# Notes for HackerYou Summer Academy

## GitHub

### Pushing Projects onto a GitHub Repository

In the command line, enter the following commands

**git init**

#initializes the folder you are currently in for git

**git add .**

#adds all the files located in the folder (‘.’ 🡪 selects all, but you can add files one at a time if you want)

**git status**

#not necessary, but checks the status of what has changed

**git commit –m “*your message here”***

#commits the changes you have made

**git log**

#good practice to check what you have done so far

**git remote add origin *<insert link here>***

#used to remotely connect your files offline to your repository online

**git push -u origin master**

#pushes local repo to server

### Forking a Repository

**Click ‘Fork’ on the repository you want on Github**

#forks over the repo

**git clone *<link to your forked repository>***

#creates a copy on your local repository

**cd clone**

#enters folder

**git remote add upstream *<link to your original>***

#assigns the original repository to a remote called “upstream”

**git fetch upstream**

#Pulls in changes not present in your local repository, without modifying your files

## Agile Development Notes

### Models, Views, and Controllers (MVC)

* An architecture used to develop interactive applications
* **Model:** 
  + Responsible for maintaining the state of the application
  + Is the only item that interacts with the database
  + Can also be used to code validation tests
    - Very useful since it is the gatekeeper of the database
    - Thus, before you store anything into the database, the model can check if the data is valid
    - Otherwise, you may end up storing invalid data
  + Rails does not store anything
  + Always saves to the database
  + Models are smart, they do your logic
  + Always have to tell it what type of data
    - Migrations
      * Every migration is run one time, tells the database to change their schema
      * rake db migrate
        + Causes it to update the database
      * Migrations can be run up and down so you can change it
        + Before it was basically like a massive excel file and if you changed one data scheme, you could not undo it (ie have to reinsert it and screws up the order)
      * Ask about AddDescriptionToProducts
    - Rake
      * Runs code within your app without having to run it
      * rake –T
      * db (database)
    - Validations
      * presence: true 🡪 checks if it is there
    - Callbacks
      * Triggers when stuff happens
      * Ex: e-mail a user when an order has been shipped
      * Ex: class < Order::Base
        + send\_email customer.e-mail “Your order has been placed!”
      * Finite set of things you can do with the model
        + You don’t need to create your own
    - Custom Logic/business logic
* **View:**
  + Generating the user interface based on data in the model
  + Can be used with HTML, CSS, and ERB
  + Never handles data, only displays
  + Contain only html and ERb files
  + Rendered by the browser
  + Views do not do ANY logic
    - Only some, like loops and conditionals
  + Output vs. logic
    - <%= Time.now %>
    - <% %>
  + ***Ask question about how the browser sends information regarding Ruby***
* **Controller:**
  + Orchestrates the application
  + All user input goes through the controller
  + It interacts with the model and then displays an updated view for the user
  + Access control
  + Handling errors
  + resources :products
    - Opens up all of the links quickly
    - rake routes 🡪 shows all the possible links your website can open
    - Can also see this in the errors

****

### File Configurations

* Convention over configuration
* You don’t need to tell Rails where to look for shit, it knows as long as you place them in the right folder

### Cool Methods in Ruby

self.send(input)

dynamically sends itself questions? \*\*look this up \*\*

## Lecture / ROR Notes

### Creating an Application w/o a Scaffold

Sometimes when we are creating applications, it is actually more useful to build ***without*** a scaffold. This is because a scaffold sometimes adds a lot of unnecessary things in your application. Furthermore, it links a whole bunch of things together and as you know, it is much more difficult to debug if you look after the fact.

Consequently, it is useful to know how to build without one sometimes so you can keep track of everything.

**Step 1: Generate a new project**

*rails new depot*

**Step 2: Generate a controller**

*rails generate controller Product*

**Step 3: Make views and actions for what you need**

Create all the views manually by inserting them in the appropriate folders in your file. You will need to account for the following:

* html.erb files in Views
* Insert all code in it that is needed

**Step 4: Generate your model**

*Rails g model Product title:string…*

*rake db:migrate*

### Testing

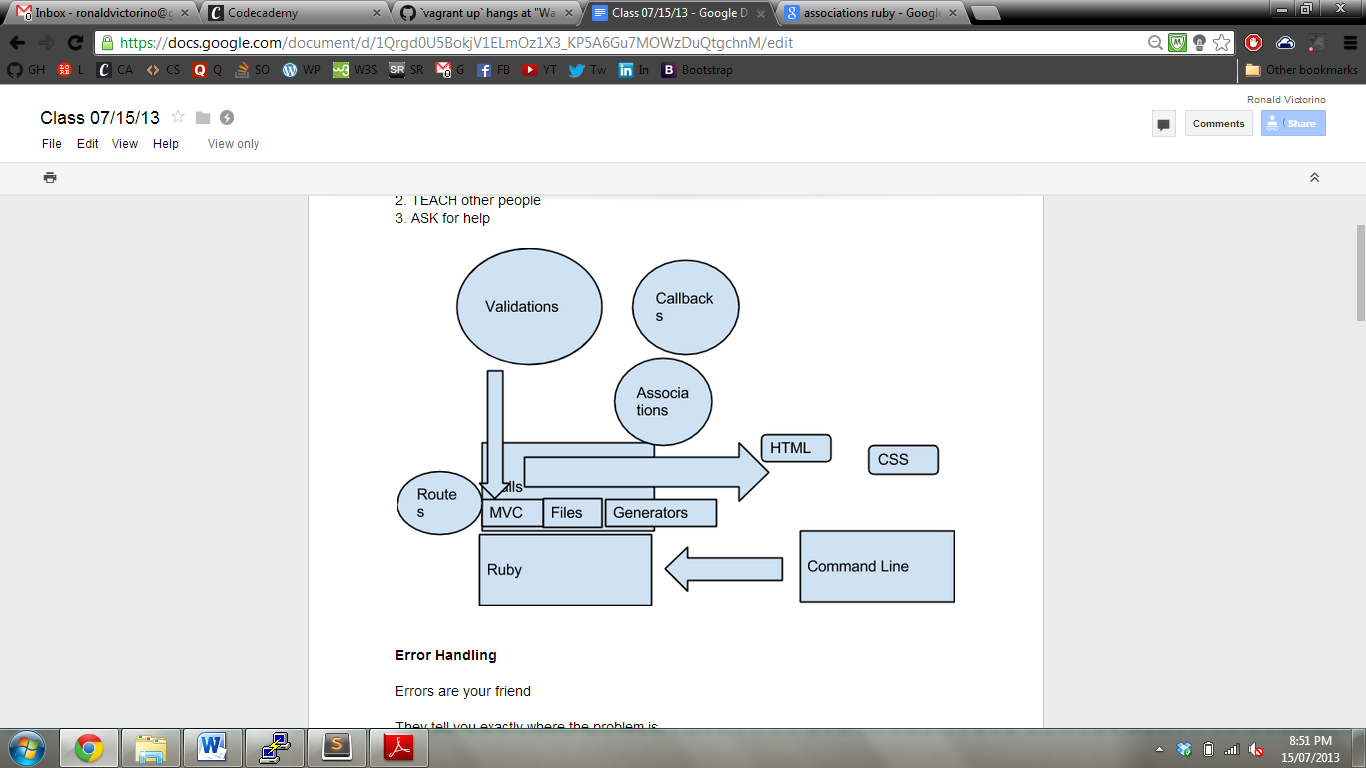
* Always asserting to make sure things are false
* Test fails come one at a time as to not overwhelm the user
  + Tells you exactly which test line failed
  + That’s why you use skip first
* You want to test to make sure you can account for change
  + Modularity
  + Failure
  + Efficiency
* Test fails come one at a time as to not overwhelm the user
  + Tells you exactly which test line failed
* Test driven development vs. cowboy coding
  + Write a test first rather than the code
  + This way, you are actually testing for functionality before you even write the code
  + Cowboy coding introduces a bias into the tests you write
* Corey Haines and Aaron Patterson Video
  + “If you have an idea, start typing. And the second person will eventually know when to jump in”
* Skip
  + Written into the methods so that only one test runs at a time
  + Must take out one by one

**Later stuff to cover**

* Check out minitest
* Regular Expressions
  + Used to match a pattern against strings
  + Ex: \d will capture all of the digits
* Testing

### Errors

* Errors are your friend because they tell you where the problem is
* Make sure you are familiar with the code written
* No route matches 🡪 not the view
  + A result of no route being identified in routes
* Debugger
  + Enter the word *“debugger”* into any part of your code
  + Will stop when it is first accessed
    - next = go to the next line
    - step = go to the source of this function
    - continue = stop debugging and keep running the program
    - p = print the value of an object or variable
    - e = run some ruby code (put code after) – stands for evaluate



class ProductTest < ActiveSupport::TestCase

def test\_the\_truth

assert true

end

end

These two are the same, just different formatting

class ProductTest < ActiveSupport::TestCase

test "the truth" do

assert true

end

end

* Use rake test:models
  + Use instead of ruby test.rb
  + Using ruby adds a whole bunch of shit that you don’t need
* i18n 🡪 internalization
  + Used to translate things
* Testing takes a long time because it’s constantly clearing a database, reinitializing it, then jamming it into the code

### Associations

* Unique functionality on top of models
* Nothing to do with validations, just another feature
* **Expresses a relationship between two models**
* Ex: blog post vs. comments
* You will want to keep the associations in the database
  + Have the comments point to the blog post, rather than vice versa
  + Easier
* t.references :order is the same as t.integer :order\_id
* Concerns
  + New to rails 4
  + Great for creating code that you are going to be sharing across all things
  + Every user has their own session from when they visit the site
  + Session is a hash
  + File name is important
* Rescue
  + Allows you to deal with an error
  + Ex: rescue ActiveRecord::RecordNotFound
    - @cart = cart.create
    - Session[:cart\_id] [=@cart.id](mailto:=@cart.id)
* Rails make it much easier for everyone to add line items to a database
* 3 steps: look in textbook! t.references, has\_many,

## Design Notes

### Twitter Bootstrap

#### Grid System

* Twitter Bootstrap uses a 960 grid statement
  + Total width of the page is 960px with 20px margins on the left and right
* Automatically has a container that spans the whole webpage
  + The containers have a -20px start already
* Uses 16 columns (for v1.4)
  + span1 all the way up to span16 with 20px between each
  + can actually use span-one-third as well
* With v2.0, they use a 12 column grid
* There is an automatic 60px padding at the top for a toolbar

#### Media Grid

* In Twitter Bootstrap, there exists a class called media-grid used in an unordered list
  + Automatically places images in a container
  + Identified by <li> tags
  + Ex: <ul class = ‘media-grid’><li><img></li>
  + The only input needed is the size of the image
* Placeholder.it is a website that shows you how large the image will be prior to changing the width and height of an image
  + Can type in right in the url (placeholder.it/widthxheight) and it will return you the image

#### Making Custom CSS Sheets

* If you want to add on to the Bootstrap style sheet, make a new file and use import
  + @import(url); 🡪lets our CSS file inherit all of Bootstrap
  + This allows you to add more classes and IDs to the existing bootstrap CSS without changing bootstrap by accident
* With v2.0 you can customize different settings before you downloa

#### Random Notes

* Use pull-right/left to float an element right/left
* To embed videos, go to Youtube file and click that way
  + Can also change width and such there and they will give you the appropriate resizing
  + Youtube videos use iframes
* Changelog is better now
  + Shows differences between versions
* Use the components section! It will be your best friend for design!
  + Contains all of the different things you may need to create

#### Creating a Drop Down List

* In the nav-collapse container, it will collapse as you resize the screen
  + It will also change it into a drop down menu
  + We have to initialize that menu
* To create the dropdown menu and search for your toolbar:
  + Dropdown menu
    - Inside the nav-collapse container, each link is added as a <li>
    - Place a ‘nav’ ul inside it
    - Make a new <li>, with a “dropdown” class
      * Also put a <a href = #></a>
    - Inside the <li>, create a ul called “dropdown-menu”
  + Search
    - Copy code from video
    - Uses “search-query”
* To actually drop down, you will need javascript

#### Tips and Tricks

* Debugging can be done right from the website
  + Click inspect element
  + Sees all the html and css rules for each element
  + Use magnifying glass to look over image
  + Shift + up /down while highlighting the px to grow/shrink the element
  + Click the middle button to check all the pseudo classes (ex: hover, focus, etc.)
  + Can change the elements right on the page to look at it
  + ***Very useful for debugging because you can see EXACTLY what is happening***
  + You can only use the web inspector to look at and change, but not save
    - Have to go into Sublime to change it later