Coding & Cocktails Session 7 Worksheet

Overview

During the session we'll discuss distributed version control, primarily through a tool called Git. This will be a very interactive session with lots of practice during the talk.

> ***If you work best with a paper copy please print this worksheet and the cheat sheet listed in Step 5 below before coming Saturday night!***

Prep Work

We strongly recommend that you do these steps prior to coming to the session.

1. Verify that you have a Github login and that Git is installed on your computer. If you do not have it, click on or type http://bit.ly/cnctools to be redirected to our instructions for downloading developer tools.

2. Open Git Bash (Windows) or iTerm2 (Macs). At the prompt ($ or >), type: `git config --global user.name "Your Name"` (use your actual name) to set your user name. Be sure to use the quotation marks around your name. Remember to press <enter> after every command to execute it.

3. At the next prompt, type: `git config --global user.email yourname@email.com` (use your actual email address) to set your email address. This is critical because you are setting up your login verification with GitHub. You will be asked for it every time you commit changes before pushing them up to Github. We'll explain what that means in detail Saturday night.

4. Finally, just to ensure that you have entered everything correctly, type `git config --list` to list all the settings Git can find at that point. Ensure that user.email and user.name are displayed in this list.

5. While there are many Git command cheat sheet sites, we found this one particularly useful: http://bit.ly/CnCGitCS. You may want to print this as well ahead of time and bring it with you for easier access during the session.

> **NOTE:** If you have problems at home, you can take a screen shot of your Git Bash or iTerm2 window, and share it in our Slack channel (our online chat channel) where one of our mentors can help or email it to codingandcocktails@kcwomenintech.org before the event and we can help you resolve it.

Stop here and save the rest for class!

Class Exercises

Part 1: Exploring the Interfaces

1. If you use Windows, open Git Bash, or if you use a Mac, open iTerm2, which is your command line tool. Note that it accepts typed commands we covered covered last month. For a refresher, check out the session here: http://bit.ly/cmdln. Try these:

    a. check your folder to see if you have a Coding & Cocktails folder set up. If not, use command `mkdir` to build the folder.

    b. `cd codingandcocktails`: changes the working directory to your Coding And Cocktails directory.

2. Open the desktop app either by typing `git gui` on the command line, or using your search directory to find it on your machine. Click **Create New Repository**. This will bring up the GUI (graphical user interface), or "gooey", which you may explore but we will not go into detail in class.

3. Go to Github.com and log into your account. We will talk about various features of the GitHub page together.

Part 2: Initialize a Repository

Note: these examples use root location c:\users\yourname\codingandocktails. You may choose your own location if you like.

1. On Github: From your profile page, click on the repositories tab and then click the green **New** button or in the upper right corner, click the dropdown menu next to the plus (+) and select **New Repository.**

2. Give your repository a name, like MyFirstRepo.

3. Give it a description, and click I**nitialize with a Readme file**.

4. Click **Create Repository**.

> ***TIP:*** *Unless you decide to buy a paid GitHub account (not necessary for what we do) all of your repositories are automatically publicly searchable. You can ignore the git.ignore and license buttons for the time being. You won't need them for what we do as part of Coding & Cocktails.*

Congratulations! Have a drink! You've established your presence on GitHub!

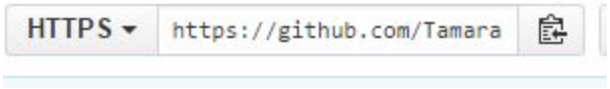
Part 3: Cloning Your Local Repository

Your newborn repository exists in The Cloud for now, but that's not where you will be writing your code. That happens on your own computer. From this point on, your computer is referred to as "local" and your GitHub repo is "remote."

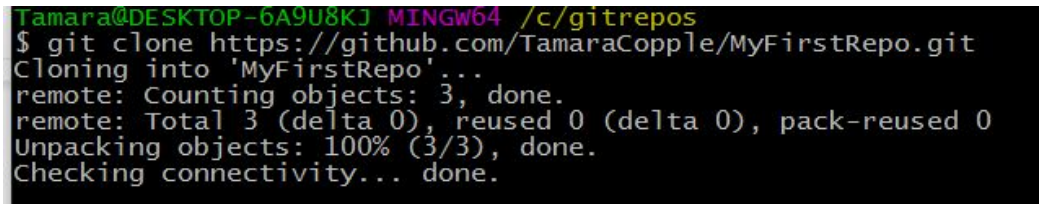
1. Open Git Bash or iTerm2.
2. If you are not already there, use command `cd` to change directories to the location where you want your repository to live.

> **TIP:** Do not use *mkdir* to make a new directory for the file you are about to clone - the cloning process does this for you.

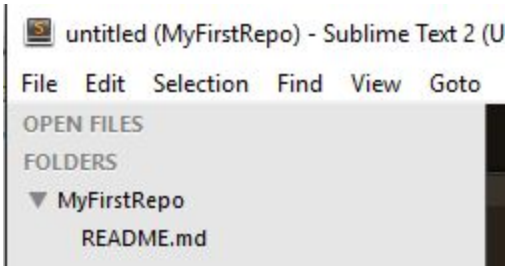
3. Back on GitHub, look for a URL assigned to your new repo. Click on the clipboard icon to the right of the URL. This copies it to your system clipboard. HTTPS is the recommended option, and this is fine for what we will do. SSH stands for Secure Shell, but we will not be using it.



4. Return to Git Bash or iTerm2. Type `git clone` and then paste in the URL of your repo. You should see it churn out something that looks like this:



5. Just to confirm, you can use your system's file manager (i.e. Windows Explorer or Mac Finder) to verify that it did indeed copy down the ReadMe file. You could also use the `ls` command to review the contents of the repo from your command line. Now you have a copy on Github, and a local copy that you can modify without affecting the original.
6. Next, we are going to make a simple change to the readme file, so that you can track the changes in the next section. Open Sublime Text. Select **File-->Open Folder** and drive to the repo you just cloned down to your machine. There should only be one file in it - your ReadMe.md file.



7. Open the README.md file and in Sublime Text, change something in the file, such as adding a sentence, or a list of the ladies sitting around you tonight. Save the changes.
8. Back in Git Bash or iTerm2, use `cd myfirstrepo` to change the directory to the new repo you just cloned.

Part 4: Staging, Committing, and Pushing to GitHub

There are three vital steps to pushing your changes up to GitHub: **staging, committing,** and **pushing.** The first step is to tell Git which files you want to add to the repository. This is called *staging* your changes. Then, you prepare to push them using the *commit* command. Finally, the committed changes are pushed up to the cloud and Git assigns a unique alpha-numeric log entry, known as a *commit hash*, to track that change. Let's try it now.

1. In Git Bash or iTerm2, type `git add README.md` . Alternatively, you could also type `git add --A`, which would add all files you have modified lately. To verify that your files were added, use `git status`, which will tell you all about your repo. the last line will show which files have been modified.

```
Tamara@DESKTOP-6A9U8KJ MINGW64 /c/gitrepos/myfirstrepo (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

2. Commit your changes with a brief message like this: `git commit -m "First Commit of the README.md file".` The -m message is always necessary and should accurately and concisely describe the content of the changes. This is important information for those who will be responsible for maintaining your code later. Your changes have been committed and are ready to load up to GitHub.

```
Tamara@DESKTOP-6A9U8KJ MINGW64 /c/gitrepos/myfirstrepo (master)
$ git commit -m "First Commit of the README.md file"
[master f187468] First Commit of the README.md file
 1 file changed, 1 insertion(+), 1 deletion(-)
```

3. Use the command `git push origin master` to initiate the push. "Origin" tells Git you are pushing the changes to the original remote location and "Master" indicates which branch within the remote location. This will be more important later.

4. At each prompt, enter your username and password. Your password will be masked so don't be surprised if it looks like you are not typing anything on that line.

5. Note that the last line assigns the commit hash for this first commit. Voila! You have pushed your first revision to GitHub!

Part 5: Practice with Forking

When **forking**, you are pulling a copy of someone else's repository into your own local environment. You can review their code, modify it, and test it locally. If you have a suggestion, you can then push your updates to their code into the "upstream" repository (your cloud copy of their repo) and issue a *pull request*. You are asking the owner of the original repository to review your code and incorporate it into their master branch. *The key is, the owner of the original repo that you forked from is the one who gets to decide when and if to merge your code with theirs.*

1. Point your browser to the Kansas City Women in Technology GitHub Page: https://github.com/KansasCityWomeninTechnology/LearningResources (or type http://bit.ly/kcwitLR if you printed this off)
2. Look in the upper right corner of the page, near your profile photo for the "fork" Icon. Click this link.
3. When asked, "Where should we fork this repository?" Select your own profile. If it does not show an option, don't worry! It will fork the repository to your own account.
4. Once the repository is copied to your profile, it will open up the page dedicated to your fork of the Learning Resources Repository.
5. Clone this repository down to your desktop. (*Hint: Review the steps for cloning a repository in Part 3).*

Once you have cloned it down to your desktop you can open this folder in Sublime text and add your own updates to any of the markdown documents. If you have a definition or a term you would like to see added, push the changes up to your forked repository (your *upstream).* Once your commit has been saved in the cloud, use the green "pull request" button to submit a request to have someone review your code for possible inclusion in the main repository. If you don't want to suggest a change, you can still treat it like any other project of yours and continue committing updates to your fork without ever pushing it back up to the KCWiT repository where it originated.

Take a break! You've earned it!

Part 6: Branching and the Art of Collaboration

At this point you should have two repositories listed in your GitHub account: the one you built for yourself, and the one you just forked. Now we are going to collaborate on a very simple web based application.

1. Point your browser to https://github.com/KansasCityWomeninTechnology/DrinkOrderApp. Note that in the **issues** tab there are a number of things we want to fix. You will be assigned just one of these issues to change.
2. Clone this repository down to your Coding & Cocktails folder. (*Hint: Review the steps for cloning a repository in Part 3).*

3. Since you will want to be working within this new repo you just cloned, change your directory to be your DrinkOrderApp directory.

4. Now we want to make a new branch of this repository. A *branch* is a copy of the repository that (for our purposes) focuses on one change or kind of change to the repository. In your command line (Git Bash or iTerm2) type `git checkout -b [newbranchname]` where "newbranchname" is the branch name for the issue we will assign to you in class. We suggest using issue-## as your branch name so if you were assigned issue #4 your branch name would be issue-04.

5. To list all branches within the repo, use `git branch .` This should show you at least two branches - master, and the one you just created. The branch you are working in will be highlighted with an asterisk (*).

> **TIP:** To switch between branches, use `git checkout [branchname]`.

6. Take a look at the folder where you created your Drink Order App copy. You'll see that there is still only one copy of the code in this folder. That's because the changes you make are being routed through Git using the command line functions. Thus, which branch you are in when you commit your changes makes all the difference!

7. Open SublimeText and navigate to your Drink App folder. Based on whichever issue you were assigned, make your change to the file and save your changes.

8. When the change is made, use `git status` to find out if your branch is up to date. If it is, add your changes to the branch, commit them with a message describing what you updated, and use `git push origin [branchname]`, substituting your own branch name. Now you have pushed your changes to the *origin* (that is, to the cloud), to your specific branch.

9. Next, you have to migrate your changes from the branch you have been working on to the *master* branch. To do this we create a **pull request.** Go to https://github.com/KansasCityWomeninTechnology/DrinkOrderApp and click on the green **New Pull Request** button.

10. You will be prompted to select the "base", which is where you are moving your changes TO, and the "compare", which is where you are moving them FROM. If Git detects no conflicts, it will give you a checkmarked message saying "able to merge." Below that will be your *commit* message. Scrolling down further you will be able to see the commit history that is being included in this request, and the actual changed code. At the bottom for your message box, click on **Create pull request**.

11. Typically we would expect someone to have oversight into merging the pull requests into the master branch to become the final version of the code. Once your pull request is accepted by this person, you can view your code in the master branch, along with all of the changes your peers have made.

Congratulations! You've collaborated on a project for the first time!

Epilogue - Resolving Merge Conflicts

Most of the time when you are working on your own branch, and your peers are each working on their own branches, merge conflicts may not happen as often, but eventually you will run into one. We are going to demonstrate how to resolve a merge conflict in class for the sake of time, but this is also a skill that you can expect to be trained on according to your team's procedures.

The Morning After. . .

Additional Resources

1. The Atlassian Git Tutorial: https://www.atlassian.com/git/
2. The Official Git Starting point: https://git-scm.com/
3. Linus Torvalds discusses what motivated him to develop Git - https://www.youtube.com/watch?v=4XpnKHJAok8
4. Remember - there are mentors online in our private chat room - kcwit.slack.com!

Homework

1. Think of one new tech-related term you have heard recently, and research its meaning. Update the markdown document in your repository, push it to your repository on GitHub, and issue a pull request to have your term and its definition added to our learning resources Glossary.
    a. Never used markdown before and need some help figuring it out? Check out these resources:
        i. https://help.github.com/articles/basic-writing-and-formatting-syntax/
        ii. https://guides.github.com/features/mastering-markdown/
        iii. http://markdowntutorial.com/
2. Take the Codecademy Introduction to Git: https://www.codecademy.com/learn/learn-git
3. Just for fun - Learning Github via a game! http://pcottle.github.io/learnGitBranching/
4. Sign up for next month's Coding & Cocktails session!