b. Delete lines to delete a commit
   c. Change commands at the front of lines for desired effect (see command list)

```
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
```

4. Press escape and enter ":wq" to leave insert mode and accept changes
5. If the rebase is in progress and does not continue on its own, type "git rebase --continue". If you're not sure if the rebase is still in progress, type "git status" for details.
6. Rename changed commits/comments if necessary (press insert, edit comments, press escape + ":wq" + enter)
7. $ git push -f // Force push local changes to remote. Push must be forced because you
   // locally changed the content of a commit that currently exists in the
   // remote. The discontinuity prevents a push unless forced.

Storing Credentials Example (unverified):

1. $ git config --global user.name "<Name>" // Stores username that will show in central repository
2. $ git config --global user.email "<Email>" // Stores email that will show in central repository
3. $ git config --global credential.helper store // Stores git login used to commit to central repository
   // Note: must have already logged in after committing