

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-14-resources` repo to your computer:
  - Open the terminal
  - `cd` to the `Documents/JSD/JS-SF-14-resources` directory
  - Type `git pull` and press **return**
2. In your code editor, open the following folder:  
`Documents/JSD/JS-SF-14-resources/03-conditionals-functions`

---

**JAVASCRIPT DEVELOPMENT**

---

# **CONDITIONALS & FUNCTIONS**

# LEARNING OBJECTIVES

At the end of this class, you will be able to

- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow.
- Differentiate among `true`, `false`, `truthy`, and `falsy`.
- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values

# **AGENDA**

- Comparison operators
- Logical operators
- Conditional statements
- Functions

---

## CONDITIONALS & FUNCTIONS

---

# WEEKLY OVERVIEW

### WEEK 3

Conditionals & Functions / Scope & objects

### WEEK 4

Slack Bot Lab / JSON & Intro to DOM

### WEEK 5

Intro to jQuery / Events & APIs

## **EXIT TICKET QUESTIONS**

1. `array.map` is new to me and curious how it works
2. I get the basics, but hard for me to write the more slick code
3. what does `'=>'` do and how do I use it?
4. the function parameters

---

**CONDITIONALS & FUNCTIONS**

---

# **HOMEWORK REVIEW**



# HOMEWORK — GROUP DISCUSSION

---



## EXERCISE

### TYPE OF EXERCISE

---

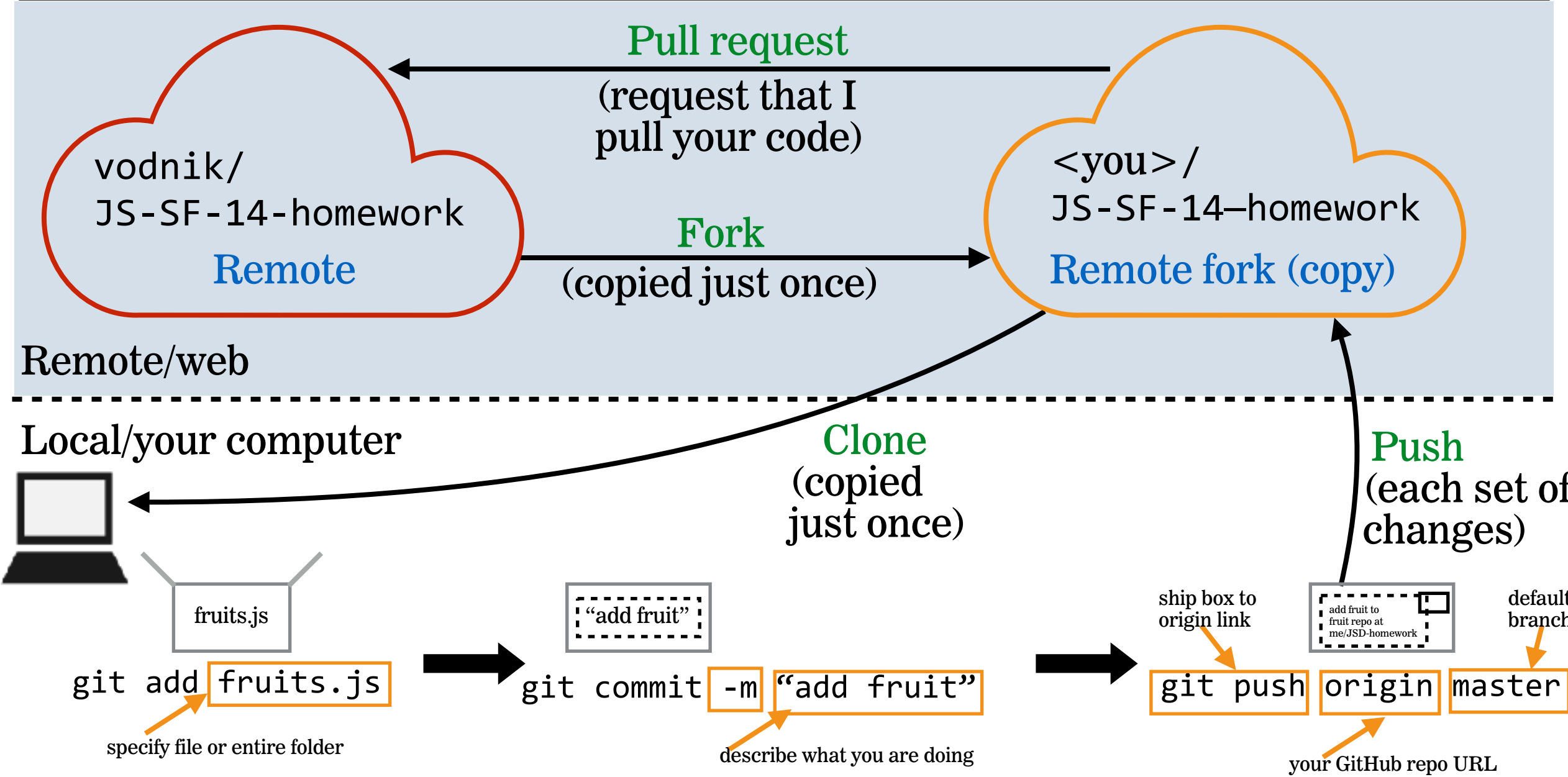
► Pairs

### TIMING

---

*5 min*

1. Take turns showing and explaining your code.
2. Share 1 thing you're excited about being able to accomplish.
3. Have each person in the group note 1 thing they found challenging for the homework. Discuss as a group how you think you could solve each problem.
4. Did you work on the Madlibs bonus exercise? Show your group what you did!

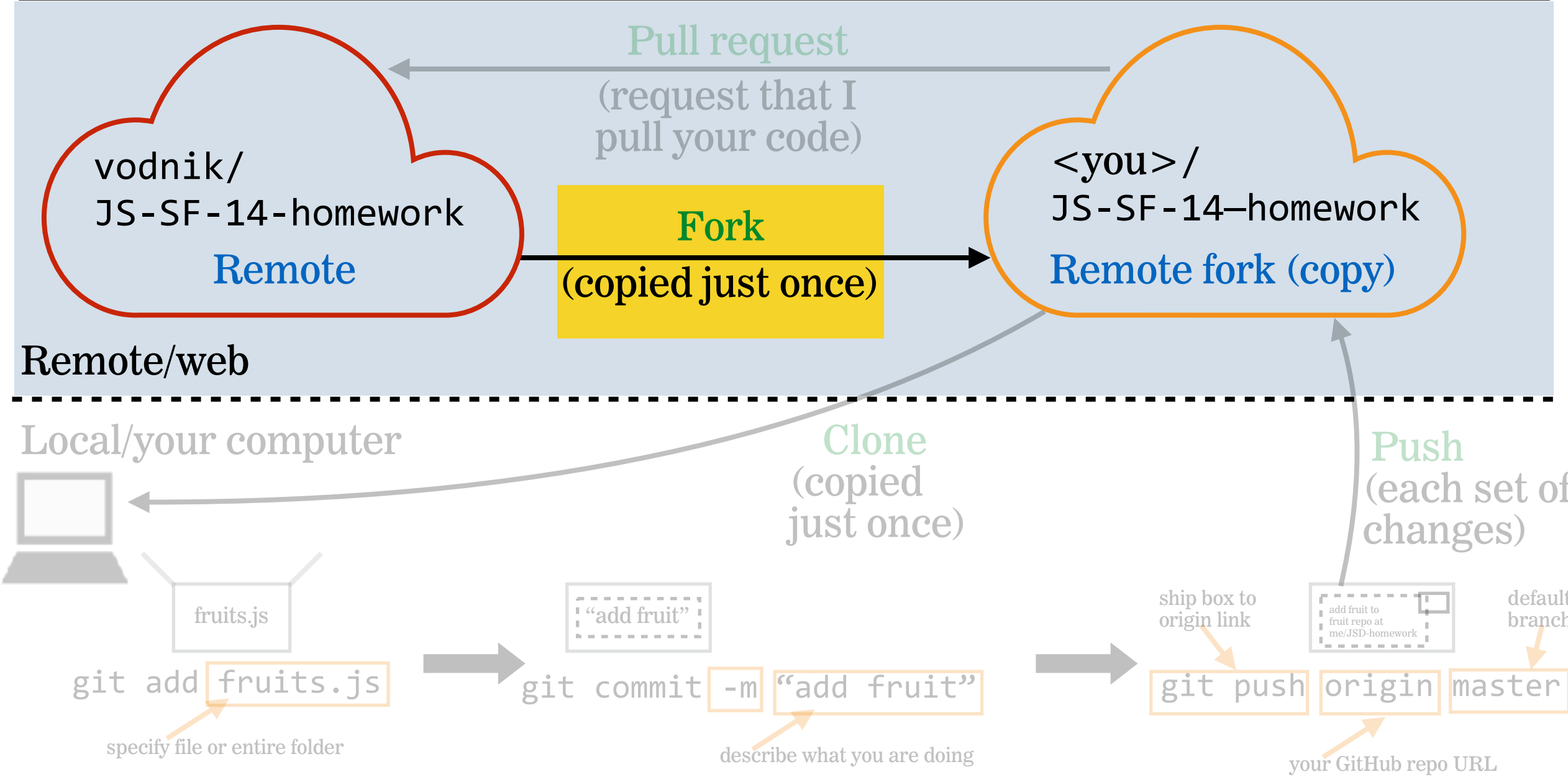


# **SUBMIT HOMEWORK: SETUP (ONE TIME ONLY)**

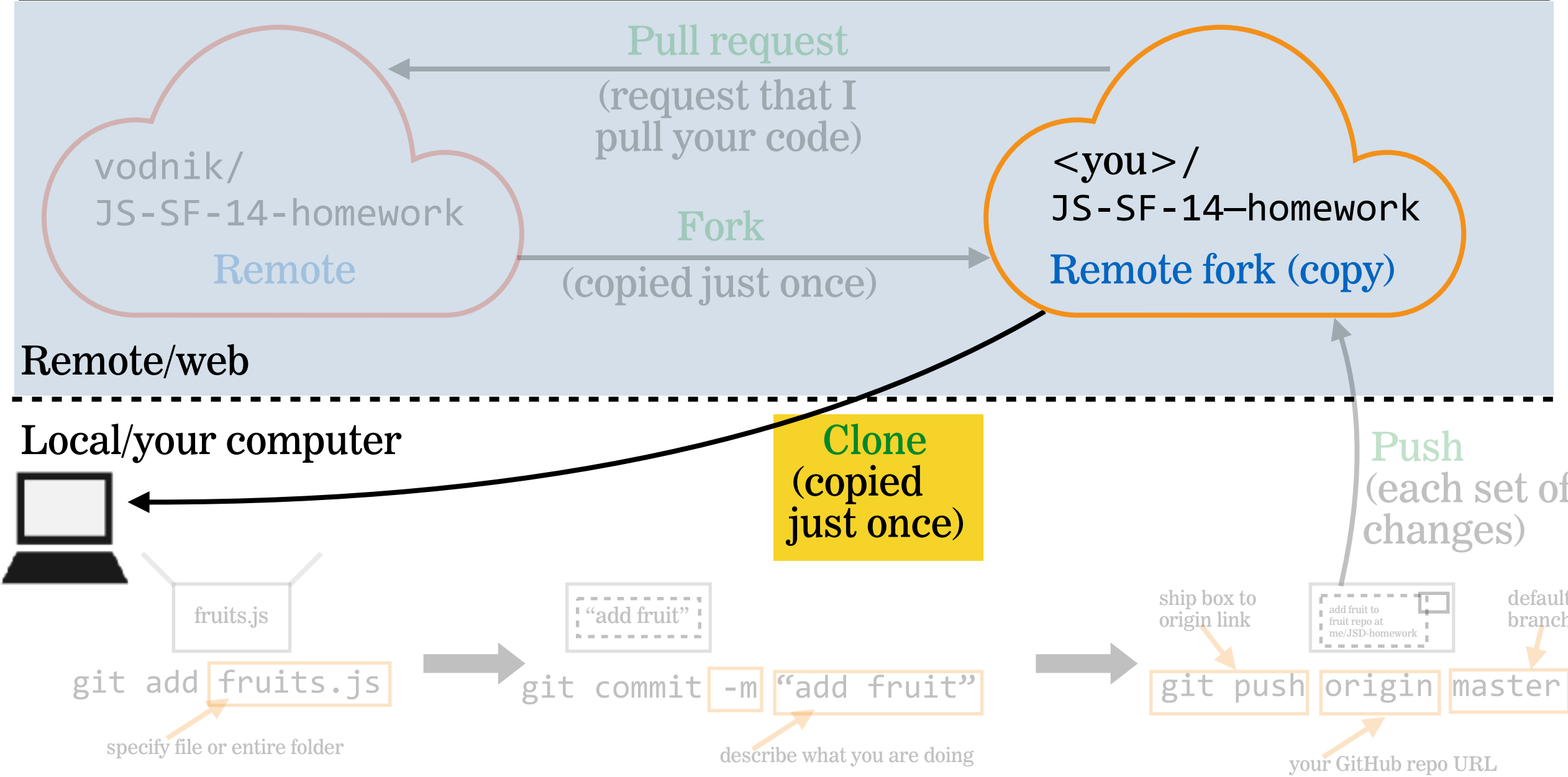
**On `github.com`:**

- Open <https://git.generalassemb.ly/vodnik/JS-SF-14-homework>
- Fork this repo to your GitHub account
- Clone your fork to your computer, within your JSD folder

# USING THE JS-SF-14-HOMEWORK REPO



# USING THE JS-SF-14-HOMEWORK REPO



# HOMework FOLDER LOCATION



JSD



JS-SF-14-homework



new folder for  
your clone of the  
homework repo



JS-SF-14-resources



`username.git.generalassemb.ly`

## **SUBMIT HOMEWORK: SETUP (CONTINUED)**

- Within your new **JS-SF-14-homework** folder, create a new subfolder and name it your first name, a hyphen, and your github name. For instance, Sasha's folder would be **Sasha-vodnik**.

# PERSONAL ASSIGNMENTS FOLDER LOCATION



JSD



JS-SF-14-homework



*firstname-username*



new folder for  
your completed  
assignments



JS-SF-14-resources



*username.git.generalassemb.ly*



# **SETUP DONE!**

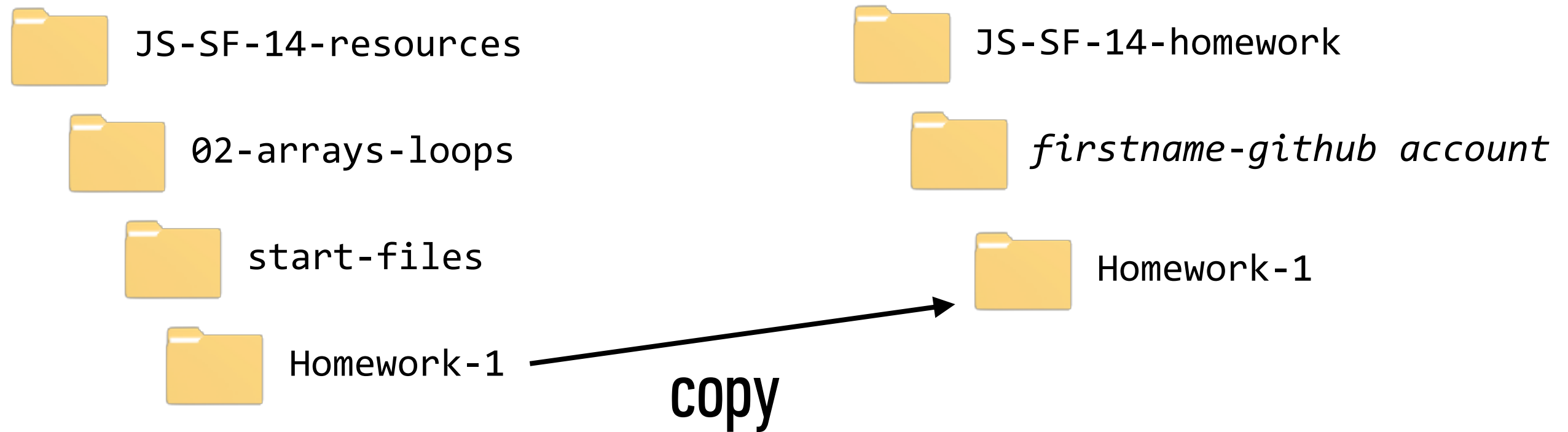
- Reminder: Now that you've completed the preceding setup, you never have to do it again!
- Each time you submit homework for the rest of this course, you'll repeat only the steps that follow.

# **SUBMIT HOMEWORK: STEP 1**

## **In Finder:**

- navigate to *firstname-username* folder (example: Sasha-vodnik)
- copy your completed Homework-1 folder from last Wednesday into your *firstname-username* folder.

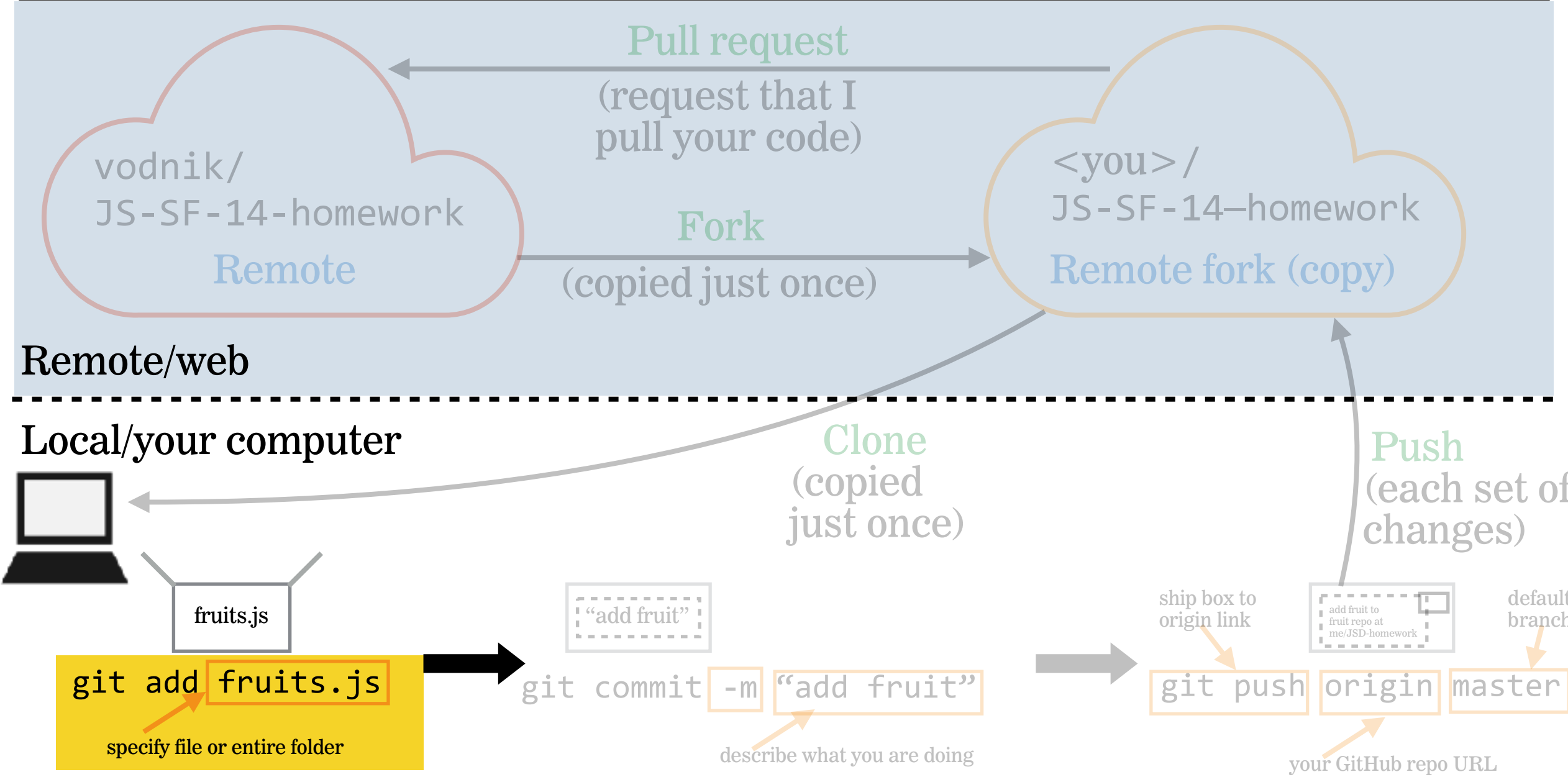
# SUBMIT HOMEWORK: STEP 1 ILLUSTRATION

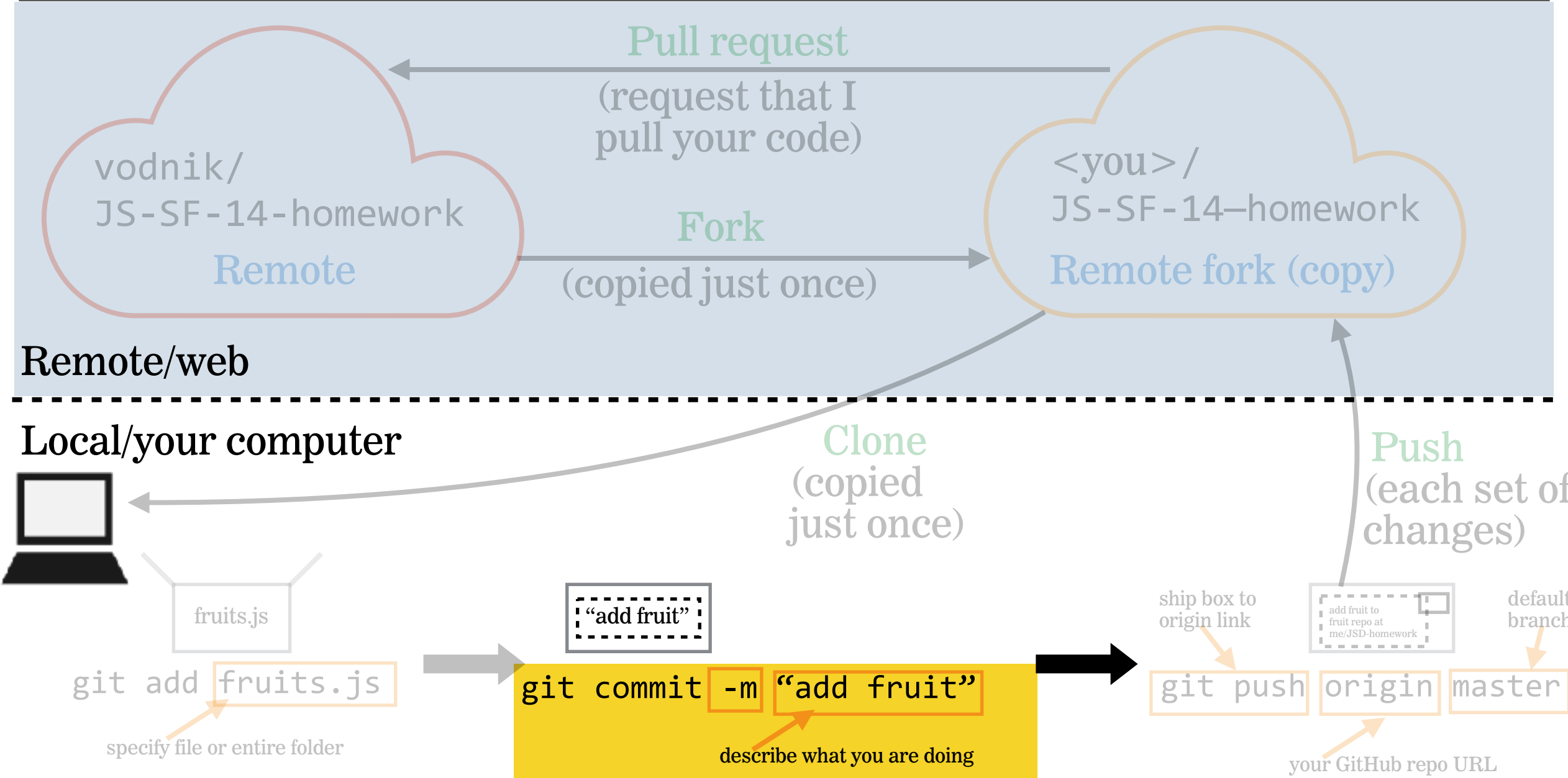


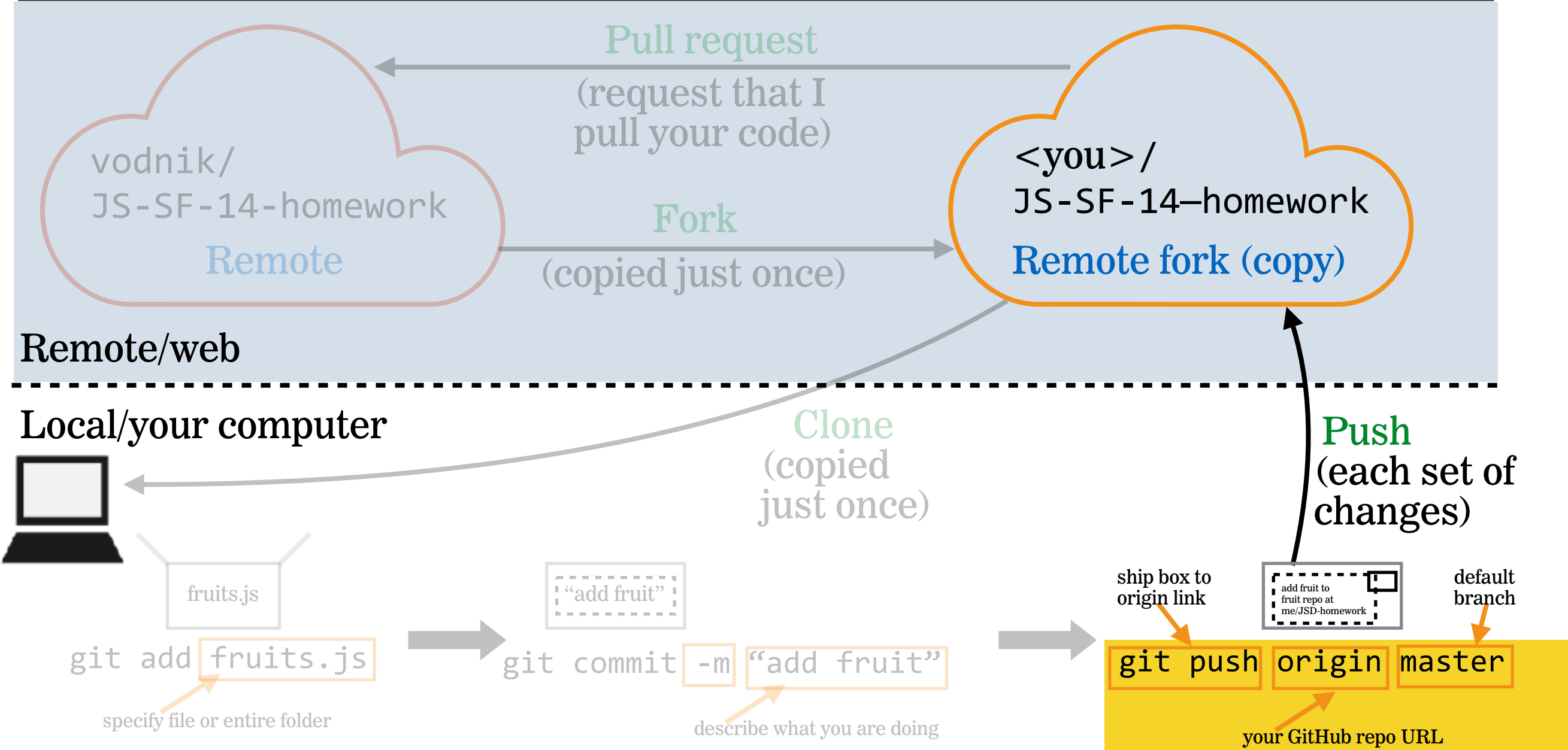
# **SUBMIT HOMEWORK: STEP 2**

## **In Terminal:**

- navigate to JS-SF-14-homework folder
- `git add .`
- `git commit -m "submitting Homework 1"`
- `git push origin master`





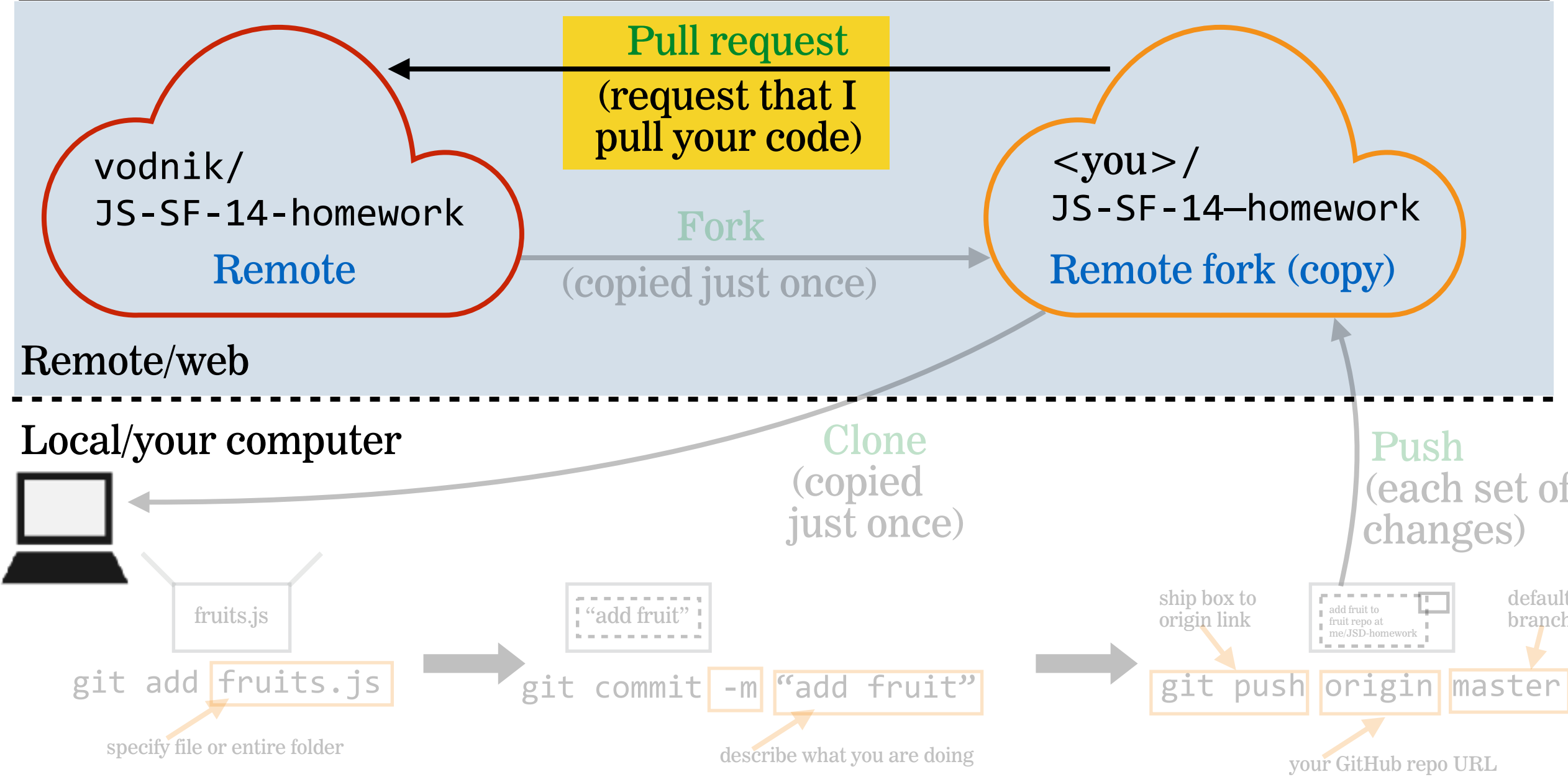


# **SUBMIT HOMEWORK: STEP 3**

## **In Browser:**

- Go to your fork of JS-SF-14-homework on `git.generalassemb.ly`
- click **New pull request**
- click **Create pull request**
- click **Create pull request** (again)





# **How to you decide what to have for dinner?**

- What factors do you consider?
- How do you decide between them?

# **CONDITIONALS**

# CONDITIONAL STATEMENTS

- Decide which blocks of code to execute and which to skip, based on the results of tests that we run
- Known as **control flow statements**, because they let the program make decisions about which statement should be executed next, rather than just going in order

# if STATEMENT

```
if (expression) {  
  code  
}
```

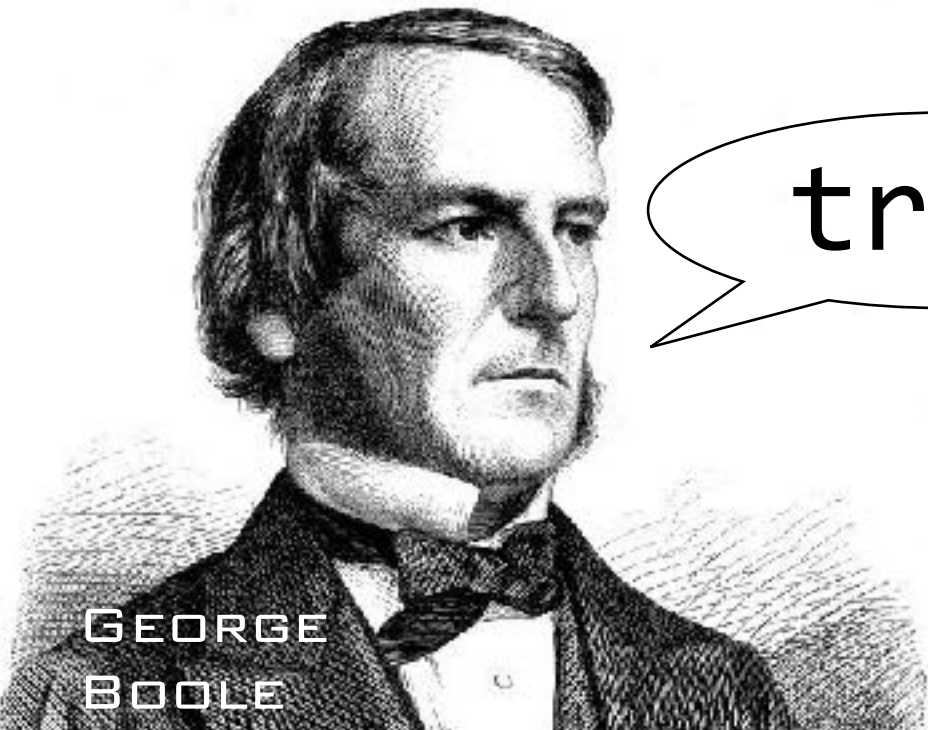


```
if (expression) { code }
```

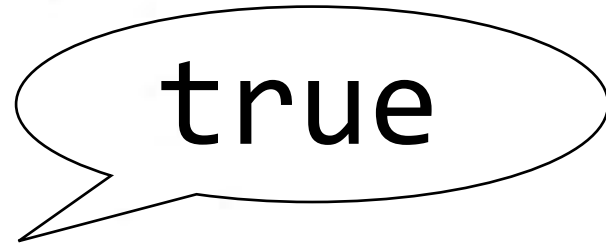


- JavaScript doesn't care about white space, so these are equivalent.
- **However**, putting block contents on a separate line is best practice for code readability.

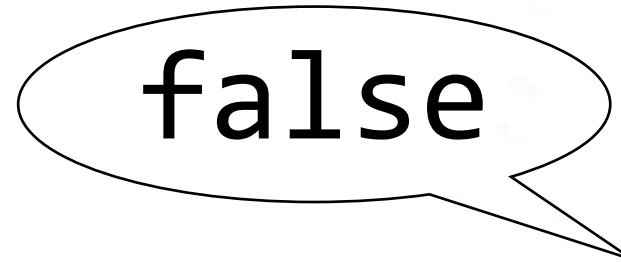
# BOOLEAN VALUES



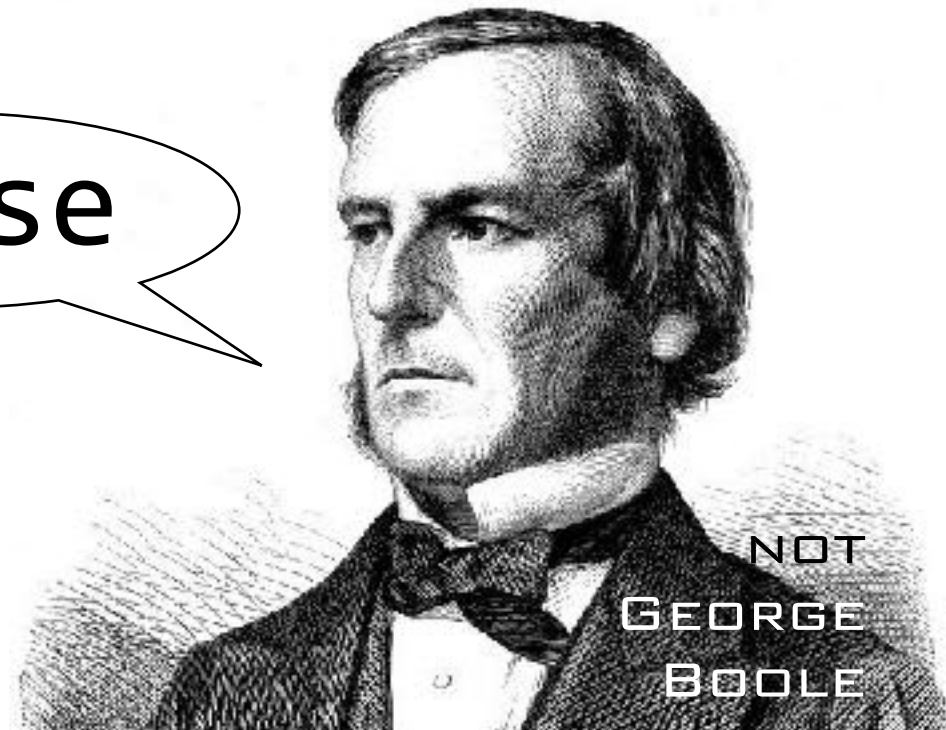
GEORGE  
BOOLE



true



false



NOT  
GEORGE  
BOOLE

# COMPARISON OPERATORS

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
===	strict equal (use this one)
==	coercive equal (AVOID)
!==	strict not equal (use this one)
!=	coercive not equal (AVOID)

# TYPE COERCION



I'm going to assume you meant this:

```
let total = "53";  
console.log(total == 53);
```

```
let total = 53;  
console.log(total == 53);
```

```
> true
```





# TYPE COERCION



Those are not  
the same.

```
let total = "53";  
console.log(total === 53);
```

```
let total = "53";  
console.log(total === 53);
```

```
> false
```



### if STATEMENT

```
let weather = "sunny";
```

```
if (weather === "sunny") {  
    console.log("Grab your sunglasses");  
}
```

### if/else STATEMENT

```
let weather = "sunny";
```

```
if (weather === "sunny") {  
    console.log("Bring your sunglasses");  
} else {  
    console.log("Grab a jacket");  
}
```

## else if STATEMENT

```
let weather = "sunny";
```

```
if (weather === "sunny") {  
    console.log("Bring your sunglasses");  
} else if (weather === "rainy") {  
    console.log("Take an umbrella");  
} else {  
    console.log("Grab a jacket");  
}
```

# TERNARY OPERATOR

*(expression) ? trueCode : falseCode;*

# TERNARY OPERATOR

- › Can produce one of two values, which can be assigned to a variable in the same statement

```
let name = (expression) ? trueCode : falseCode;
```

# BLOCK STATEMENTS

- Statements to be executed after a control flow operation are grouped into a block statement
- A block statement is placed inside braces

```
{  
    console.log("Grab your sunglasses.");  
    console.log("Enjoy the beach!");  
}
```

# LOGICAL OPERATORS

- Operators that let you chain conditional expressions

&&	AND	Returns true when both left and right values are true
	OR	Returns true when at least one of the left or right values is true
!	NOT	Takes a single value and returns the opposite Boolean value



# TRUTHY AND FALSY VALUES



## **FALSY VALUES**

- All of these values become `false` when converted to a Boolean:

`false`

`0`

`""`

`NaN`

`null`

`undefined`

- These are known as **falsy values** because they are equivalent to `false`

# TRUTHY VALUES

- All values other than `false`, `0`, `""`, `NaN`, `null`, and `undefined` become `true` when converted to a Boolean
- All values besides these six are known as **truthy values** because they are equivalent to `true`
- `'0'` and `'false'` are both **truthy**! (Why?)

# BEST PRACTICES

- Convert to an actual Boolean value
  - Adding ! before a value returns the *inverse* of the value as a Boolean
  - Adding !! before a value gives you the *original* value as a Boolean
- Check a value rather than a comparison



just use  
`if (!name)`



instead of  
`if (name === false)`

## TYPE CASTING

```
// Number() produces a number  
let total = Number(sum)
```

```
// String() produces a string  
let item = String(title)
```

```
// Boolean() produces a Boolean  
let canAttend = Boolean(rsvp)
```

# LAB — CONDITIONALS

---



## EXERCISE

### TYPE OF EXERCISE

---

‣ Pair

### LOCATION

---

‣ `starter-code > 1-ages-lab`

### TIMING

---

*15 min*

1. Write a program that outputs results based on users' age. Use the list of conditions in the `app.js` file.
2. BONUS 1: Rewrite your code to allow a user to enter an age value, rather than hard-coding it into your program. (Hint: Read up on the [window.prompt method](#).)
3. BONUS 3: Rewrite your code to use a [switch statement](#) rather than if and else statements.

# **FUNCTIONS**

# FUNCTIONS



### GROUP STEPS

Allow us to group a series of statements together to perform a specific task



### REUSABLE

We can use the same function multiple times



### STORE STEPS

Not always executed when a page loads.  
Provide us with a way to 'store' the steps needed to achieve a task.



## CONDITIONALS & FUNCTIONS

**DRY =  
DON'T  
REPEAT  
YOURSELF**



# FUNCTION DECLARATION SYNTAX

```
function name(parameters) {  
    // do something  
}
```

# FUNCTION DECLARATION EXAMPLE

```
function speak() {  
    console.log("Hello!");  
}
```

# FUNCTION EXPRESSION SYNTAX

```
let name = function(parameters) {  
    // do something  
};
```

# FUNCTION EXPRESSION EXAMPLE

```
let speak = function() {  
  console.log("Hello!");  
};
```

# ARROW FUNCTION SYNTAX

```
let name = (parameters) => {  
  // do something  
};
```

# ARROW FUNCTION EXAMPLE

```
let speak = () => {  
  console.log("Hello!");  
};
```

---

## CONDITIONALS & FUNCTIONS

---

### CALLING A FUNCTION

```
function pickADescriptiveName() {  
    // do something  
}
```

To run the function, we need to *call* it. We can do so like this:

```
pickADescriptiveName();
```

**pickADescriptiveName()** + parentheses



---

# EXERCISE — WRITING FUNCTIONS

---



## EXERCISE

### **KEY OBJECTIVE**

---

- Practice defining and executing functions

### **TYPE OF EXERCISE**

---

- Individual/paired

### **LOCATION**

---

- `starter-code` > `3-functions-exercise` (part 1)

### **EXECUTION**

---

*4 min*

1. Follow the instructions under Part 1

# **PARAMETERS**


### DOES THIS CODE SCALE?

```
function helloVal() {  
  console.log('hello, Val');  
}
```

```
function helloOtto() {  
  console.log('hello, Otto')  
}
```

### USING A PARAMETER

```
function sayHello(name) {  
  console.log('Hello ' + name);  
}
```



A green rectangular box labeled "parameter" has an arrow pointing to the word "name" in the function definition `sayHello(name)`.

```
sayHello('Val');
```



An orange rectangular box labeled "argument" has an arrow pointing to the string "Val" in the function call `sayHello('Val')`.

```
=> 'Hello Val'
```

```
sayHello('Otto');
```

```
=> 'Hello Otto'
```

# USING MULTIPLE PARAMETERS

multiple parameter names  
separated by commas

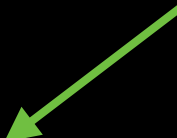


```
function sum(x, y, z) {  
    console.log(x + y + z)  
}
```

```
sum(1, 2, 3);  
=> 6
```

# USING DEFAULT PARAMETERS

default value to set for parameter if no argument is passed when the function is called



```
function multiply(x, y = 2) {  
  console.log(x * y)  
}
```

```
multiply(5, 6);
```

```
=> 30 // result of 5 * 6 (both arguments)
```

```
multiply(4);
```

```
=> 8 // 4 (argument) * 2 (default value)
```

# EXERCISE — READING FUNCTIONS

---



## EXERCISE

### KEY OBJECTIVE

---

- ▶ Given a function and a set of arguments, predict the output of a function

### TYPE OF EXERCISE

---

- ▶ Groups of 2 - 3

### LOCATION

---

- ▶ `starter-code` > `3-functions-exercise` (part 2)

### EXECUTION

---

*3 min*

1. Look at Part 2 A and B. Predict what will happen when each function is called.

# EXERCISE — READING FUNCTIONS

---



## EXERCISE

### KEY OBJECTIVE

---

- Create and call a function that accepts parameters to solve a problem

### TYPE OF EXERCISE

---

- Groups of 2 - 3

### LOCATION

---

- `starter-code` > `3-functions-exercise` (part 3)

### EXECUTION

---

*8 min*

1. See if you can write one function that takes some parameters and combines the functionality of the *makeAPizza* and *makeAVeggiePizza* functions.
2. BONUS: Create your own function with parameters. This function could do anything!



# EXERCISE — FUNCTIONS

---



## EXERCISE

### KEY OBJECTIVE

---

- Describe how parameters and arguments relate to functions

### TYPE OF EXERCISE

---

- Turn and Talk

### EXECUTION

---

*1 min*

1. Summarize why we would use functions in our programs. What purpose do they serve?
2. What is a parameter? What is an argument? How are parameters and arguments useful?

# THE `return` STATEMENT

# **return STATEMENT**

- Ends function's execution
- Returns a value — the result of running the function

# return STOPS A FUNCTION'S EXECUTION

```
function speak(words) {  
    return words;  
}
```

// The following statements will not run:

```
let x = 1;  
let y = 2;  
console.log(x + y);  
}
```

# console.log() vs return



`console.log()`

**VS**



`return`

- ▶ Write a value at any point in a program to the browser console
- ▶ Helpful for developer in debugging
- ▶ Not seen by user or used by app

- ▶ Sends a value back wherever the current statement was triggered
- ▶ Can use a function to get a value and then use that value elsewhere in your app
- ▶ Does not appear in the console unless you're executing commands there

## CONDITIONALS & FUNCTIONS

# return in action

call `sum()` function,  
passing 3 and 4 as  
arguments

```
let z = sum(3,4);
```

with `x=3` and `y=4`,  
return the result  
of `x + y`, which is 7

```
function sum(x,y) {  
  return x + y;  
}
```

```
z = 7
```

# EXERCISE — FUNCTIONS LAB

---



## EXERCISE

### KEY OBJECTIVE

---

- Create and call a function that accepts parameters to solve a problem

### TYPE OF EXERCISE

---

- Individual or pair

### LOCATION

---

- `starter-code > 4-price-calculator`

### EXECUTION

---

*15 min*

1. Write code to calculate a customer's total cost in dollars based on product price, tax rate, shipping cost, and the currency they're using for the purchase (dollars or euros).
2. BONUS 1: Convert your function to assume a currency of "dollar" by default.
3. BONUS 2: Convert your code to use arrow functions.

# **Exit Tickets!**

**(Class #3)**



## **LEARNING OBJECTIVES – REVIEW**

- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow.
- Differentiate among `true`, `false`, `truthy`, and `falsy`.
- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values

# **NEXT CLASS PREVIEW**

## **Scope & hoisting**

- Determine the scope of local and global variables
- Create a program that hoists variables

# Q&A