**Rails Notesheet:**

Text highlighted in black is a console command.

Gray highlighted text is code inside the IDE.

**Starting a new project:**

rails new app\_name

If you need to, change the line involving sqlite3 to gem 'sqlite3', '~> 1.3.13' in the Gemfile, if you get the error, “Gem::LoadError: Specified 'sqlite3' for database adapter, but the gem is not loaded” Use cd app\_name to get into the app directory and then do bundle install afterward and bundle update (bundle update if you had to change the Gemfile because of sqlite)

**Starting a new project that uses MySQL instead of Sqlite3:**

sudo apt update

sudo apt-get install mysql-server mysql-client libmysqlclient-dev

rails new app\_name -d mysql

Get the service up and running (You need to do this next command every time you boot up):

sudo service mysql start

Log in as root:

mysql -u root -p

And enter your mysql password, if asked. Then, the rest is a one-time only thing:

mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('mypassword')

And in /config, type the mysql password into default:

default: &default

adapter: mysql2

encoding: utf8

pool: 5

username: root

password: ‘YOUR PASSWORD HERE’

host: localhost

At some point, it’d be much more secure if you made a new MySQL username that isn’t root, and also didn’t include your password in the file (especially if your app is in a public repository). Ideally, instead of typing your password in single quotes like above, you should be using an environment variable, like:

<%= ENV['APPNAME\_DATABASE\_PASSWORD'] %>

**Create a new database using MySQL:**

Log in as root (This won’t work unless you start the mysql service, as shown above!):

mysql -u root –p

Type these lines, not including the “mysql>” part of it. Also note that some values use single quotes, others will not, so look closely!:

mysql> CREATE DATABASE IF NOT EXISTS insertDatabaseNameHere\_production;

mysql> CREATE DATABASE IF NOT EXISTS insertDatabaseNameHere \_development;

mysql> CREATE DATABASE IF NOT EXISTS insertDatabaseNameHere \_test;

mysql> GRANT ALL PRIVILEGES on insertdatabasenamehere \_test.\* to 'insertYourAppNameHere'@'localhost' identified by 'insertYourMySQLPasswordHere';

mysql> GRANT ALL PRIVILEGES on insertDatabaseNameHere \_production.\* to 'insertYourAppNameHere'@'localhost';

mysql> GRANT ALL PRIVILEGES on insertDatabaseNameHere \_development.\* to 'insertYourAppNameHere'@'localhost';

mysql> FLUSH PRIVILEGES;

And under /config/database.yml, change each line with a “database” value to

database: databaseNameGoesHere\_development

(or databaseNameGoesHere\_production or databaseNameGoesHere\_test; It should be those 3)

Side note: There is also rails db:create, but I’m not sure if that’s only meant for sqlite3.

**Running a rails server:**

**If you use Codenvy:**

Just click the triangular play button, and if the drop-down comes down, select “run,” and it’ll make that command the default, so every time you click that button, it will be the command that it runs. A new tab will open up where the terminal was, and a blue link will appear near the top of it, next to the white text, “preview: ” and this is the link that shows your website in a new tab. Keep in mind that if you use Codenvy, **sometimes the “run” command button might not have set itself up properly**. If that happens, you need to modify the command, which can be done in the “Commands” tab. By default, the “Projects” tab is open on the far left, next to the blue Codenvy side-panel. Select the “Commands” tab underneath the “Projects” tab and under the “Run” category should be a single command called, “run.” Double-click it and a new tab will open up that allows you to modify the command. It currently looks like this:

cd && rails server -b 0.0.0.0

but should instead look like this:

cd your\_project\_name\_here && rails server -b 0.0.0.0

Change that, and in the bottom-right of that editor window is the save button. Don’t forget to save the command. Now it should work just fine. If you use Codenvy, the “run” button should work fine most of the time, but occasionally, this may happen with new projects.

**If you don’t use Codenvy:**

Make sure you cd into the project root:

cd project\_name

Now, you should be able to run the server:

rails server –b 0.0.0.0

**Add a new controller and add pages to it:**

Note that this is a manual way of adding it. Trying to do rails generate controller controllerName actionName did not always work for some reason, but otherwise, that’s the automatic way.

Go to routes.rb, under the /config/ folder, and add the line, get 'controllername/pagename', to: 'controllername#pagename' where “controllername” is the name of your controller, and “pagename” represents each page you want to add to that controller. Add this line for each page you want to add to a controller, or for each controller you want to add.

Under the /app/controllers/ directory, add a new file named after whatever you put above for “controllername”, plus “\_controller.rb” (example: controllername\_controller.rb). In my example, the controller is named, “pages” so the class inside pages\_controller.rb is called “PagesController” as shown below. Add this class declaration to the newly-created file:

class PagesController < ApplicationController

end

In between the class declaration and the end line, add a definition for each page that you put in the routes.rb above as “pagename”. In the end, it should look something like:

class PagesController < ApplicationController

def pagename

end

def perhaps\_another\_page\_you\_wanted\_to\_add

end

end

Note that the page definitions inside the class are indented as shown above (2 spaces)!

Then, under /app/views/, add a new folder named after your controller (whatever you put in the place of “controllername” in the step above), which in my case would be a folder called, “pages”

Inside that folder, add a file named pagename.html.erb substituting “pagename” with whatever pages you’ve defined. Add new html.erb files in that folder for each page in that particular controller. Add new folders in the views folder for each controller, and new html.erb files inside those folders for each view defined in the controller file and as defined in the routes.rb file.

You can now start adding html code to these files. Remember: html tags starting with % are for ruby code, tags starting with %= are for displaying that code in the view and all ruby tags end with %

**Set the root (home) page:**

Under your routes.rb file, add this line above the “get” lines:

root 'controllername#pagename'

Example: “controllername” will be “pages” and “pagename” will be “home” so the line will say:

root 'pages#home'

Now, you have to find the “get” line that in my case says:

get 'pages/home', to: 'pages#home'

and delete it, otherwise the site will not work. Additionally, all pages within that same controller need to have the controller name taken off. Example:

root 'pages#home'

get 'pages/about', to: 'pages#about'

needs to become (changed parts are highlighted in red/green)

root 'pages#home'

get 'about', to: 'pages#about'

get ‘some\_other\_page’, to: 'pages#some\_other\_page'

and link\_to links *to* your root view need to become <%= link\_to “Link\_Text”, root\_path %> instead of <%= link\_to “Link\_Text”, controllername\_pagename\_path %> and other links that use that controller need to have the controller name dropped off of them, so they go from something like <%= link\_to “About”, pages\_about\_path %> to looking something like this instead: <%= link\_to “About”, about\_path %>

**Changing the default title for the website:**

In /app/views/layouts/application.html.erb, near the top is text in between the <title> </title> tags, and that’s your default website title. Change that to whatever you want.

Adding a favicon:

Under /app/assets/images/ upload or paste the new .ico/.png/.jpg or whatever you prefer.

Then, in /app/views/layouts/application.html.erb add the following line between your head tags:

<%= favicon\_link\_tag asset\_path(‘filename.png’) %>

**Linking css to a webpage:**

First, put the new stylesheet under /app/assets/stylesheets and I recommend naming it after the page you want to connect it to, just in case you might want a different stylesheet for another page, but you would rather not reuse it. So, name it pagename.css (replacing “pagename” with the actual name of the page and remember to do that for the remaining lines of code in this section) and that should do just fine. Then, inside your html.erb webpage that has the same name, add:

<%= stylesheet\_link\_tag "pagename" %>

Then, under /config/initializers/assets.rb, under the line about the assets version (probably always version 1.0 by default, and it’s the only uncommented line) add this:

Rails.application.config.assets.precompile += %w(pagename.css)

Once you restart your server, you should now see the changes.

**Setting up version control using git (locally on machine, this isn’t a Github repo, yet):**

git --version

will tell you what version of git you have, or will tell you if you don’t have git installed.

Set up configurations (you have to tell Git who will be committing the code):

git config --global user.name “your\_username\_here”

git config --global user.email “your\_email@cool\_cats.com”

To see what’s already in the config file:

git config or git config --list

To inititialize the .git stuff in your project:

git init

Add files to be tracked or committed:

git add –A or later on you might use git add . (Note the “**.**” At the end of the second command!)

Now commit:

git commit –m “I made a new rails app and added some pages. Cool.”

You can use whatever message for the commit that you feel like using.

If you break something, don’t commit, use git checkout -f to restore back changes that might’ve broken the app. If you goofed hard, didn’t realize it, committed it, etc. then to revert back to a previous commit, git reset --hard HEAD will bring you back to the last commit, while using the command git reset 67ae32cc (substituting the hex value with the hex value for the commit you want to revert back to. Look on Github for the list of commits, if you aren’t sure what the hex value is).

git status

will tell you if there are unpushed commits.

**Dumping that local repository onto Github:**

Easier way to add an SSH key(in Codenvy):

(Menu at the top of the viewport): Profile > Preferences > VCS > Click the Github logo. Then just let it connect through Github and the rest is taken care of for you. Now just jump down a few lines below this to the paragraph that starts with, “Still on your Github page,” and follow the instructions from there.

Generate an SSH key:

cd ~/.ssh/

Then,

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

Substitute your email with the placeholder, keeping the quotes. You should name it id\_rsa and you can leave the passphrase blank, if you want. Print out the key:

cat ~/.ssh/id\_rsa.pub

It’ll dump out a lot of text. Copy all of it, and go to your Github account, and go to the menu at the top-right of the screen with your picture and click,

Settings > SSH and GPG Keys > New SSH Key

And now just name the key, and paste the key contents in and save it.

Still on your Github page, add a new repository (next to account, its icon is usually a “+” sign). I would recommend that you name the repository after whatever your IDE’s project is named. Once you’ve named it (and made it public or private) then create it (click “create” or “next”, whatever it was). At the top of the form, your 3 options are “Setup in desktop”, “HTTPS”, or “SSH”. Choose “SSH”. Now copy the line in the middle of the form that says something like,

git remote add origin git@github.com:Username/your\_app\_name.git

and paste it into the command line and run it (substituting “Username” and “your\_app\_name” with their proper names). Now copy/paste the line below it into the command line (Don’t forget to git add and git commit first, if you haven’t!):

git push -u origin master

Now, your local repository and your Github repository are now linked!

Note: If that didn’t work, try git add . (don’t forget the “**.**” at the end!) then git commit –m “Just adding some files to my new Github repo” and then try git push –u origin master once more.

**Deploying to Heroku:**

You may optionally want to do this because it really helps with catching problems you might see in your production when you deploy your app. If you deploy to Heroku, you can’t use SQLite3.

Switching from **SQLite3** to **Postgresql**:

Before getting started, you may not have postgres installed. To fix this,

sudo apt-get install postgresql

sudo apt-get install libpq-dev

Now you can use

gem install pg

and it should all work now.

Open the Gemfile, cut and paste the lines that say:

# Use sqlite3 as the database for Active Record

gem 'sqlite3', '~> 1.3.13'

And move them below, into group :development, :test do (just below that line). Outside of that group, make a new group, :production and the whole group, etc. that you add should look like:

group :production do

gem ‘pg’ (Retype these single-quotes, they don’t copy correctly into Codenvy)

end

If using below Rails 5, then also add gem ‘rails\_12factor’ below the gem ‘pg’ (‘pg’ means postgres) line. Run:

bundle install --without production

And commit.

Now install the Heroku CLI using

curl https://cli-assets.heroku.com/install-ubuntu.sh | sh

And then to log in using the CLI (Use –I if you’re in Codenvy and can’t open a browser)

heroku login -i

and give it your login credentials for your Heroku account. Add your ssh key with

heroku keys:add

Now you can create a new app on the Heroku website (which allows you to choose your own name) or using your command line (use heroku create for command line which will give it some custom randomized name).

heroku git:clone –a the\_full\_heroku\_app\_name

git add .

git commit -am “Deploying to that sweet new Heroku app I made”

git push heroku master

Done. However, if you had to delete the original app you created in Heroku, like I did (it refused to think that postgres was a thing, no matter what), then

git remote set-url heroku git@heroku.com:sublime-range-23056.git

git remote –v

And then deploy to Heroku. Sometimes the production version breaks beyond repair, and you might need to simply nuke it and redeploy it, if that happens (your Heroku app, not your local or Codenvy app). If you want to rename the app:

heroku rename whatever-name-you-want

**Adding a database to store things like articles, etc.**

You might want a database to store all of your articles so that you can fetch the latest one, keep track of the oldest one, perhaps even search your database of articles. Make sure (and this applies to all sections above as well) that you start by making sure your current directory is inside your app (cd app\_name). If you’re using Rails 4 or earlier, use rake instead of rails in the line below:

rails generate migration create\_coolnames

I really hope I don’t have to explain anymore that things like “coolname” need to be replaced with whatever name you need to use, like “article” or “news”. Inside /app\_name/db/migrate, you’ll see a file that’s something along the lines of 2102049\_create\_coolname.rb and you need to open that. Add this to the create\_table loop so it looks something like:

def change

create\_table :coolnames do |t|

t.string :title

t.text :description

t.date\_time :created\_at

t.date\_taime :updated\_at

end

end

Save it, and then rails db:migrate (or rake db:migrate if your Rails is that old). Keep in mind that rails db:rollback will wipe your schema.rb if you forgot to add something. But you shouldn’t do that. Instead, if you have to add something later down the road, rails generate add\_parametername\_to\_coolname and in that new file, inside the def change block of code, add:

add\_column :tables\_name, :name\_of\_new\_field\_to\_add, :fields\_type

and if you haven’t yet, make sure that you have fields for when something is created or last modified. You can still add these to the same file you just generated. Remember, these two new field are of type datetime:

add\_column :articles, :created\_at, :datetime

add\_column :articles, :updated\_at, :datetime

Under /app/models/ create a new file named after your table. If your table is called “article”, your file will be named, “article.rb”. Open it, and add the following to it (Example still involves “article” as the name of the table):

class Article < ActiveRecord::Base

validate :title, presence: true, length: {minimum: 3, maximum: 50}

validate :description, presence: true, length: {minimum: 10, maximum: 250}

end

Use those “validate” lines for each of your required fields, so that anything important that isn’t filled in automatically, doesn’t get saved to the database with a value of “nil” and that they meet the size requirements. After that, if you are in the ruby console making changes to the database (to get to the ruby console, run rails console), run the console command reload! (with the “!”, a.k.a. “bang”) and the new changes should take effect. If you have to make changes to this file, make sure you run this console command. If you encounter an error when trying to add a new element to the database, run new\_element.errors.full\_messages and it should print out the issue.

**Adding input to your views:**

When you want to create new articles, or users, etc. you want to go to your routes.rb file and add the name of the controller you want to use, as a resource: resources :articles and if you want to see the routes now generated, run rails routes afterward. In /app/controllers/, add a new file named after the controller you’ve been using up to this point and with “\_controller.rb” at the end, so something along the lines of “articles\_controller.rb” is what it should be called. Add the following to that file (On next page):

class ArticlesController < ApplicationController

def new

@article = Article.new

end

def create

@article = Article.new(article\_params)

if @article.save

flash[:notice] = "Article was created successfully"

redirect\_to article\_path(@article)

else

render 'new'

end

end

private

def article\_params

params.require(:article).permit(:title, :description)

end

end

Bear in mind that whatever you might put in the place of “new”, you must have a view for that. You don’t need a page for create, though. That’s just going to be used when we need to create new articles. Also, if you’re going to have input, you need the variable instantiated here before you can use it in the form helper. If you have not yet, go to /app/views/ and add a new folder named after your controller (like, a folder named “articles”). Inside that folder, add files named after each method defined in the controller you just made, so “new.html.erb” is the filename. Open this file, and now you can insert the form inputs here. Listed below are a number form helpers:

The most basic helper form tag:

<%= form\_tag do %>

Form contents

<% end %>

Supposing you have something like this:

<h1>New Article</h1>

<%= form\_for @article do |f|%>

<p>

<%= f.label :title %>

<br>

<%= f.text\_field :title %>

</p>

<p>

<%= f.label :description %>

<br>

<%= f.text\_area :description %>

</p>

<p>

<%= f.submit %>

</p>

<% end %>

Much of this should be self-explanatory, but that is what an example of a file.html.erb looks like. Notice the f.submit at the bottom of it. Over in /app/views/layouts/ open up application.html.erb, and inside the block for <body> add this:

<% flash.each do |name, msg| %>

<ul>

<li><%= msg %></li>

</ul>

<% end %>

**Edit articles:**

If you can add articles, etc. in a controller, you may also want to edit them. Edit your controller to look similar to this (adjusted however necessary to fit your project):

def new

@article = Article.new

end

def edit

@article = Article.find(params[:id])

end

def create

@article = Article.new(article\_params)

if @article.save

flash[:notice] = "Article was created successfully"

redirect\_to article\_path(@article)

else

render 'new'

end

end

def update

@article = Article.find(params[:id])

if @article.update(article\_params)

flash[:notice] = "Updated the article"

redirect\_to article\_path(@article)

else

render 'edit'

end

end

def show

@article = Article.find(params[:id])

end

private

def article\_params

params.require(:article).permit(:title, :description)

end

Now add a file to your views/conrollername/ folder, named “edit.html.erb” and add something along these lines to it:

<h2>Edit page </h2>

<% if @article.errors.any? %>

<h2>Fix the following errors:</h2>

<ul>

<% @article.errors.full\_messages.each do |msg| %>

<li><%= msg %> </li>

<% end %>

</ul>

<% end %>

<%= form\_for @article do |f|%>

<p>

<%= f.label :title %>

<br>

<%= f.text\_field :title %>

</p>

<p>

<%= f.label :description %>

<br>

<%= f.text\_area :description %>

</p>

<p>

<%= f.submit %>

</p>

<% end %>

You may have to make sure you run rails db:migrate before it will work, and you also need to make sure that you have any articles to edit in the first place. My url that I use to make a new article looks like: http://node15.codenvy.io:37707/articles/new and to edit it, type the id number after “article/” instead of typing “new”, so it looks like http://node15.codenvy.io:37707/articles/1/edit if I wanted to edit the first article.

**How to list all articles in the controller:**

Add to your controller file (I recommend that you make this the first function):

def index

@articles = Article.all

end

You can call @articles whatever you want, but it’s usually a good idea to keep it consistent. Make a new file in views/controllername/ called “index.html.erb” and add to it:

<h2>Welcome to the index!</h2>

<p>

<%= link\_to "New article", new\_article\_path %>

</p>

<table>

<tr>

<th>Title</th>

<th>Description</th>

</tr>

<% @articles.each do |article| %>

<tr>

<td><%= article.title %></td>

<td><%= article.description %></td>

<td><%= link\_to "Edit", edit\_article\_path(article) %></td>

<td><%= link\_to "Show", article\_path(article) %></td>

</tr>

<% end %>

</table>

Add this line (optionally): <%= link\_to "Go to index", articles\_path %> at the bottom of your edit, new, and show views in views/controllername, and optionally add:

<%= link\_to "Edit", edit\_article\_path(@article) %> to your show.html.erb view.

**Creating a separate branch on Github to work on:**

git checkout -b new-branch-name

Uploading/Downloading content: