Git Manual

This manual should help get you up to speed on how to use Git at Student Life Technology. If you are new to Git, it is highly recommended that you take 15 minutes right now and do the official Git tutorial **here** (https://try.github.io/levels/1/challenges/1).

Basic Concepts

A *repository* (or *repo* for short) is sort of like a storage space for a single project. Each web site at SLT will have its own separate repository. All the repositories are stored on the same server. (TODO: add server name here)

We have some servers intended for code development, which are accessible only from within the office (e.g. `pegleg`) and some which are accessible to everyone. (e.g. `thunderbolt`) These are called `production` and `development` servers respectively. Our central repository lives on neither kind of server.

The Git repository acts as the definitive definition of the source code. If you change something directly on one of the servers, you have just committed a **GRIEVOUS SIN**!! When we need to add/modify code, we clone the site onto our desktop machine, make the changes, commit our changes, then push from our desktop to the server.

Desktop Setup

This is how to configure your desktop computer so everything works smoothly:

1. Make sure you have Git installed.
2. Make a place for your repos to live.
   Use a folder called `Projects/` on your desktop machine.
3. Setup user name and email.
   Do this so we know who committed what. You can set the user name and password like this: (you don't have to be in a particular folder)

   ```
   $ git config --global user.name "Your Name"
   $ git config --global user.email "youremail@provider.com"
   ```

You should only ever have to do that once. Once you're setup, do the following to get setup with a web site:

1. Clone the site.
   Run `git clone programmer@sltrepo.byu.edu:repos/new_site/`
2. Create your own branch.
   Do this to keep your changes isolated from everyone else's changes. Give it a descriptive name, starting with your name:

   ```
   $ git branch ashton_new_instructions
   $ git checkout ashton_new_instructions
   ```

Happy hacking!

Practices

- At the end of every day, **COMMIT AND PUSH YOUR CODE!!**
  We don't have backups of the desktop machines.
- Commit often.
  Even if all you've done is added a few lines of code, commit. You should be committing code multiple times a day. You don't necessarily have to push.
- Keep branch `master` clean.
  Don't merge into `master` unless you have had your code reviewed, and are sure it is bug-free. We want to keep `master` deployable at all times.

Creating a New Web site

1. Go to the central repository.
   Pretend our repository is stored under the home directory of `sltrepo.byu.edu`.

   ```
   $ ssh programmer@sltrepo.byu.edu
   $ cd repos/
   ```

2. Create a new folder.

   ```
   $ mkdir my_new_site
   $ cd my_new_site/
   ```

3. Run `git init --bare` to build the site's repository.

   ```
   $ git init --bare
   ```

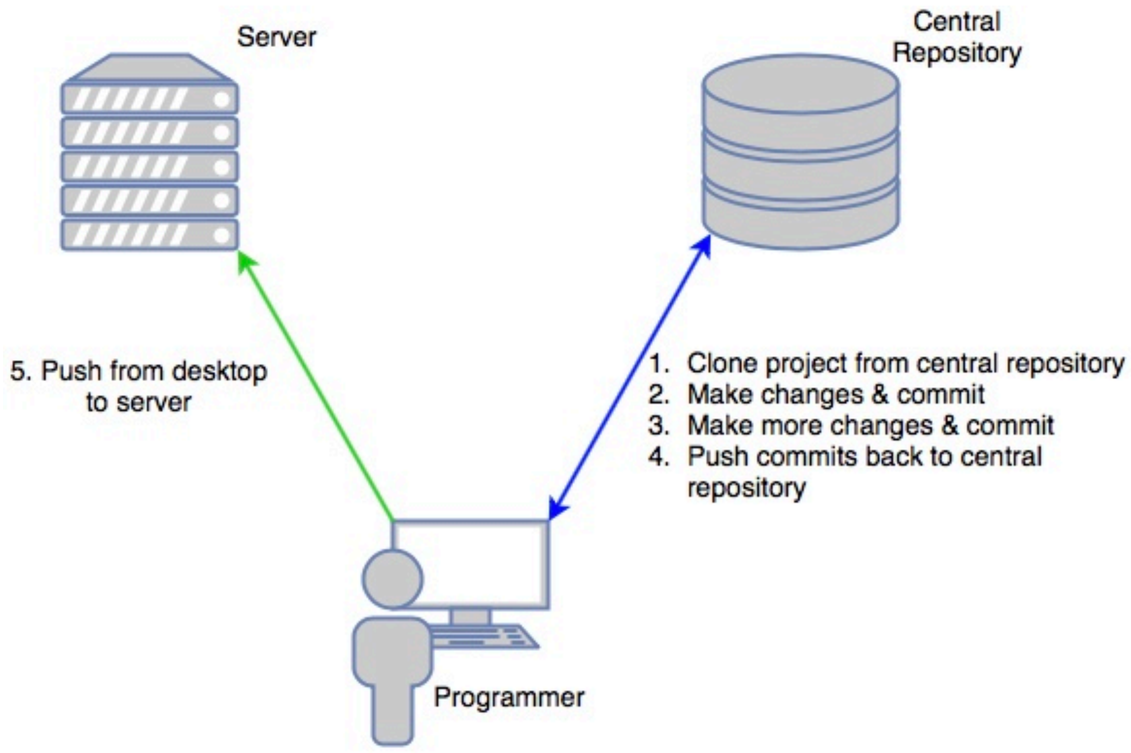
4. Go to your desktop and clone.

   ```
   $ cd Projects/
   $ git clone programmer@sltrepo.byu.edu:repos/my_new_site/
   ```

5. Make a build script. (optional)
   This makes it convenient to push code. If you don't have a build script, you will have to use Filezilla or `rsync` to copy files to the server. (Note: if you're using `rsync`, you're already half way to having a simple build script.)

Workflow

Solo Development/Fixing an old project


Server
Central Repository
5. Push from desktop to server
1. Clone project from central repository
2. Make changes & commit
3. Make more changes & commit
4. Push commits back to central repository
Programmer

1. Clone the source from the central repository.

   ```
   $ git clone programmer@sltrepo.byu.edu:repos/my_site
   ```

2. Make changes and commit.

   ```
   $ emacs index.php
   (...)
   $ git add index.php
   ```

```
$ git commit -m "changed some stuff in index.php"
```

3. Make more changes and commit.

```
$ emacs index.php
(...)
$ emacs foo.html
(...)
$ git add index.php foo.html
$ git commit -m "made index.php pull in cool stuff from foo.html"
```

4. Push changes to central repository.

```
$ git pull
$ git merge

(fix merge conflicts, if any)

$ git push
```

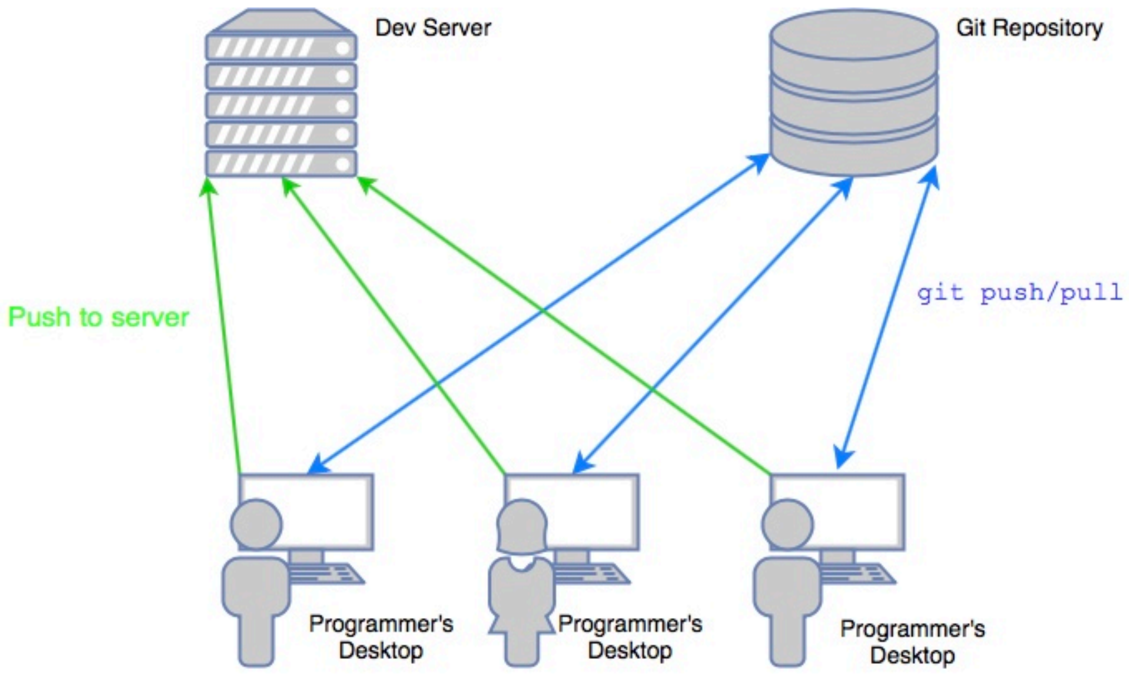
5. Push from desktop to server.

```
$ make deploy
```

or something like:

```
$ rsync -aivz --exclude=.git . programmer@ayeaye.byu.edu:/var/www/my_site/
```

Collaboration



Each programmer should have their own development branch for active development. Make sure your name is in the branch so we know who is working on what. For example, if Arthur and Ashton were to be working on the same project, they would do this:

(Ashton's side)

```
$ git clone programmer@sltrepo.byu.edu:repos/supplytracker/
  Cloning into 'supplytracker'...
  remote: Counting objects: 1629, done.
  remote: Compressing objects: 100% (1347/1347), done.
  remote: Total 1629 (delta 956), reused 631 (delta 229)
  Receiving objects: 100% (1629/1629), 1.68 MiB | 0 bytes/s, done.
  Resolving deltas: 100% (956/956), done.
$ cd supplytracker/
$ git branch ashton_new_feature
$ git checkout ashton_new_feature
  Switched to branch 'ashton_new_feature'
```

(Arthur's side)

```
$ git clone programmer@sltrepo.byu.edu:repos/supplytracker/
  Cloning into 'supplytracker'...
  remote: Counting objects: 1629, done.
  remote: Compressing objects: 100% (1347/1347), done.
  remote: Total 1629 (delta 956), reused 631 (delta 229)
  Receiving objects: 100% (1629/1629), 1.68 MiB | 0 bytes/s, done.
  Resolving deltas: 100% (956/956), done.
$ cd supplytracker/
$ git branch arthur_some_other_feature
$ git checkout arthur_some_other_feature
  Switched to branch 'arthur_some_other_feature'
```

Okay, now Ashton and Arthur each have their own branch. Now, when Ashton changes something, he can commit and push his branch to the server. Arthur can do the same.

When Arthur has finished a particular feature that Ashton wants to work with, Ashton can do something like this from *his* branch:

(Ashton's side)

```
$ git pull arthur_some_other_feature
$ git merge arthur_some_other_feature
```

At the end of the day, Arthur and Ashton should commit and push their changes:

```
$ git add -A
$ git commit -m "notes for tomorrow"
$ git push
```

These will push changes to the programmer's personal branches. As soon as they are ready to pass off a feature, they should review each other's code, then merge with `master` and push:

```
$ git checkout master
$ git merge arthur_some_other_feature
$ git push
```

Build scripts and deploying code

Use Filezilla to copy your project from your desktop to the server. Alternatively, you can use build scripts to automate this process.

A build script can make deploying code to a server very convenient. Here's an example build script for the supply tracker/print jobs site, stored in `Makefile`:

```
## Makefile
## Ashton Wiersdorf
## Started: Fri Jan  6 16:07:52 MST 2017

ifeq ($(MODE),production)
SERVER = thunderbolt.byu.edu
else
SERVER = ayeaye.byu.edu
```

```
endif

USER        = programmer
PWD         = /var/www/supplytracker/laravel/
LOCAL_PWD    = $(HOME)/www
EXCLUDE      = .git/ *.sqlite vendor storage .env artisan public/.htaccess database/database.db
RSYNC_OPTIONS = $(addprefix --exclude=, $(EXCLUDE)) --delete --no-p --no-t


dry-deploy:       # pretends to upload the project
   rsync -aivz --dry-run --exclude=*~ $(RSYNC_OPTIONS) . $(USER)@$(SERVER):$(PWD)

deploy:
   rsync -aivz --exclude=*~ $(RSYNC_OPTIONS) . $(USER)@$(SERVER):$(PWD)
   ssh $(USER)@$(SERVER) 'cd $(PWD); php artisan migrate:reset; php artisan migrate'
```

To deploy, you run `make deploy` in your shell, and `make` uses `rsync` to copy the files onto the server. This only copies files that have been changed. Use what you like; just make sure the code on the server matches the code on your desktop exactly. (Using `rsync` is highly recommended.)

It's also possible to use this build script to deploy to `thunderbolt`, instead of `ayeaye`. Instead of `$ make deploy`, run `$ MODE=production make deploy` and `$(SERVER)` will resolve to `thunderbolt.byu.edu` instead of `ayeaye.byu.edu`.

Further Reading

Here are some resources to help you with using Git:

- **Comparing Workflows** (https://www.atlassian.com/git/tutorials/comparing-workflows#feature-branch-workflow)
  We're using something like the "feature branch workflow". This website has some other helpful tutorials.
- If you're new to Git, there's a lovely tutorial on Git's main page **here** (https://try.github.io/levels/1/challenges/1).