**Working with local repositories**

**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.

Usage:

# change directory to codebase

$ cd /file/path/to/code

# make directory a git repository

$ git init

In Practice:

# change directory to codebase

$ cd /Users/computer-name/Documents/website

# make directory a git repository

$ git init

Initialized empty Git repository in /Users/computer-name/Documents/website/.git/

**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.

Usage:

$ git add <file or directory name>

In Practice:

# 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.

It’s best practice to include a message with each commit explaining the changes made in a commit. Adding a commit message helps to find a particular change or understanding the changes.

Usage:

# Adding a commit with message

$ git commit -m "Commit message in quotes"

In Practice:

$ git commit -m "My first commit message"

[SecretTesting 0254c3d] My first commit message

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 homepage/index.html

**git status**

This command returns the current state of the repository.

*git status* will return the current working branch. If a file is in the staging area, but not committed, it shows with *git status*. Or, if there are no changes it’ll return *nothing to commit, working directory clean.*

Usage:

$ git status

In Practice:

# Message when files have not been staged (git add)

$ git status

On branch SecretTesting

Untracked files:

(use "git add <file>..." to include in what will be committed)

homepage/index.html

# Message when files have been not been committed (git commit)

$ git status

On branch SecretTesting

Your branch is up-to-date with 'origin/SecretTesting'.

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: homepage/index.html

# Message when all files have been staged and committed

$ git status

On branch SecretTesting

nothing to commit, working directory clean

**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.

There are many other variables available to edit in *git config*. From editing color outputs to changing the behavior of *git status*. Learn about *git config* settings in the official [Git documentation](https://git-scm.com/docs/git-config).

Usage:

$ git config <setting> <command>

In Practice:

# 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.

Usage:

# 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>

In Practice:

# Create a new branch

$ git branch new\_feature

# List branches

$ git branch -a

\* SecretTesting

new\_feature

remotes/origin/stable

remotes/origin/staging

remotes/origin/master -> origin/SecretTesting

# Delete a branch

$ git branch -d new\_feature

Deleted branch new\_feature (was 0254c3d).

**git checkout**

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

Usage:

# Checkout an existing branch

$ git checkout <branch\_name>

# Checkout and create a new branch with that name

$ git checkout -b <new\_branch>

In Practice:

# Switching to branch 'new\_feature'

$ git checkout new\_feature

Switched to branch 'new\_feature'

# Creating and switching to branch 'staging'

$ git checkout -b staging

Switched to a new branch 'staging'

**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.

Usage:

# Merge changes into current branch

$ git merge <branch\_name>

In Practice:

# Merge changes into current branch

$ git merge new\_feature

Updating 0254c3d..4c0f37c

Fast-forward

homepage/index.html | 297 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1 file changed, 297 insertions(+)

create mode 100644 homepage/index.html

**Working with remote repositories**

**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.

Usage:

# Add remote repository

$ git remote <command> <remote\_name> <remote\_URL>

# List named remote repositories

$ git remote -v

In Practice:

# Adding a remote repository with the name of beanstalk

$ git remote add origin git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

# List named remote repositories

$ git remote -v

origin git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git (fetch)

origin git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git (push)

*Note: A remote repository can have any name. It’s common practice to name the remote repository ‘origin’.*

**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.

Usage:

$ git clone <remote\_URL>

In Practice:

$ git clone git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

Cloning into 'repository\_name'...

remote: Counting objects: 5, done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 5 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (5/5), 3.08 KiB | 0 bytes/s, done.

Checking connectivity... done.

**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.

Usage:

$ git pull <branch\_name> <remote\_URL/remote\_name>

In Practice:

# Pull from named remote

$ git pull origin staging

From account\_name.git.beanstalkapp.com:/account\_name/repository\_name

\* branch staging -> FETCH\_HEAD

\* [new branch] staging -> origin/staging

Already up-to-date.

# Pull from URL (not frequently used)

$ git pull git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git staging

From account\_name.git.beanstalkapp.com:/account\_name/repository\_name

\* branch staging -> FETCH\_HEAD

\* [new branch] staging -> origin/staging

Already up-to-date.

**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.

Usage:

$ git push <remote\_URL/remote\_name> <branch>

# Push all local branches to remote repository

$ git push —all

In Practice:

# Push a specific branch to a remote with named remote

$ git push origin staging

Counting objects: 5, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (5/5), 734 bytes | 0 bytes/s, done.

Total 5 (delta 2), reused 0 (delta 0)

To git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

ad189cb..0254c3d SecretTesting -> SecretTesting

# Push all local branches to remote repository

$ git push --all

Counting objects: 4, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 373 bytes | 0 bytes/s, done.

Total 4 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

To git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

0d56917..948ac97 master -> master

ad189cb..0254c3d SecretTesting -> SecretTesting

**Advanced Git Commands**

**git stash**

To save changes made when they’re not in a state to commit them to a repository. This will store the work and give a clean working directory. For instance, when working on a new feature that’s not complete, but an urgent bug needs attention.

Usage:

# Store current work with untracked files

$ git stash -u

# Bring stashed work back to the working directory

$ git stash pop

In Practice:

# Store current work

$ git stash -u

Saved working directory and index state WIP on SecretTesting: 4c0f37c Adding new file to branch

HEAD is now at 4c0f37c Adding new file to branch

# Bring stashed work back to the working directory

$ git stash pop

On branch SecretTesting

Your branch and 'origin/SecretTesting' have diverged,

and have 1 and 1 different commit each, respectively.

(use "git pull" to merge the remote branch into yours)

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

Dropped refs/stash@{0} (3561897724c1f448ae001edf3ef57415778755ec)

**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.

Usage:

# 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"

In Practice:

# Show entire git log

$ git log

commit 4c0f37c711623d20fc60b9cbcf393d515945952f

Author: Brian Kerr <my@emailaddress.com>

Date: Tue Oct 25 17:46:11 2016 -0500

Updating the wording of the homepage footer

commit 0254c3da3add4ebe9d7e1f2e76f015a209e1ef67

Author: Ashley Harpp <my@emailaddress.com>

Date: Wed Oct 19 16:27:27 2016 -0500

My first commit message

# Show git log with date pameters

$ git log --before="Oct 20"

commit 0254c3da3add4ebe9d7e1f2e76f015a209e1ef67

Author: Ashley Harpp <my@emailaddress.com>

Date: Wed Oct 19 16:27:27 2016 -0500

My first commit message

# Show git log based on commit author

$ git log --author="Brian Kerr"

commit 4c0f37c711623d20fc60b9cbcf393d515945952f

Author: Brian Kerr <my@emailaddress.com>

Date: Tue Oct 25 17:46:11 2016 -0500

Updating the wording of the homepage footer

**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.

Usage:

# 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>

In Practice:

# To remove a file from the working index:

$ git rm --cached css/style.css

rm 'css/style.css'

# To delete a file (force):

$ git rm -f css/style.css

rm 'css/style.css'

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

$ git rm -r --cached css/

rm 'css/style.css'

rm 'css/style.min.css'

# To delete an entire directory (force):

$ git rm -r -f css/

rm 'css/style.css'

rm 'css/style.min.css'