Git+GitHub QuickStart for CSULB Friend Finder development

This paper is intended to get you started using Git, and may serve as a reference sheet throughout the semester.

Step 1: Download and install Git from http://git-scm.com/downloads

1. If you are running Mac, the installer will install the command-line utilities.
2. If you are running Linux, the above URL will show you how to download using your distro's package manager.
3. If you are running Windows, the "GitHub for Windows" installer will install a GUI version of Git, in addition to the *Git Shell*. I'll be showing you how to use the shell, since it will be similar to usage on Mac and Linux.

Step 2: Run the Git Shell and configure your profile

1. If you are running Mac, open Applications -> Terminal. In the terminal, you should be able to run git right away, since the installer adds the executable to your path. If not, ask Tan, etc.
    a. Then, run the following commands (2 dashes)

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

2. If you are running Windows, open the Git Shell from the Start Menu. Then, run the following commands (2 dashes)

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

3. If you are running Linux, open a terminal and run the same commands. As with Mac, the "git" command should run automatically as it is on your path. If not, ask Tan, etc.

Step 3: Download our Repo

Navigate to the folder you would like to store the csulb-ff folder in. For example, `cd Desktop`. When you clone, a folder will be created in Desktop called csulb-ff. Run the following commands:

```
git clone https://github.com/tktran/csulb-ff.git
cd csulb-ff
```

(The clone URL you will find on **https://github.com/tktran/csulb-ff**)

Step 4: Branch, make changes, and commit

After you have cloned csulb-ff, you will be in the master branch. To start your own development process, run:

```
git checkout –b myBranchName
```

This command creates a new **b**ranch, and checks it out for you to start developing. Make some changes to the files, manually or with whatever IDE we've decided to use, even create a new file if you need to.

Step 5: After developing, add everything and commit

Run the following commands, including the period in the first one (which stands for the current directory). Briefly, you add all files, old and new, commit to your myBranchName branch, go back to master branch, merge myBranch's changes with master, and upload master to back to the origin (which is the GitHub servers).

```
git add .
git commit –m "In this edit, I did blah blah blah keep it short"
git checkout master
git merge myBranchName
git push origin master
```

You're now back in master. To develop some more, checkout your branch again using `git checkout myBranchName`, or create a new branch with `git checkout –b myNewBranch`.

You'll definitely run into problems later on, involving merge conflicts etc. If you do, you can always ask Tan or send your files to him for him to merge, since that's his job as scrum master.

Here's another intro to git. It's interactive and really cool:
https://try.github.io/levels/1/challenges/10

GitHub

GIT CHEAT SHEET

V 1.1.1

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```

Creates a new local repository with the specified name

```
$ git clone [url]
```

Downloads a project and its entire version history

MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

```
$ git add [file]
```

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```

Records file snapshots permanently in version history

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

```
$ git branch [branch-name]
```

Creates a new branch

```
$ git checkout [branch-name]
```

Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```

Combines the specified branch’s history into the current branch

```
$ git branch -d [branch-name]
```

Deletes the specified branch

GIT CHEAT SHEET

REFACTOR FILENAMES
Relocate and remove versioned files

```
$ git rm [file]
```
Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```
Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```
Changes the file name and prepares it for commit

SUPPRESS TRACKING
Exclude temporary files and paths

```
*.log
build/
temp-*
```
A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```
Lists all ignored files in this project

SAVE FRAGMENTS
Shelve and restore incomplete changes

```
$ git stash
```
Temporarily stores all modified tracked files

```
$ git stash pop
```
Restores the most recently stashed files

```
$ git stash list
```
Lists all stashed changesets

```
$ git stash drop
```
Discards the most recently stashed changeset

REVIEW HISTORY
Browse and inspect the evolution of project files

```
$ git log
```
Lists version history for the current branch

```
$ git log --follow [file]
```
Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```
Shows content differences between two branches

```
$ git show [commit]
```
Outputs metadata and content changes of the specified commit

REDO COMMITS
Erase mistakes and craft replacement history

```
$ git reset [commit]
```
Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```
Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES
Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```
Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```
Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```
Uploads all local branch commits to GitHub

```
$ git pull
```
Downloads bookmark history and incorporates changes