Git Demo Notes

Requirements

- Unix like environment
- Have `git` installed
- Have `man` installed
- Have `gcc` installed
- Unix, Linux BSD, etc
  - Use your package manager to install
  - Ex. Debian/Ubuntu derivatives: `sudo apt install <x>`
- MacOSX terminal
  - homebrew: https://brew.sh
  - Ex. `brew install <x>`
- Windows
  - Windows Subsystem for Linux (WSL) *Preferred*
    - Activate Windows features:
      - Windows Subsystem for Linux
      - Virtual Machine Platform
    - Distributions available in the Windows Store
    - Ubuntu (easiest, reqs pre-installed)
  - Cygwin
  - SCM-Git

Environment Sidebar

- `~` is shorthand for your home folder.

Start A Project

Create a demo structure

- `mkdir remote-host`
- `mkdir workstation1`
- `mkdir workstation2`

Create a Hello World project in Workstation1

- `cd workstation1`
- `mkdir hello`
- `cd hello`
- `vim hello.c`
  - Add version 1 (or your something of your own choosing)

- `touch branch1`

`vim` Sidebar

- Movement: h - Left, j - Down, k - Up, l - Right
- i - insert mode
- ESC - go back to movement mode
- To save and quit: Press ESC and then Shift-Colon and type `wq` (write and quit)
  - `:wq`

Version 1

```
#include "stdio.h"

int main(int argc, char** argv){

   printf("hello world\n");
}
```

Version 2

```
#include "stdio.h"

int main(int argc, char** argv){

   if(argc > 1){
      printf("%s\n", argv[1]);
   } else {
      printf("No input\n");
   }
}
```

Setup Project with git

`git init .`

Add .gitignore for *.out

`git status`

`git add/remove`

- Concept of staging
- `git add -A`

`git commit`

- `git commit -a -m`

- `git commit -m`

Create a Remote Repository

`git init --bare ../remote-host/hello.git`

`git remote add <remote name> <url>`

- `git remote add origin ../remote-host/hello.git`

`git push --set-upstream <remote name> <branch>`

- First push: `git push --set-upstream origin master`
- Subsequent pushes: `git push`

`git clone`

- `cd ../workstation2`
- `git clone ../remote-host/hello.git`
- `ls`
  - You should see a a copy of your project in a new folder named: `hello`

`git pull`

- Make changes in `workstation1/hello` and then follow the add-commit-push flow.
- Come back to `workstation2/hello` and run
  - `git pull`
- You should see the changes applied to your *local* copy!
- If you had made local changes, you may have to tell `git` how to handle merge conflicts:
  - `git --config pull.rebase true`

So, now we have 3 copies of the project:

```
     Remote: remote-host/hello.git
                  |
                __/\__
               /      \
               |      |
workstation1/hello   workstation2/hello
```

Either one can push changes to the up-stream remote, if they have it set up.

SSH Integration

Git understands the SSH protocol. If you have your remote on a remote machine, you can set it up to use an off-site repository, just as easily.

Here is an example using a Raspberry Pi on a home network, with `Host pi` configured in `~/.ssh/config`. On the Pi, create a new `--bare` repository just like before. Navigate back to `workstation1` and add it as a remote:

- `git remote add pi-server user@pi:~/repos/hello.git`
- `git push pi-server`

It really is that easy.

.git/config

When you ran `git init .`, `git` added a hidden folder to your project: `.git`. This is where it stores the project configuration, and the compressed archive of previous versions. You can edit the configuration manually, which may actually be easier for some than using the "porcelain" commands.

If we did everything correctly, then our `.git/config` should look something like this:

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = ../../remote-host/hello-repo.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
[remote "pi-server"]
        url = user@pi:~/repos/hello.git
        fetch = +refs/heads/*:refs/remotes/pi-server/*
```

Some hosting services (GitHub/Bit Bucket) require the following format:

```
[remote "origin"]
    url = https://<username>:<access token>@<host domain>/PATH/TO/hello.git
    fetch = +refs/heads/*:refs/remotes/origin/*
```

Branching

One of the purposes of branching, is to provide a mechanism where active development will not interfere with a known stable release. Another, could be to work on an experimental feature that may not make it's way into the final product, or for individual work before merging into the larger project.

Creating a branch is easy:

```
git branch <name>

git switch <branch name>
```

When you push from a branch it is mirrored in that branch on the remote host. It's just that easy. And once you are ready to integrate your changes, to the main branch, you can just:

```
git switch master

git merge <branch>

git push
```

In many projects, especially open-source, there is an extra step. You would be doing all this with a "fork", which is essentially a clone of the project repository on the remote host for your personal use. Once you have a change that you think the project should incorporate, you can open a "Pull Request". If the Project Owner likes your change, they will run a `pull` against your fork to bring the changes into the official repository.

Hooks

One of the features that makes `git` really nice, is that you can trigger system actions in response to events that happen to your repository. Say for example, you are hosting a local Test webserver for your website on your home network. You make changes to your local working copy, and then push to your Test repository: `user@pi:~/repos/homestead.git`

On the Pi, navigate to `~/repos/homestead.git/hooks`. Any thing that is not a ".sample" file, is a live system script file that will be run when that event occurs.

So, for instance, when your repository receives a push, it could have a `post-receive` hook so that some action will occur whenever it receives a push:

```
#!/bin/sh
git --work-tree=/var/www/html --git-dir=/home/user/repos/homestead.git checkout -f master
```

This script checks out a copy of the repository into the webserver's directory.

Updating your website is now just as easy as typing: `git push test-server`

Conclusion

So, that is `git` in a nutshell. You should know how to create a create a repository, initiate a local project, commit, push and pull. Branching and hooks are both very nice features, too, and hopefully you can make use of them.

I hope this has been helpful and that you will be able to use it to be more productive. Thank you for joining me in this exploration.