iOS Foundations II
Day 1

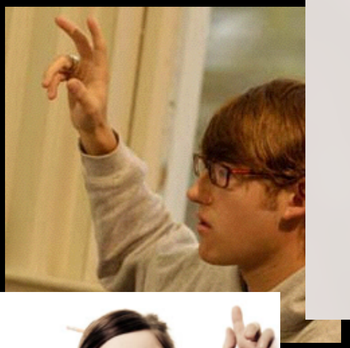
- Class Intro
- Variables
- Classes
- Method & Scope
- Properties & Variables
- Inheritance
- Intro to Git

Brad

Course Format

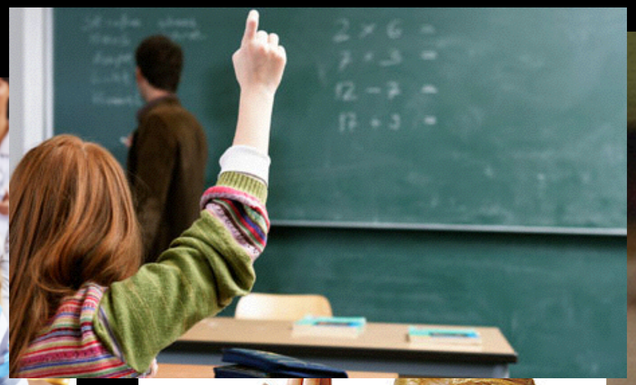
- 2 hours of lectures each class, split by a 10 minute water/restroom break in between.
- Each class will begin with homework review, except for today since there was no homework!
- Homework is graded twice, once in the middle of the course and then at the very end. But homework is assigned after each class.
- All slides, sample code, and homework will be posted to the class github repository immediately after class.

Tips for doing well





Ask all the questions


Solid Note Taking

- All slides, code, and recordings are posted immediately after lectures are over.
- So instead of blindly typing everything we type or everything on the slides, try to just absorb the information into your brain.
- Scientific Studies have shown students typically have the best results if they take notes on a notepad during class and keep their laptops closed.

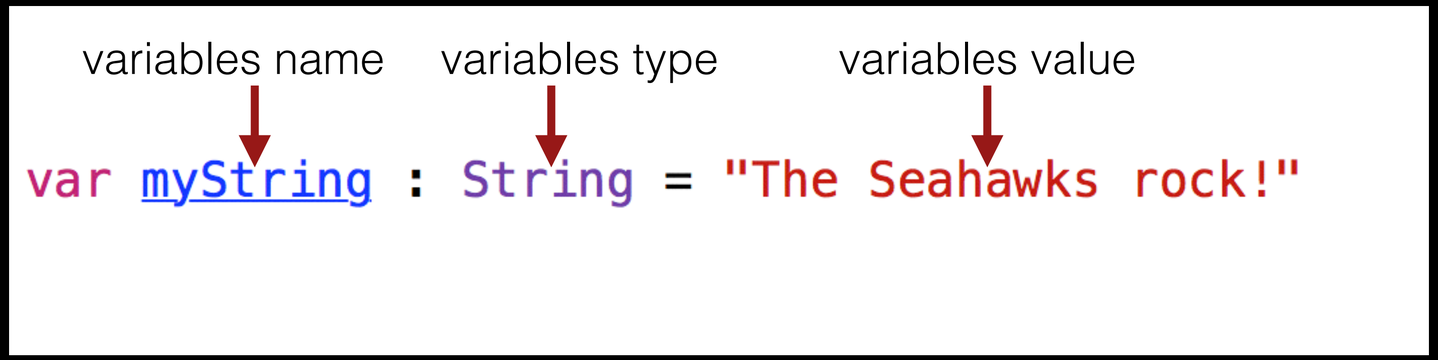
Don’t get stuck

- Remember to always consult fellow students and your teacher if you are stuck on something or if something is unclear.
- You can always look at the sample code from the lectures as well.
- Knowing how to Google your coding problems is a very important skill as a developer, so always try Googling as well. Chances are someone has asked that exact same question on stack overflow or another forum.

Variables

Variables

- You use the keyword var to make a variable.
- A variable must have the same type as the value you assign to it.
- A variable has 3 parts : name, type, and value:



- You don't need to always assign the type explicitly, as Xcode can infer the type of the variable based on the value you give it:

```
var anotherString = "Booo 49ers suck"
```

Constants vs Variables

- Use the let keyword to make a constant.
- Constants are just like regular variables, except they can only be assigned a value exactly once.

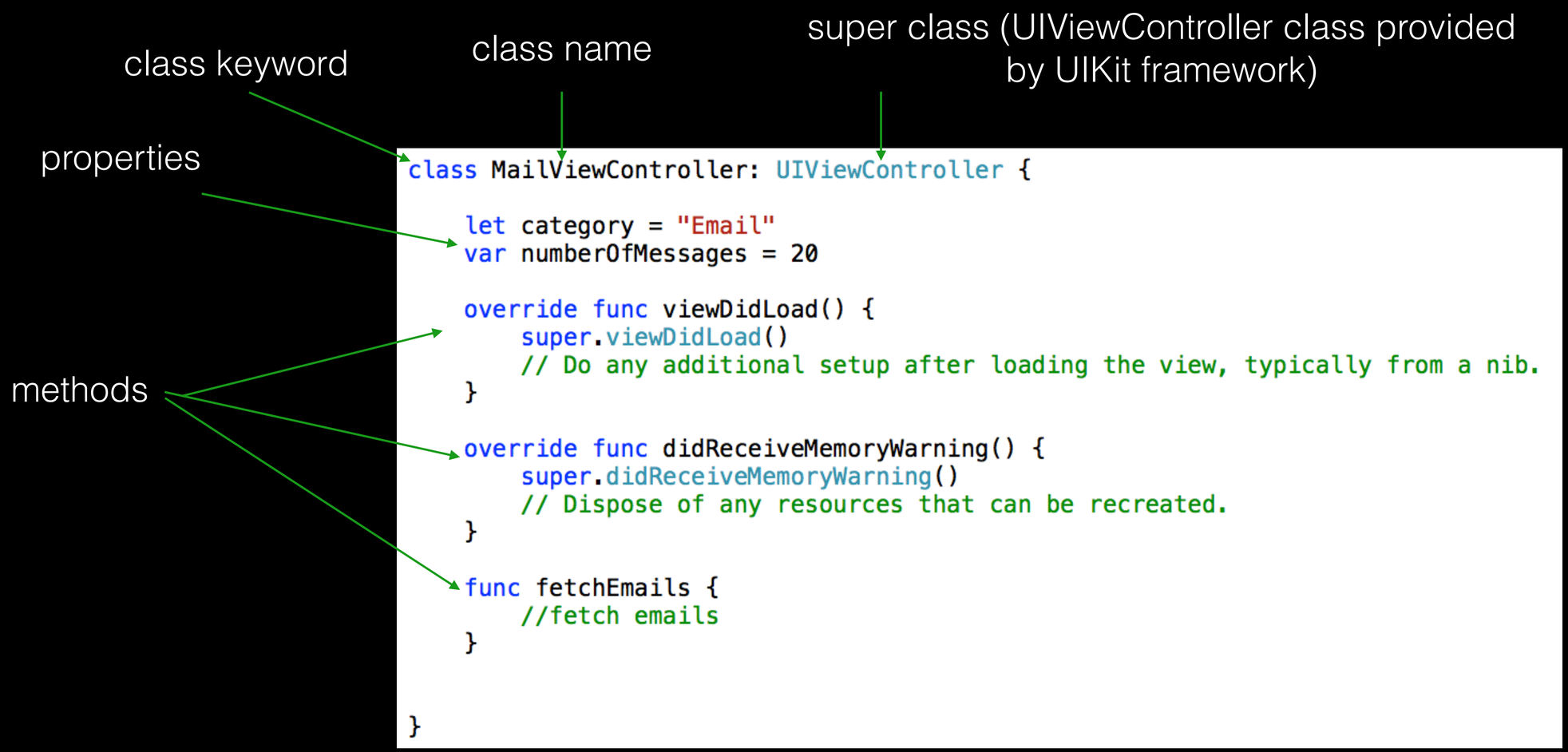
```
let myName = "Brad"
myName = "Bradley"

var city = "Seattle"
city = "Portland"
```

Invalid

Valid

Classes & Objects

What a class looks like



Classes

- A class is a template for creating objects in your code.
- What a class contains, all of its methods and properties, is considered the scope of the Class.
- An Object is just an **instance** of a Class.
- In Swift & Objective-C the scope begins and ends with curly brackets {}.

Class Scope

```
class Person {

    var firstName = "John"
    var lastName = "Doe"

    init(first : String, last : String) {
        self.firstName = first;
        self.lastName = last;
    }

    func sayHello() {
        println("Hello!")
    }

    func sayGoodBye() {
        println("GoodBye")
    }
}
```

Scope begins

Scope ends

Methods

- A function that is defined inside of a class is called a method.
- Think of a method as an action that your class can perform.
- Methods can take in info, referred to as parameters, that can then be referenced inside the methods.
- Methods can also send info back when they are finished, referred to as return value(s)

Methods

```
func doSomethingWith( title : String) -> String {

    println(title)

    return "Done!"
}
```

Parameter

Return Type

Using the parameter

Returning a String

Method Scope

- Similar to a Class, a method's scope begins and ends with the curly braces in Swift and Objective-C.
- Any variables created inside a method are destroyed once that method is finished. These types of variables are called local variables.
- Conceptually, think of the method as the owner of the variable. If no other object becomes the owner of the variable, it is then destroyed.

Method Scope

```
func sayHello() {

    var greeting = "Hello"
    println(greeting)
}
```

Scope begins

Scope ends, greeting dies

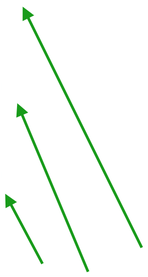
- our greeting variable is a local variable, since it is created inside of a method

Properties

- Properties store a Class's Data.
- Properties are globally available in all of the Class's methods.
- Properties stay alive as long as the object stays alive. They differ from local variables because properties don't die at the end of a method.
- There are two ways of accessing properties, which you will see in the next slides.

Properties

```
class Person {

    var firstName = "John"
    var lastName = "Doe"

    func printFirstNameThreeTimes() {
        println(self.firstName)
        println(self.firstName)
        println(self.firstName)
    }

}
```



You can access the Object's properties
with self, then a dot, then the name of the property

Properties

```
class Person {

    var firstName = "John"
    var lastName = "Doe"

    func printFirstNameThreeTimes() {
        println(firstName)
        println(firstName)
        println(firstName)
    }

}
```

Somewhat confusingly, you can also access an Object's properties without the self dot

Properties

- I greatly prefer to use the self dot, because it instantly tells me the variable I am looking at is a property and not just a local variable that was declared somewhere else in this method:

```
println(self.firstName)
```

I know for a fact firstName is a property here

```
println(firstName)
```

firstName could be a property or a local variable in this method,
now I have to search through the code and find out

Properties and their types

- Properties need to know what type they are.
- All objects are instances of a class. All primitives variables (Int's, Strings, Doubles) are a specific type. But in general we say type when referring to an object's class or a primitive's type. It can be confusing at first.
- In the next slide we will look at how to declare a properties type.

Properties

- You can explicitly declare a property's type by adding a colon after the name, and then the name of the type.

```
class Person {

    var firstName : String = "John"
    var lastName = "Doe"

    func printFirstNameThreeTimes() {
```

- But because we are giving these properties default values, Swift & Xcode can infer what type the property is! Thats why we didn't give lastName a type and it still works. Later on we will see how not giving a property a default value changes how we setup a class.

Init

- The way to create an instance of a class is to call its initializer(s). This is considered a constructor, which is a general term that all programming languages use to describe the way an object is created.
- To define an initializer in a class, you use the keyword init.
- It works just like a regular method, but it doesn't need the func keyword.
- Inits can have parameters, which are often used to set the values of properties.
- If you don't explicitly create an init, a generic one is provided for your class by default.
- In Swift, calling an Init is as simple as typing the name of the class, and then a parenthesis.

Init

- Here we have our Person class, with an init that takes in two strings as parameters.

  We then use those strings to overwrite the values of our two properties.

```
class Person {

    var firstName = "John"
    var lastName = "Doe"

    init(first : String, last : String) {
        self.firstName = first;
        self.lastName = last;
    }
```

And now we can use that Init by just typing the name of the class, and then Parens and Xcode will help us with autoComplete.

```
var bff = Person(first: "Russell", last: "Wilson")
println(bff.firstName)
```

Inheritance

- Classes can Inherit from other classes. When a Class inherits from another class, it inherits all of its methods and properties.
- The inheriting class is known as the subclass, and the class it inherits from is known as the super class.
- Inheritance is unique to Classes, it is not available on primitives and structs.
- When a subclass wants to redefine a method or property it inherits from a superclass, it is called ‘overriding’. We will look at this more in depth later on.
- Any class that does not inherit from another class is called a ‘base class’. There is no base class in Swift, and in Objective-C the base class is NSObject.
- All of Apple’s frameworks rely heavily on inheritance.

Inheritance

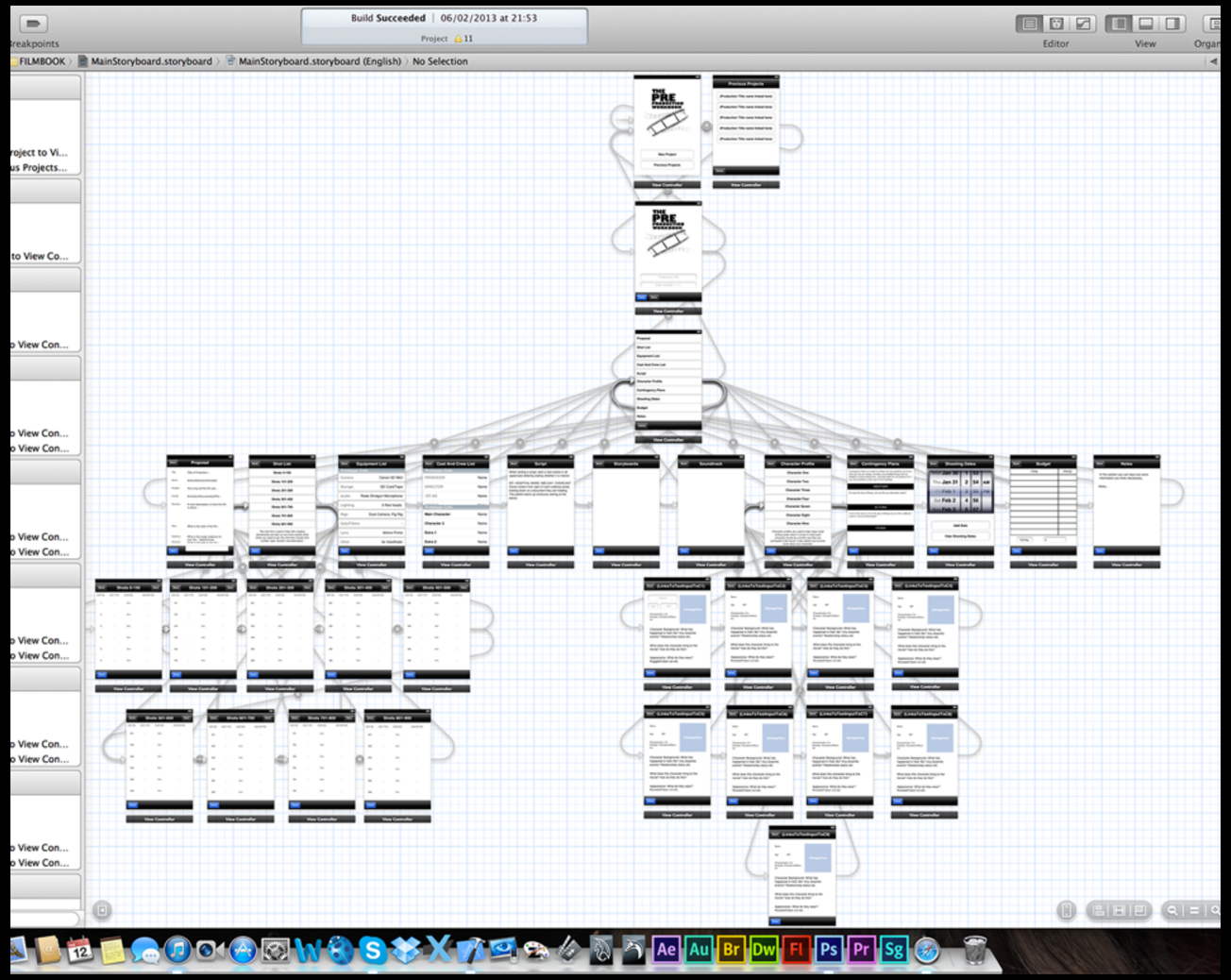
- Here we have a Librarian class that is a sub-class of the Person Class.
- Librarian can not only define it's own properties, but it has access to its super class's properties and methods as well.

```
class Librarian : Person {

    var lovesBooks = true

    func sayLastNameThreeTimes() {
        println(self.lastName)
        println(self.lastName)
        println(self.lastName)
    }

}
```

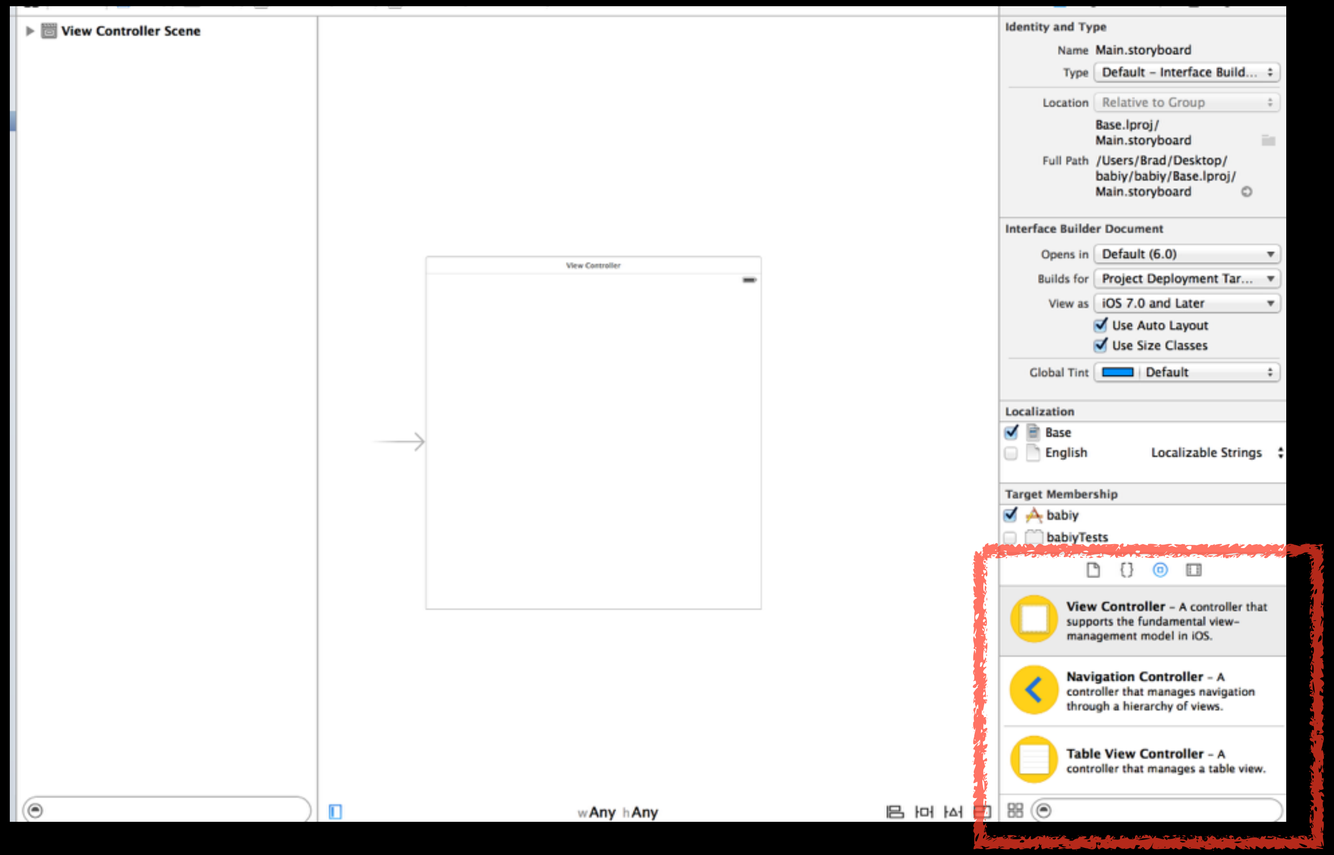
Storyboard

- “A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens”
- The storyboard is composed of a sequence of scenes, and each scene is a view controller, or one screen in your app.
- Scenes can be connected with segues to create transitions.
- Under the hood, the storyboard is just an XML file that describes everything you drag out onto your storyboard.

Storyboard


Build Succeeded | 06/02/2013 at 21:53
FILMBOOK
MainStoryboard.storyboard
MainStoryboard.storyboard (English)
No Selection

Storyboard



Drag things out of the object library and place them in your storyboard.

Git

What is Git?

- Git is an open source **version control** system.
- Git is an application you install on your computer.
- Git itself is not Github, Github is a hosting service for git repositories. Think of Github as The Cloud.
- type git —version into terminal to see if you have git installed and which version you have.

What is version control?

- Version control is a system that keeps track of changes to a file or a group of files over time so you can retrieve specific versions of the file(s) later.
- Git is considered a distributed version control system (DVCS) because any clients who check out a repository have a fully mirrored repository with all prior file histories. They essentially have a full backup.
- So, if any server or client dies, they can just check out the repo from another client or server who they were collaborating with and receive the full restore.
- A repo, or repository, is simply a place where the history of your work is stored.

Git Basics

- There are really 2 ways to get a git project on your computer: import an existing directory into git, or clone an existing repository from another server. We will talk about cloning later.
- git init - Creates an empty git repository or reinitializes an existing one in the directory you are currently in.
- After running git init, there will be a hidden .git file in the directory you are in. type ls -a to list all files including hidden files. You will see the .git file.
- After running git init in a directory, you now conceptually have a git project on your computer. So lets look at the anatomy of a git project.

Demo