Git Commands

Git Commands.

git init

This command turns a directory into an empty Git repository. This is the first step in creating a repository. After running git init, adding and committing files/directories is possible.

```
$ git init
```

git add

Adds files in the to the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area). There are a few different ways to use git add, by adding entire directories, specific files, or all unstaged files.

```
# To add all files not staged:

$ git add .

# To stage a specific file:

$ git add index.html

# To stage an entire directory:

$ git add css
```

git commit

Record the changes made to the files to a local repository. For easy reference, each commit has a unique ID.

```
# Adding a commit with message

$ git commit -m "Commit message in quotes"
```

git status

This command returns the current state of the repository.

```
$ git status
```

git config

With Git, there are many configurations and settings possible. git config is how to assign these settings. Two important settings are user user.name and user.email. These values set what email address and name commits will be from on a local computer. With git config, a --global flag is used to write the settings to all repositories on a computer. Without a --global flag settings will only apply to the current repository that you are currently in.

```
# Running git config globally

$ git config --global user.email "my@emailaddress.com"
$ git config --global user.name "Brian Kerr"


# Running git config on the current repository settings

$ git config user.email "my@emailaddress.com"

$ git config user.name "Brian Kerr"
```

git branch

To determine what branch the local repository is on, add a new branch, or delete a branch.

```
# Create a new branch

$ git branch <branch_name>

# List all remote or local branches

$ git branch -a

# Delete a branch

$ git branch -d <branch_name>
```

git checkout

To start working in a different branch, use git checkout to switch branches.

```
# Checkout an existing branch

$ git checkout <branch_name>

# Checkout and create a new branch with that name

$ git checkout -b <new_branch>
```

git merge

Integrate branches together. git merge combines the changes from one branch to another branch. For example, merge the changes made in a staging branch into the stable branch.

```
# Merge changes into current branch

$ git merge <branch_name>
```

git remote

To connect a local repository with a remote repository. A remote repository can have a name set to avoid having to remember the URL of the repository.

```
# Add remote repository

$ git remote <command> <remote_name> <remote_URL>
$ git remote add origin git@remoteurl:/acccount_name/repository_name.git
# List named remote repositories
$ git remote -v
```

git clone

To create a local working copy of an existing remote repository, use git clone to copy and download the repository to a computer. Cloning is the equivalent of git init when working with a remote repository. Git will create a directory locally with all files and repository history.

```
$ git clone <remote_URL>
```

git pull

To get the latest version of a repository run git pull. This pulls the changes from the remote repository to the local computer.

```
$ git pull <branch_name> <remote_URL/remote_name>
```

git push

Sends local commits to the remote repository. git push requires two parameters: the remote repository and the branch that the push is for.

```
$ git push <remote_URL/remote_name> <branch>
# Push all local branches to remote repository
$ git push —all
```

git log

To show the chronological commit history for a repository. This helps give context and history for a repository. git log is available immediately on a recently cloned repository to see history.

```
# Show entire git log
$ git log
# Show git log with date pameters
$ git log --<after/before/since/until>=<date>
# Show git log based on commit author
$ git log --<author>="Author Name"
```

git rm

Remove files or directories from the working index (staging area). With git rm, there are two options to keep in mind: force and cached. Running the command with force deletes the file. The cached command removes the file from the working index. When removing an entire directory, a recursive command is necessary.

```
# To remove a file from the working index (cached):

$ git rm --cached <file name>

# To delete a file (force):

$ git rm -f <file name>

# To remove an entire directory from the working index (cached):

$ git rm -r --cached <directory name>

# To delete an entire directory (force):

$ git rm -r -f <file name>
```