GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-8` repo to your computer:
   - Open the terminal
   - `cd` to the `JSD/JS-SF-8-resources` directory
   - Type **`git pull`** and press **return**
2. In your code editor, open the following folder: `JSD/JS-SF-8-resources/03-loops-conditionals/starter-code`

LOOPS AND CONDITIONALS

LEARNING OBJECTIVES

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

- Build iterative loops using `for` and `forEach` statements.
- Iterate over and manipulate values in an array.
- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow based on Boolean tests.
- Differentiate among true, false, truthy, and falsy.

AGENDA

- Loops
- Comparison operators, logical operators, & conditional statements

WEEKLY OVERVIEW

WEEK 3	Loops & Conditionals / Functions & Scope
WEEK 4	Slackbot Lab / Objects & JSON
WEEK 5	Intro to the DOM / Intro to jQuery

EXIT TICKET QUESTIONS

1. array maps, Array Helper/Iterator Methods
2. some of the syntax around foreach.function(el)
3. What are we doing with the consoleLog? (at the end of class)
4. How to store data back into an array once you've manipulated it.
5. Why .length doesn't have ()s. Why let is better than var. When to use .forEach() vs. a for loop.
6. more time in class to go through the assignment together would be great
7. A lot of this material felt like review from the prework, and this material was taught as if we were seeing it for the first time.

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

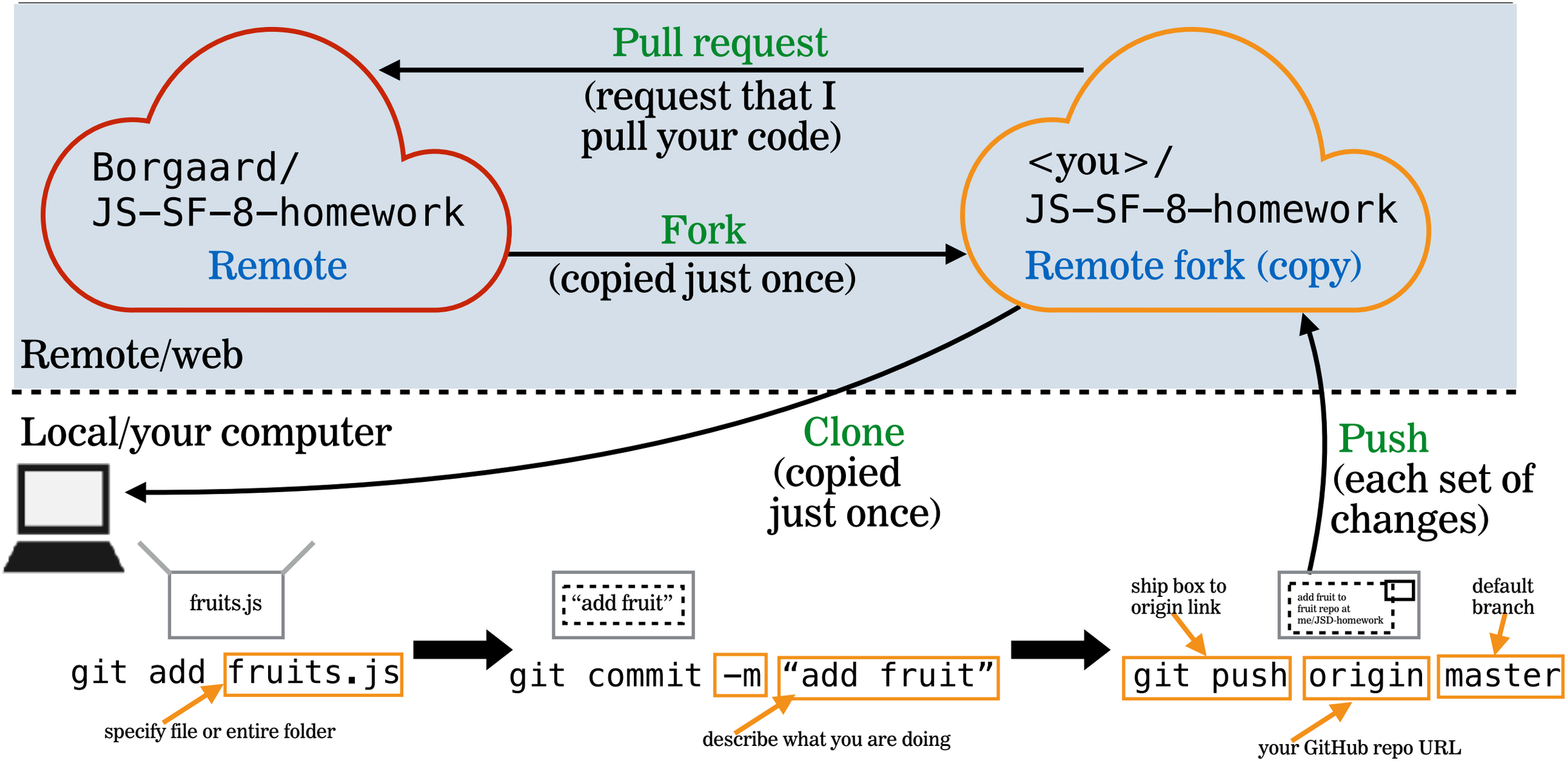
- Groups of 3

TIMING

10 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 exercise? Show your group what you did!

USING THE JS-SF-8-HOMEWORK REPO



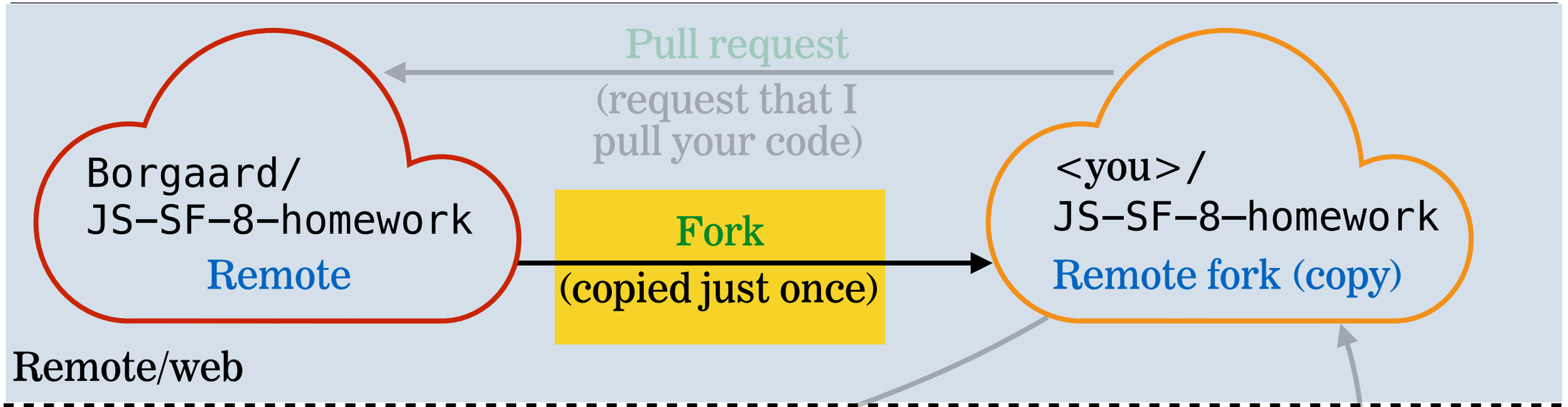
SUBMIT HOMEWORK: SETUP (ONE TIME ONLY)

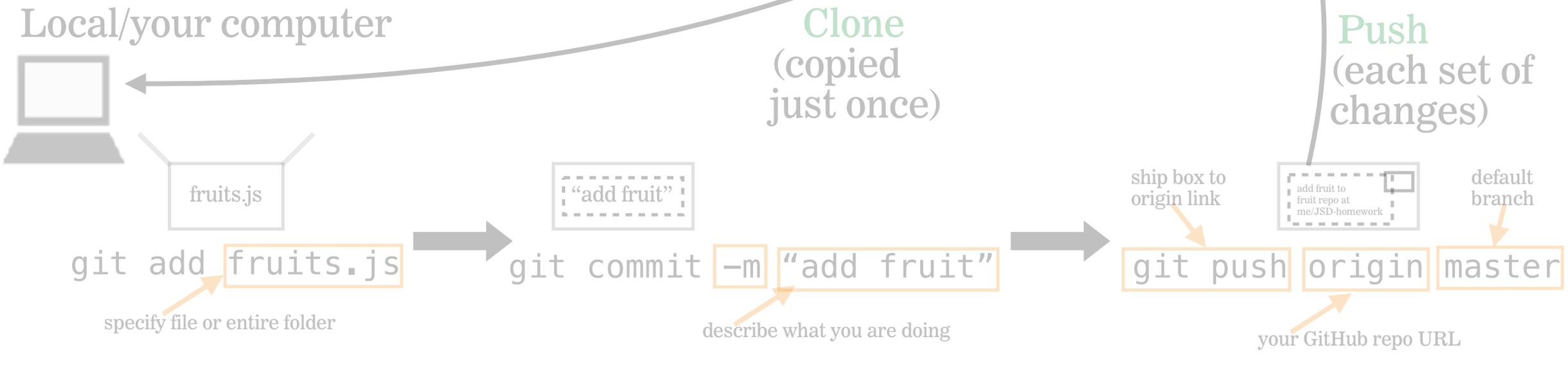
- Last week, we cloned the JS-SF-8-homework repo. We need to fork it instead, so navigate to the JSD folder, then rename the `JS-SF-8-homework` folder as `OLD-JS-SF-8-homework`.

On github.com:

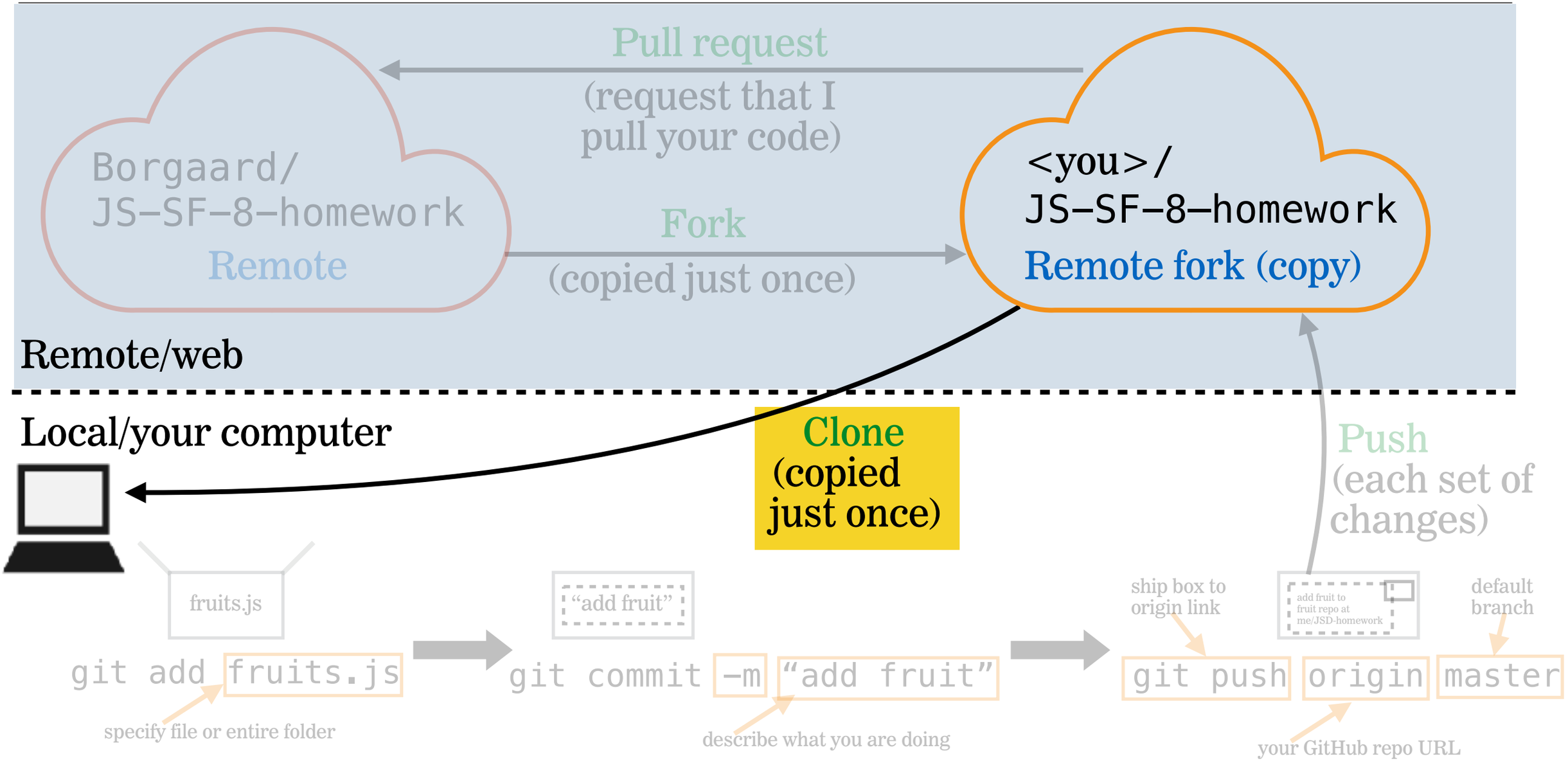
- Open Borgaard/JS-SF-8-homework
- Fork this repo to your GitHub account
- Clone your fork to your computer, within your JSD folder

USING THE JS-SF-8-HOMEWORK REPO





USING THE JS-SF-8-HOMEWORK REPO



SUBMIT HOMEWORK: SETUP (CONTINUED)

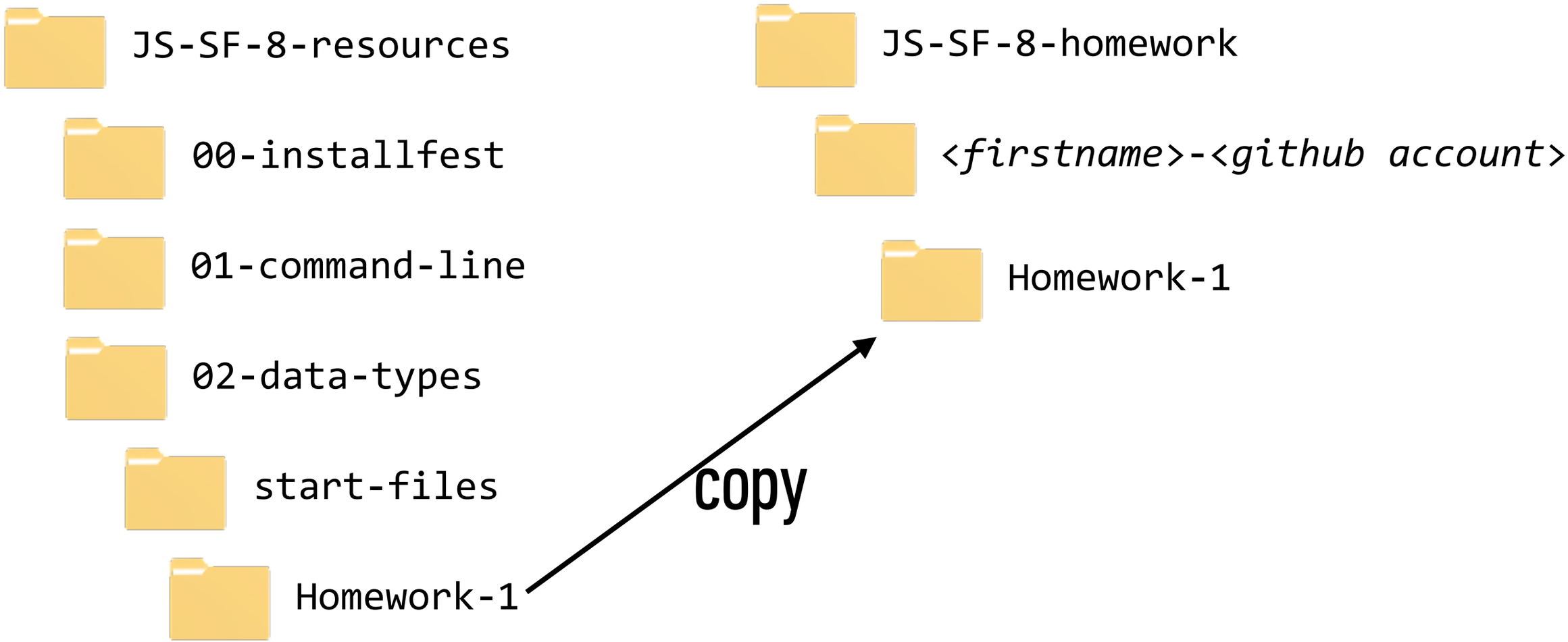
- Within your new JS-SF-8-homework folder, create a new subfolder and name it your first name, a hyphen, and your github name. For instance, Sasha's folder would be `Sasha-svodnik`.

SUBMIT HOMEWORK: STEP 1

In Finder:

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

SUBMIT HOMEWORK: STEP 1 ILLUSTRATION

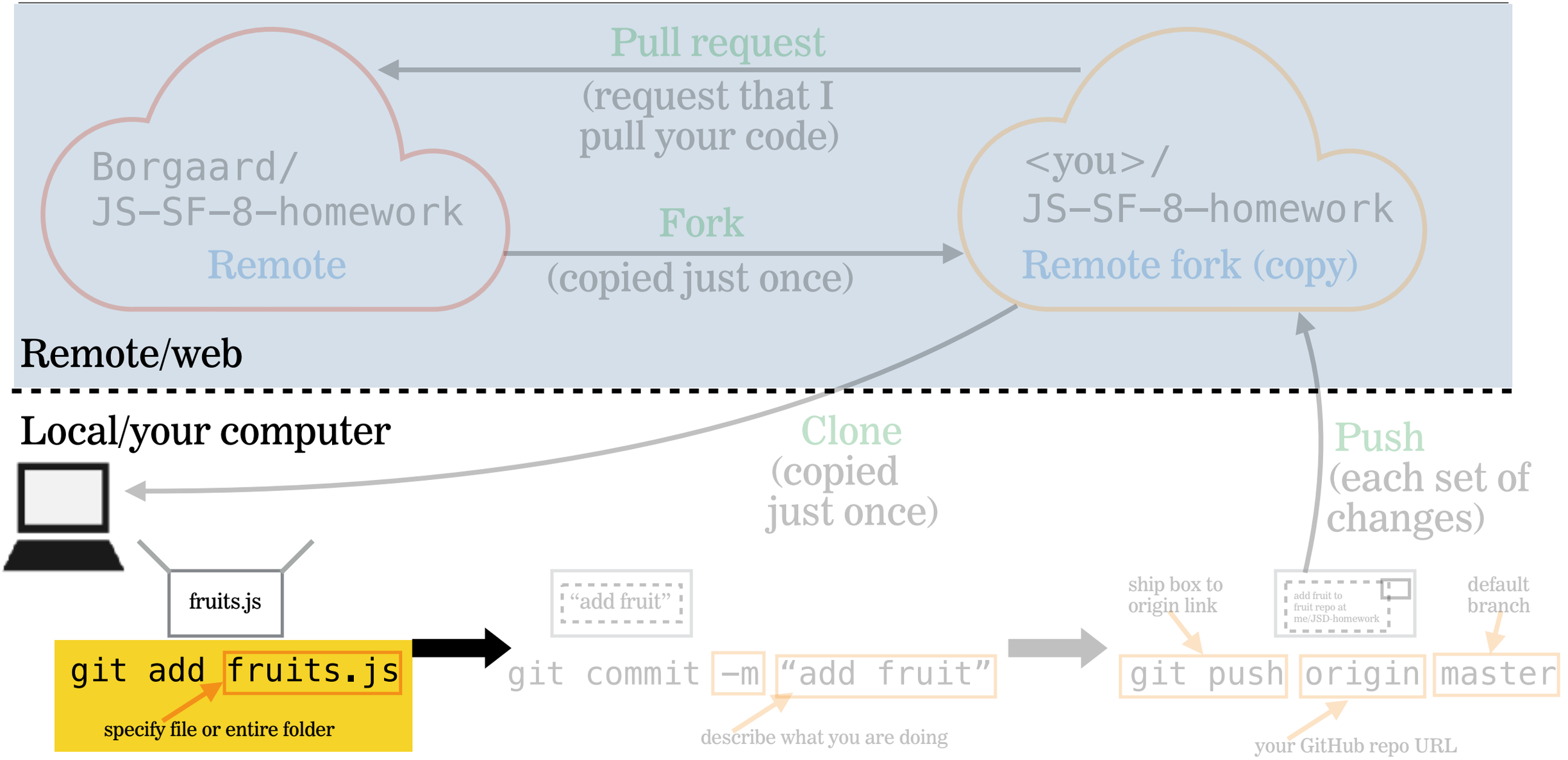


SUBMIT HOMEWORK: STEP 2

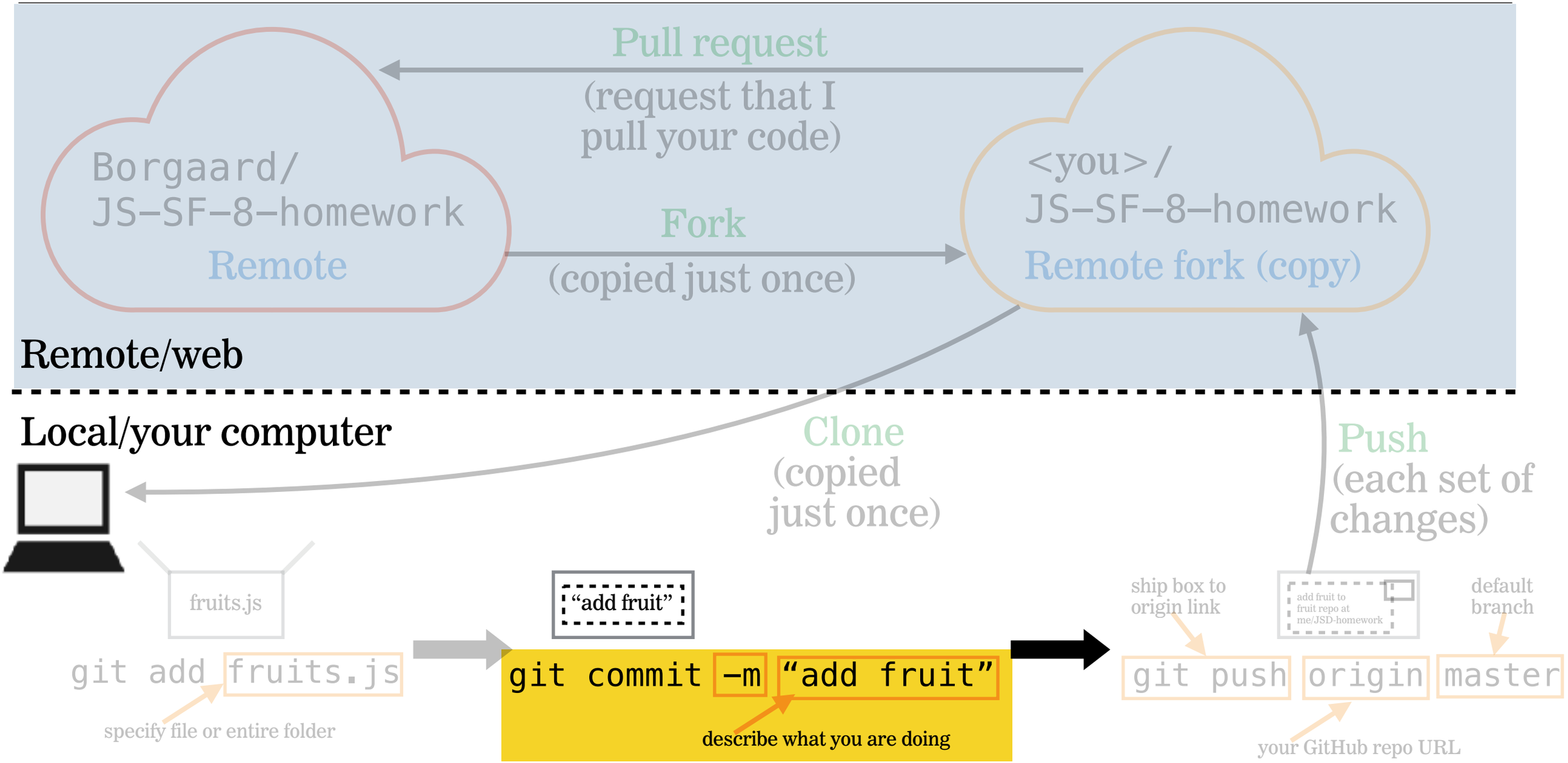
In Terminal:

- navigate to *`firstname-username`* folder
- `git add .`
- `git commit -m “submitting homework 1”`
- `git push origin master`

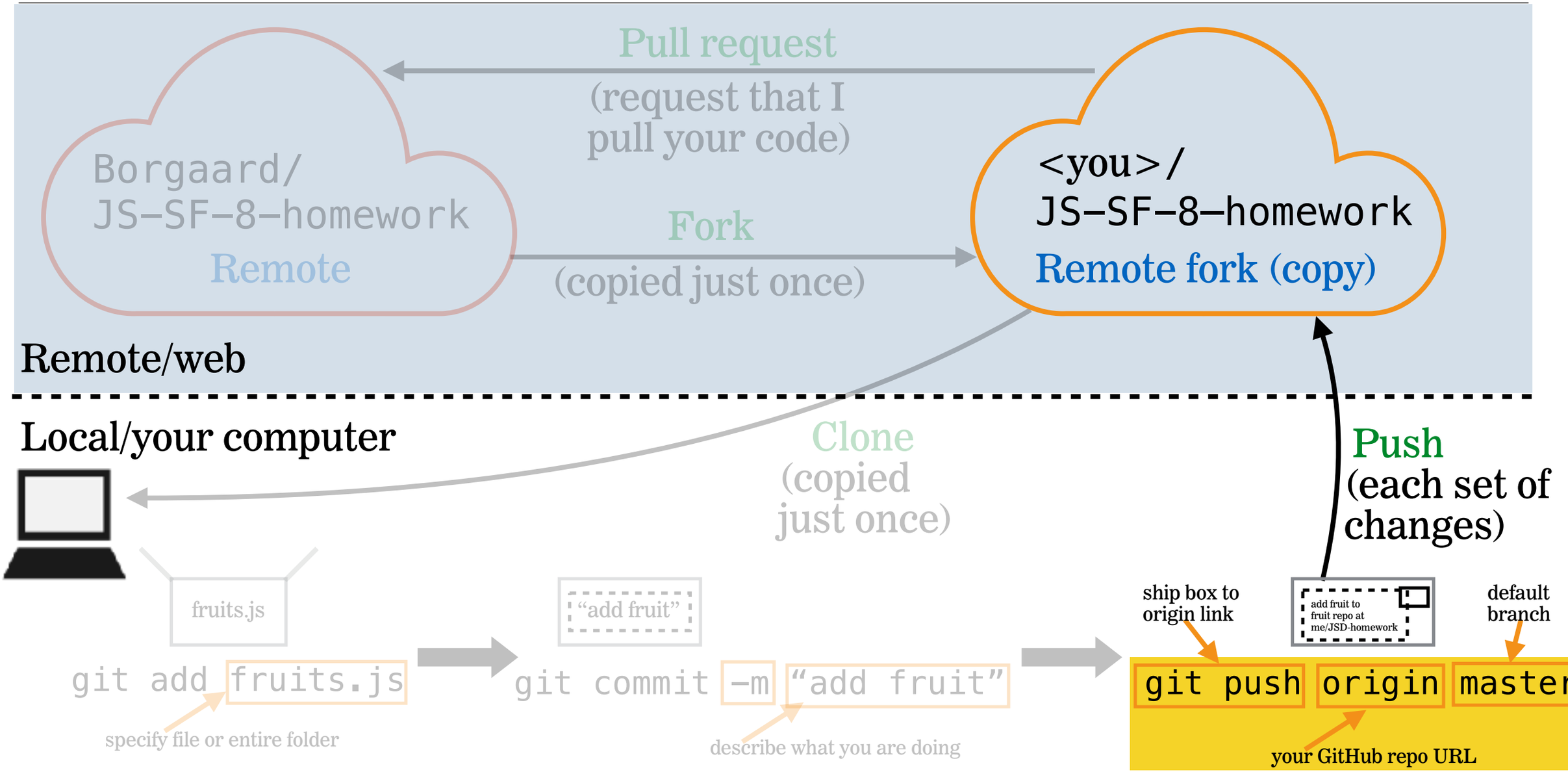
USING THE JS-SF-8-HOMEWORK REPO



USING THE JS-SF-8-HOMEWORK REPO



USING THE JS-SF-8-HOMEWORK REPO

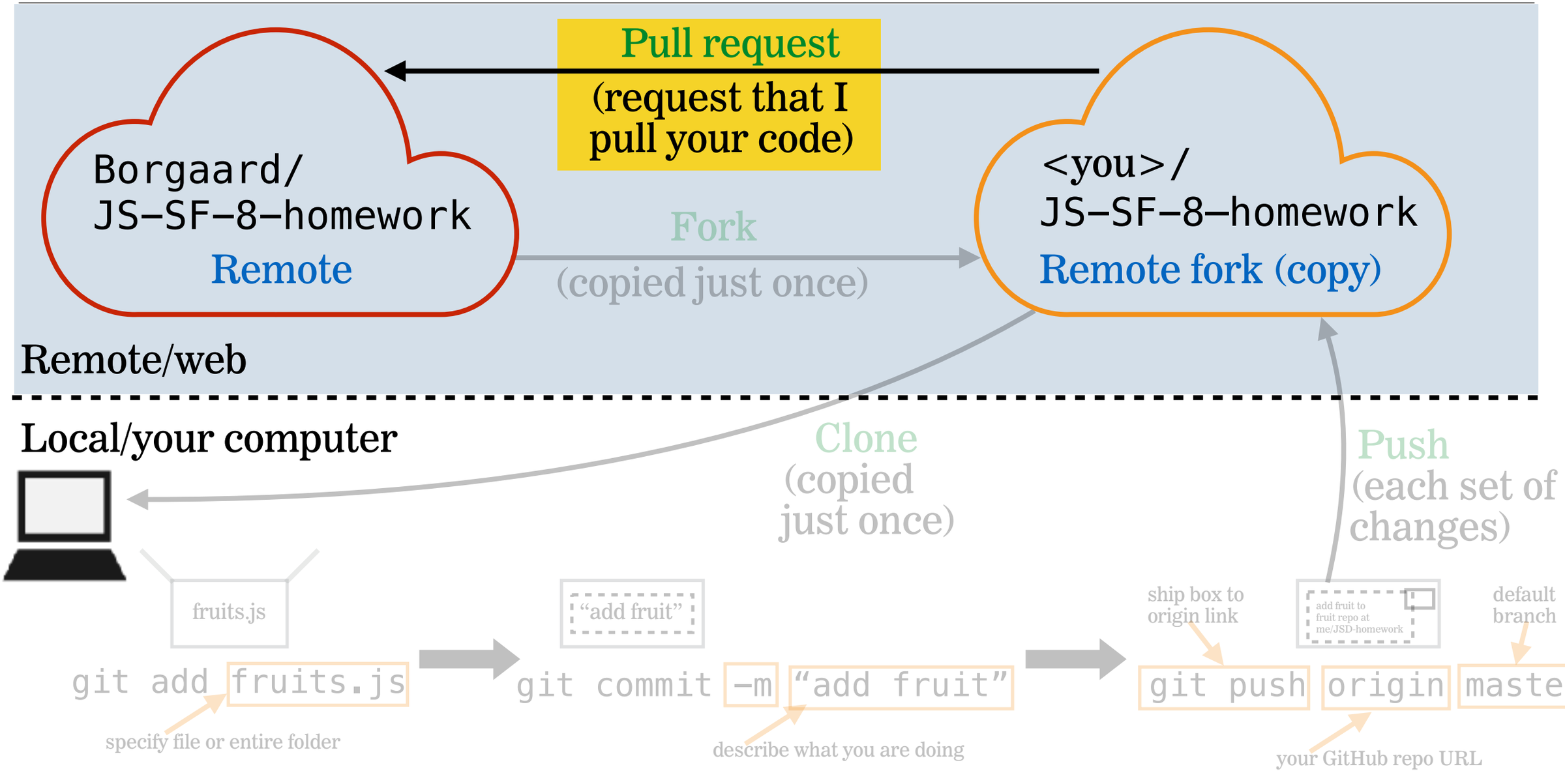


SUBMIT HOMEWORK: STEP 3

In Browser:

- Go to your fork of `JS-SF-8-homework` on github.com
- click **New pull request**
- click **Create pull request**
- click **Create pull request** (again)

USING THE JS-SF-8-HOMEWORK REPO



How to you decide what to have for dinner?

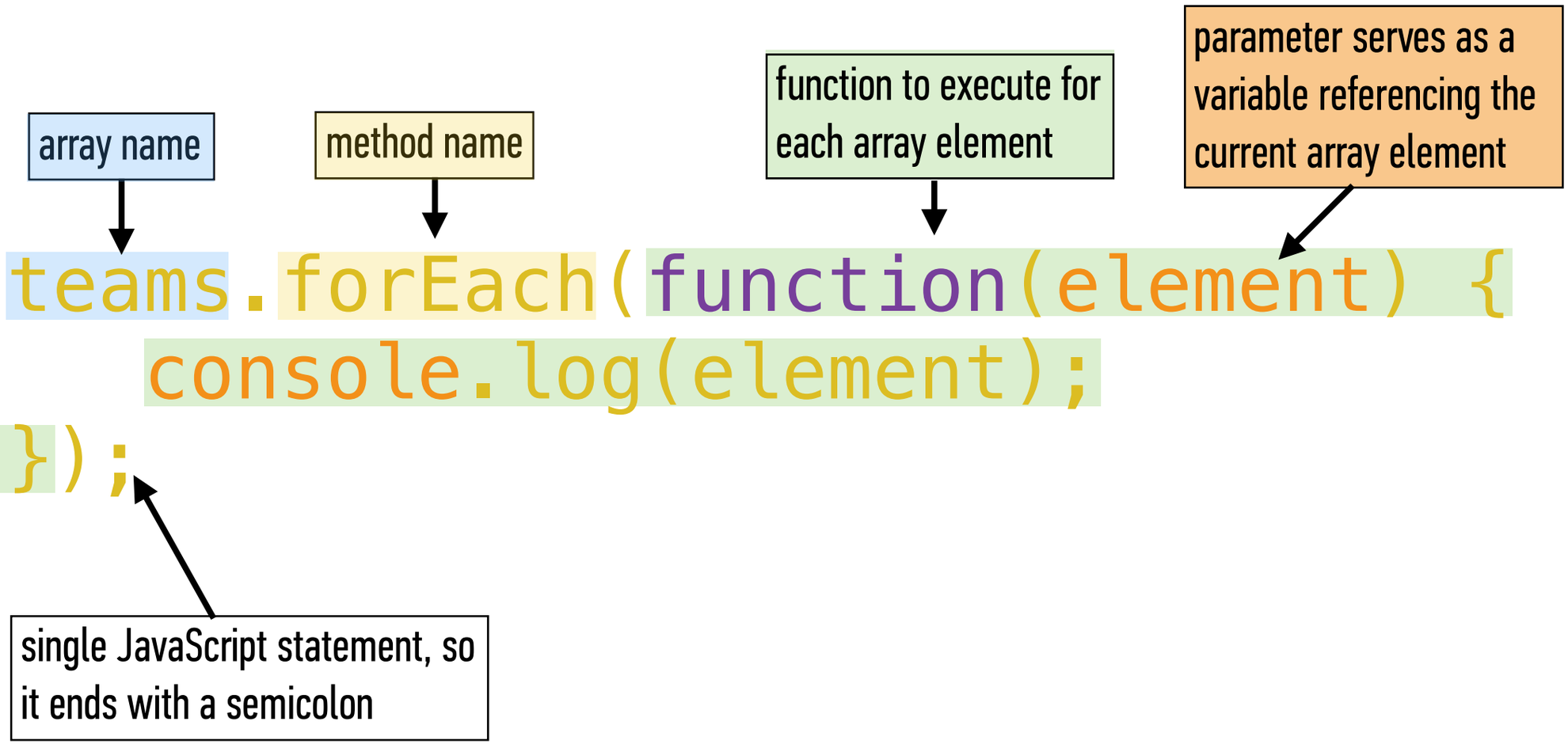
- What factors do you consider?
- How do you decide between them?

LOOPS

ITERATING

Going through the same process with a bunch of items, one at a time

forEach()

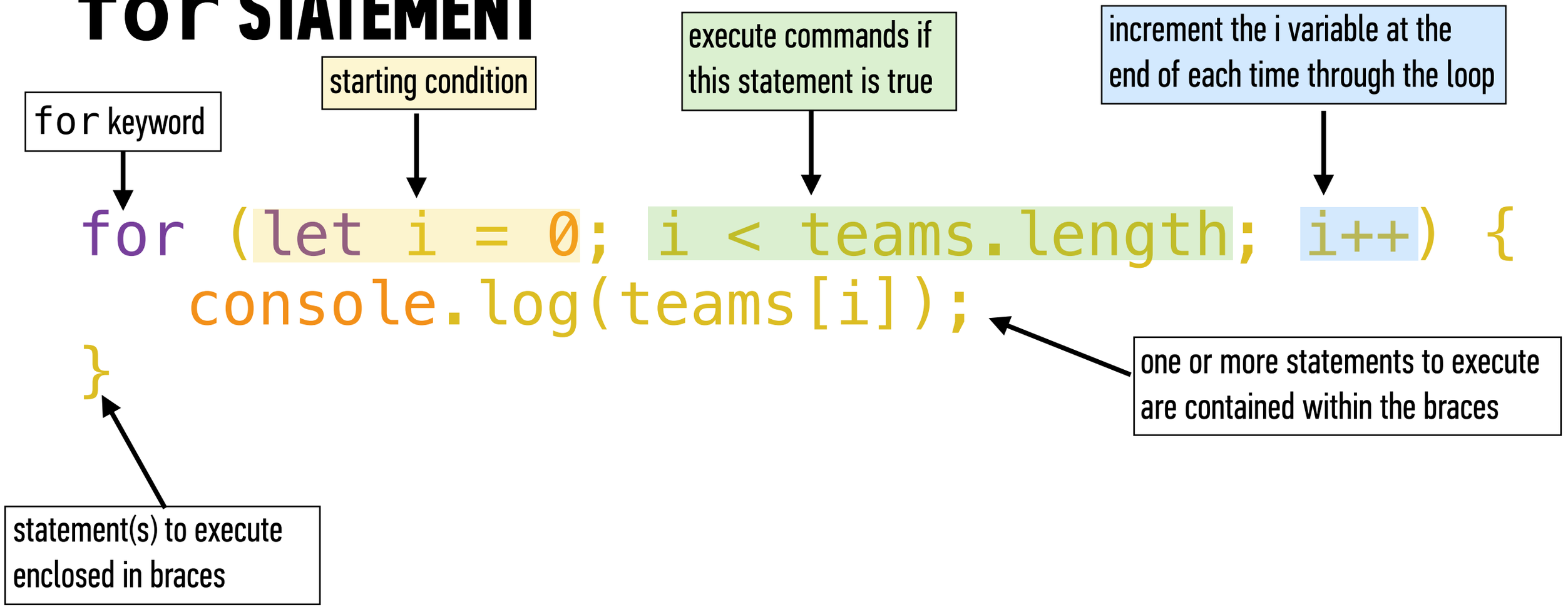

array name
method name
function to execute for each array element
parameter serves as a variable referencing the current array element
teams.forEach(function(element) {
console.log(element);
});
single JavaScript statement, so it ends with a semicolon

forEach() EXAMPLE

```
let teams = ['Bruins', 'Bears', 'Ravens', 'Ducks'];

teams.forEach(function(element) {
    console.log(element);
});
```

`for` STATEMENT



`for` STATEMENT

```
let fruits = ['apples', 'oranges', 'bananas'];

for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
});
```

result in console:

```
< "apples"
< "oranges"
< "bananas"
```

LAB — FOR LOOPS

EXERCISE

TYPE OF EXERCISE

- Individual / Pair

LOCATION

- `starter-code > 1-exponent-lab`

TIMING

15 min

1. Write code that creates a `for` loop that calculates 2 to a given power, and console.logs each step of the calculation. (Full instructions in the `app.js` file.)
2. BONUS 1: Rewrite your code to allow a user to enter the exponent value, rather than hard-coding it into your program. (Hint: Read up on the window.prompt method.)
3. BONUS 2: Rewrite your code to use a while loop rather than a `for` loop.
4. BONUS 3: Rewrite your code to use a do/while loop rather than a `for` loop or `while` loop.

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

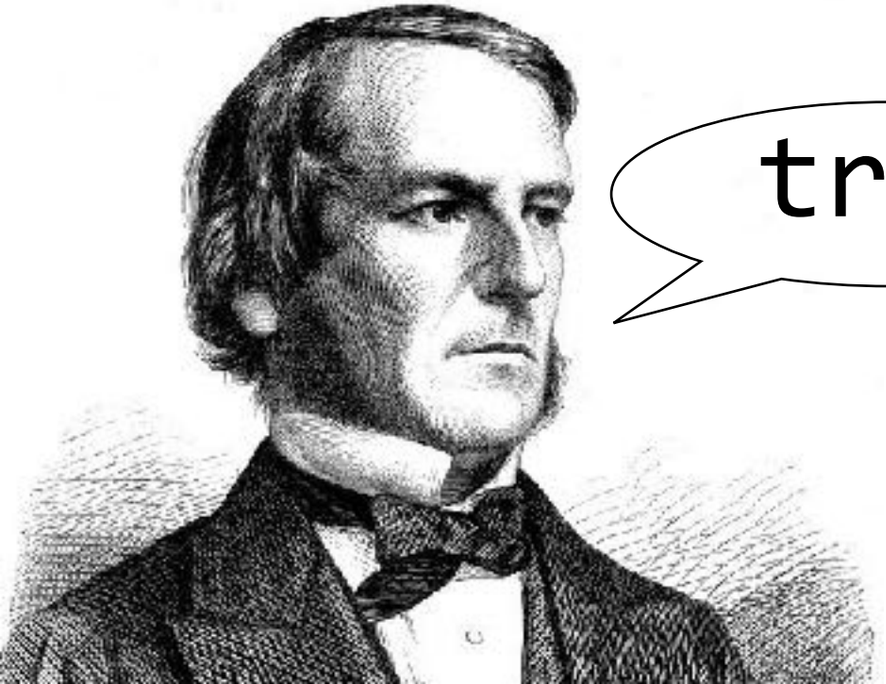
`if` STATEMENT

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

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

BOOLEAN VALUES

- A separate data type
- Only valid values are `true` or `false`
- Named after George Boole, a mathematician



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

- JavaScript “feature” that attempts to make it possible to run a comparison operation on two objects of different data types
- Results are sometimes unpredictable
- == and != use coercion if necessary to arrive at an answer — avoid them
- === and !== do not use coercion — best practice is to use these rather than the coercive operators

`if` STATEMENT

```
let weather = "sunny";

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

`if/else` STATEMENT

```
var weather = "sunny";

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

`else if` STATEMENT

```
var 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

- A compact `if/else` statement on a single line
- “ternary” means that it takes 3 operands

TERNARY OPERATOR

```
(expression) ? trueCode : falseCode;
```

TERNARY OPERATOR

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

```
var 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!");
}
```

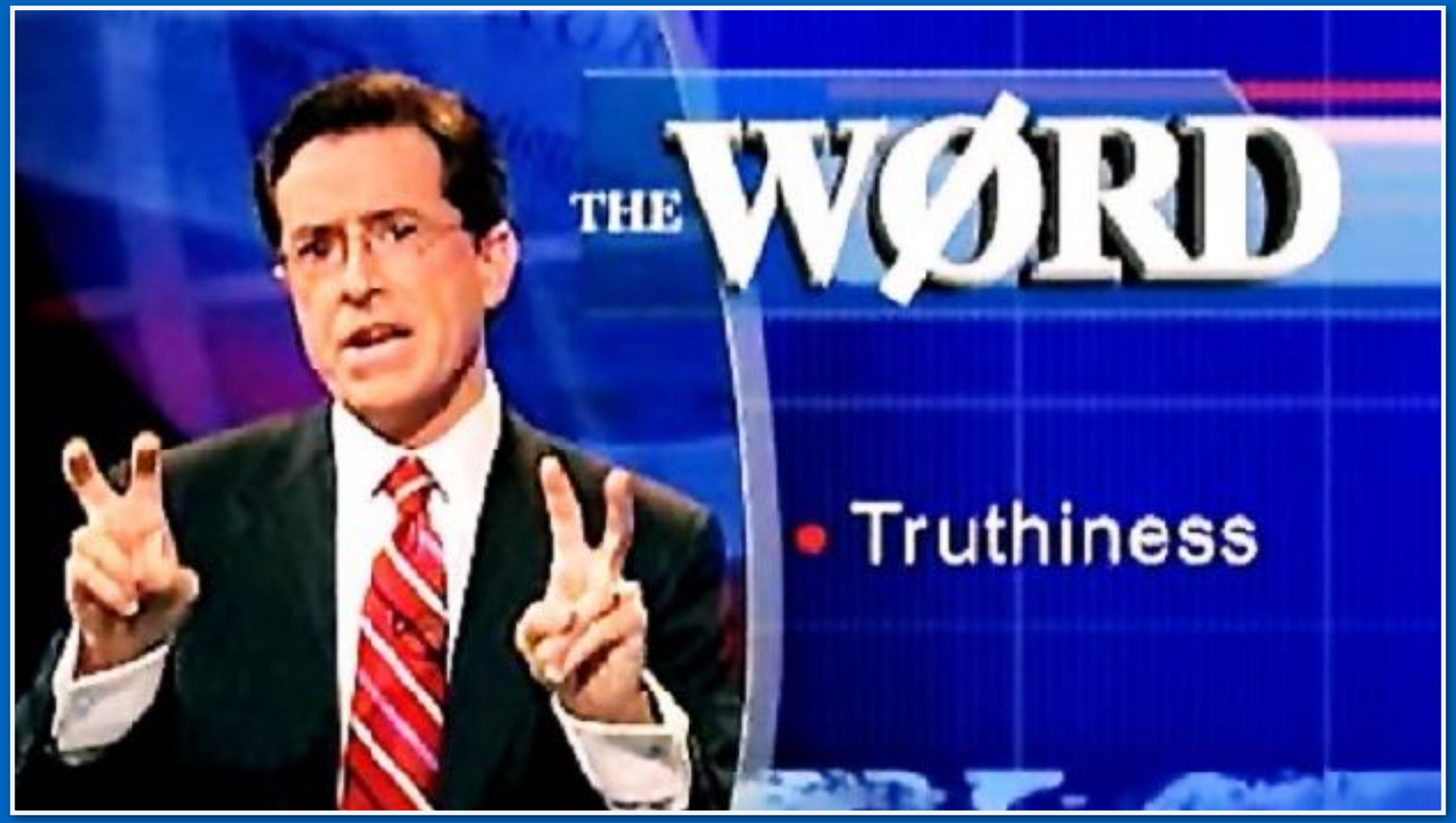
LOGICAL OPERATORS

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
  - instead of `if (name === false)`, just use `if (name)`

LAB — CONDITIONALS

EXERCISE

TYPE OF EXERCISE

- Pair

LOCATION

- `starter-code > 3-ages-lab`

TIMING

until 9:15

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.

LEARNING OBJECTIVES - REVIEW

- Build iterative loops using `for` and `forEach` statements.
- Iterate over and manipulate values in an array.
- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow based on Boolean tests.
- Differentiate among true, false, truthy, and falsy.

NEXT CLASS PREVIEW

Functions and Scope

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

Exit Tickets!

Q&A