**git Docs**

<http://guides.beanstalkapp.com/version-control/common-git-commands.html>

**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.

**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.

**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.

**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.*

**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).

**git branch**

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

**git checkout**

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

**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.

**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.

**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 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 push**

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

**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.

**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.

**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.