SIC

Thursday, September 28th 2017

Any New Members?

Charity

- **Casa De Peregrinos (Food Bank)**
  - **999 West Amador Suite F**
  - **Available MWF**
- **Duties:**
  - **Bagging, Sorting, Distributing, Cleaning, etc.**
- **We are helping to schedule groups and carpools**

http://casadeperegrinos.org/home/volunteers

Charity

- **PEEK Las Cruces Street Fair**
  - **Locust Street on the New Mexico State University campus**
  - **Thursday, October 5, 2017**
- **Duties:**
  - **Drinks, kids areas, general volunteer.**

https://peekoflc.nmsu.edu/

Events

- Halloween Bash
  - Bash or Carnival? Lets put it to a vote!
  - New Event Ideas?
  - How to handle food? Sodexo, Potluck, or Both?
- Updates
  - Other clubs were notified, we'll check back to you on progress with collaborators
  - As a A&S recognized club, we do have some funding available, more can be "unlocked"
  - Space/Time is set: **Friday, October 27th**

Game Night!

- We will be joining the CS 272 class for an evening of fun and board games! We will be playing Wizard (a card game) and Qwirkle.
  - **WHEN**: Friday Oct. 6 at 6:30 pm
  - **WHERE**: SH 124

Events

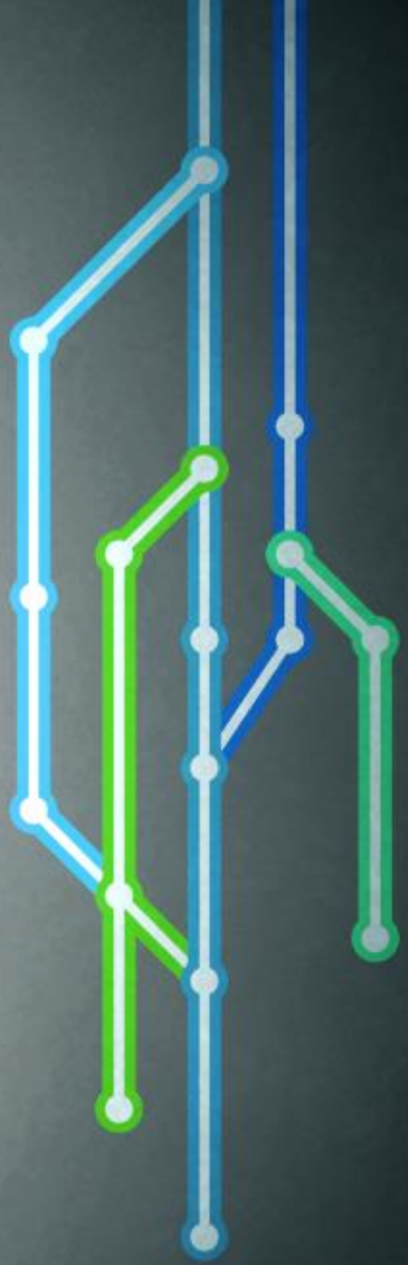
- A&S Council Meetings
  - 2/3 meetings attended for club recognition
  - Actually interesting and informative, worth going to
- Senate Meetings
  - Held in the Senate Chambers (3rd Floor Corbett)
  - Significant food available
- Meetings are held every other week

Nerd Moment

- Game Development Club has started development on their game! They meet after us, so stick around if you can ☺

Version Control: GiT

what it is, and why we use it



Quick Start

- GitHub and Code School teamed up to bring you an online terminal tutorial for using Git via try.github.io

- **You <u>can</u> use Git without a terminal.**
  - GitHub offers GitHub Desktop
  - Many IDEs (Visual Studio, Eclipse, …) have plugins or native support for connecting to Git repositories.
  - I recommend using these tools over the command line for new users, but learning the command line is good for tricky situations

- Comprehensive documentation is available at git-scm.com

Terminology

- **Version control**: A system which manages changes to files
  - i.e., it manages different *versions* of the same file(s)
- **Repository**: A location where project files exist
  - Can be local or remote (e.g., on the internet)
- **Hosts**: Something that contains one or more repositories
  - e.g, GitHub is a popular free online repository host

Version Control 101

- Other Names: **Revision Control**, **Source Control**
- Version Control systems help manage changes to files by different people at different times
- In a nutshell, it’s a managed way of managing a project files used by a team (or an individual)

Version Control 102

- Sophisticated version control software can track individual changes to files, manage several distinct "branches" of the code independently, and merging inconsistencies between changes no the same file

- Several flavors of version control exist
  - Subversion (SVN) and Git are popular in the CS department
  - Microsoft offers Team Foundation Server
  - Gaming development companies often have other types of version control as well

Git

- Git is presently the most popular choice for version control, and for good reason!
  - It's supported by most IDEs
  - It's decentralized
    - Every body gets an full copy of the repository
    - Code versions are managed via changes to files, not entire files
    - Multiple people can work on the same files without breaking stuff
      - Conflicts can be manually resolved.
- In general, most new projects should be managed by Git

Git vs GitHub

- GitHub manages Git-based repositories, but it’s by no means the only one.
  - Bitbucket is also relatively common

- Some hosts also provide additional team centric features
  - This can include Issue Tracking, Team Management, User Stories, Kanban boards, Continuous Integration, Code Review, etc.
  - Visual Studio Team Services is popular for private .NET teams
  - GitLab is popular for private Java teams
  - Both offer “free” versions, with premium features/versions available

Git: Workflow

- Create a remote Git repository (e.g., on Github)
- Clone the remote repo to a local one on your computer

- In your local repo:
  - Add some files
  - Commit your changes to Git (with a helpful message)
  - Change/add files
  - Commit your changes again (with a helpful message)
  - Push all your commits to the remote repo (e.g., Github)

Git Notes/Terminology

- A repository (repo) is simply a folder with files, including some special files describing itself
- Changes including everything do you to change the repo
  - E.g., adds, deletes, renames, etc.
  - Every change needs to recorded!
- A file can exist in your local repo, but not be tracked by Git!
  - YOU choose what the repo recognizes
- Commits are like checkpoints. Commits do not affect the remote repository until you tell them too.
  - You can return to a checkpoint state at anytime once it’s made
  - You can even revert a single file

Demo

The Basics

Working with Git Repos

- Often, you'll encounter a link to a web hosted repo.
- Often, that repo will contain a README file of some kind
  - These include author written instructions/guides for the code in the repo!

- If you are making a web hosted Git repo yourself, guides exist!
  - E.g., GitHub walks you through the process

Git: Essential Commands/Actions

```
git init
```

- Creates a Git repo in the current directory
- Not connected to remote repo yet!

```
git remote add origin <repo url>
git push -u origin master
```

- Sets the remote repo

```
git clone <repo url>
```

- Creates a new folder and creates a Git repo inside it
- Copies everything from the remote repo to the new local one

Git: Essential Commands/Actions

`git add <files>`

- Uses to start tracking specified files
- Can use wildcards (e.g., git add *.txt)
- Can use folders (e.g., git add src/)

`git rm <files>`

- Opposite of add!
- NOTE: you <u>must</u> remove files you delete

- Git can often figure out when files are moved/renamed, so long as you don't change files too much in the process

Git: Essential Commands/Actions

```
git commit –m “commit message”
```

- Wraps add/rm/etc commands into a **commit** with a specified commit messag
  - Your commit message matters! Others will read it to quickly figure what changes the commit contains!
- If –m “message” is excluded, a simple **vi** text editor will open
  - If you forget and you don’t know how to use vi, then Ctrl-C your way out of there! Lookup basic vi usage to get started, but it’s not important if you use a GUI

Git: Essential Commands/Actions

`git fetch`

- Checks the remote repo for updates, but doesn’t actually apply them

`git pull`

- Applies changes from the remote repo, if such changes exist

`git push`

- Applies your changes to the remote repo
- MUST pull before you push! (else, you will get an error)

Git Workflow 2.0

```
git fetch
git pull
<do changes>
git add <changes>
git commit –m <message>
git fetch
git pull
git push
```

Demo

Two Clients

Git: Branching

- Often, you want a group to work on a new feature/bug/version without messing with your nice, mostly stable project
- To do this, you can create **branches** of your project.
  - These are distinct versions of the code that teams can commit to, separate from the original, or **master**, branch.
- For the most part, you don't want to push to **master**, you want to push to another branch.
- Once your changes are worthy, you **merge** your new branch back with the master branch (or another branch!)

Git: Branching

`git branch <branch name>`

- Creates a new branch
- Current branch is <u>unchanged</u>

`git checkout <branch name>`

- Switches the current branch to the specified one

`git checkout -b <branch name>`

- Creates and switches to a new branch

`git branch -l`

- Lists available branches

Git: Merging

- Merging will combine a two branches together. So long as nothing conflicts, the merge will occur seamlessly.
- If issues do exist, you need to go through those issues and determine which changes you want in the final result.

```
git merge <other branch>
```

- Merges the other branch into <u>the current branch</u>

Git Workflow 3.0

```
<current branch is master>
git branch hotfix
git checkout hotfix
<make/commit changes>
git checkout master
git merge hotfix
git commit –m “message”
git push
```

Git Conflicts

- Scenario:
  - Amy and Phillip both clone **JavaFX_Tutorial**
  - Amy changes the README, adds the change, and commits
  - Phillip changes the README, adds the change, and commits
  - Amy pushes her commit.
  - Phillip wants to push his commit, but gets and error.
- Which version of the should we take?
  - Potentially both! We just need to sort out which changes to the file we want, then commit those changes.
  - However, this is still a pain, and should the scenario should be avoided. However, it is often a necessary evil.

Merge Conflicts

- Scenario:
  - Amy and Phillip both clone **JavaFX_Tutorial**
  - Amy changes the README, adds the change, and commits to **Master**
  - Philip changes to a new branch, "Docs".
  - Phillip changes the README, adds the change, and commits **Docs**
  - Amy pushes her commit.
  - Phillip pushes his commit.
  - Phillip tries to merges **Docs** into **Master**
  - Automatic merging fails due to conflicts
  - Phillip resolves conflicts with a new commit
  - Phillip pushes to **Master**

DEMO

Conflicts

Important Git Folders/Files

- Git tracks changes and all branches via the `.git/` folder
  - TL;DL: don't touch it!
  - Sometimes hidden, but it is there!
- Git can automatically ignore certain files/folders if specified in a `.gitignore` file
  - Must add it first before it starts to apply!
  - PS – this is just a plain text file
- Additional settings are specified in the `.gitattributes` file
  - Also a plain text file
- NOTE: all this special stuff is in the root folder of your project

Git Credentials

- To push to a remote repository, you will often need to give a username and password.
- You can skip entering it if you specify your global username and password
- You can also connect to your repo with SSH when you clone it
  - More work upfront, but ideal

What to Put in Your Repo?

- Plain text files
    - code, scripts, xml, …
- Some images
    - smaller is better
    - Try not to change them
- Some audio files
    - again, keep it small

- Once pushed, a file is **permanently** part of your repo
    - If you clone one branch, you get them **all**, keep large files **out** of your repo

Okay, but I need large files

- Use Git LFS
  - Large File Storage
- Keeps specified files somewhere else, but pretend its in your repository
- Great for game development, doesn’t change workflow once configured

- Install with (do this **only once**):

```
git lfs install
```

Git LFS

- Tracking can be setup individually:

```
git lfs track “*.psd”
```

- Or it can be set in `. gitattributes`
  - Must add!

Questions?