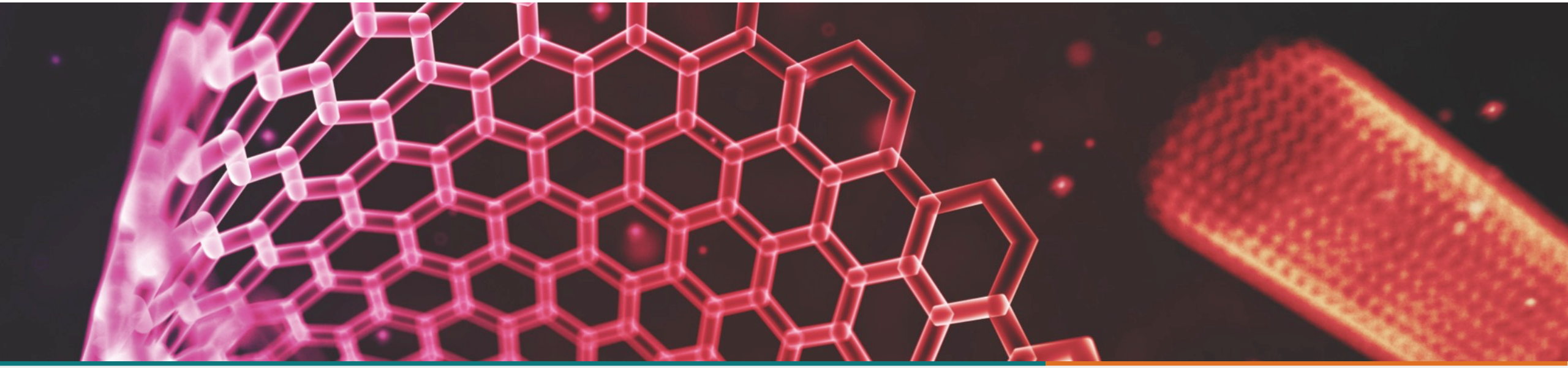
STEVENS
INSTITUTE of TECHNOLOGY
Schaefer School of Engineering & Science

CS 546 – Web Programming I

Collaborative Programming and Intro to HTML



STEVENS

INSTITUTE of TECHNOLOGY

Schaefer School of Engineering & Science

stevens.edu

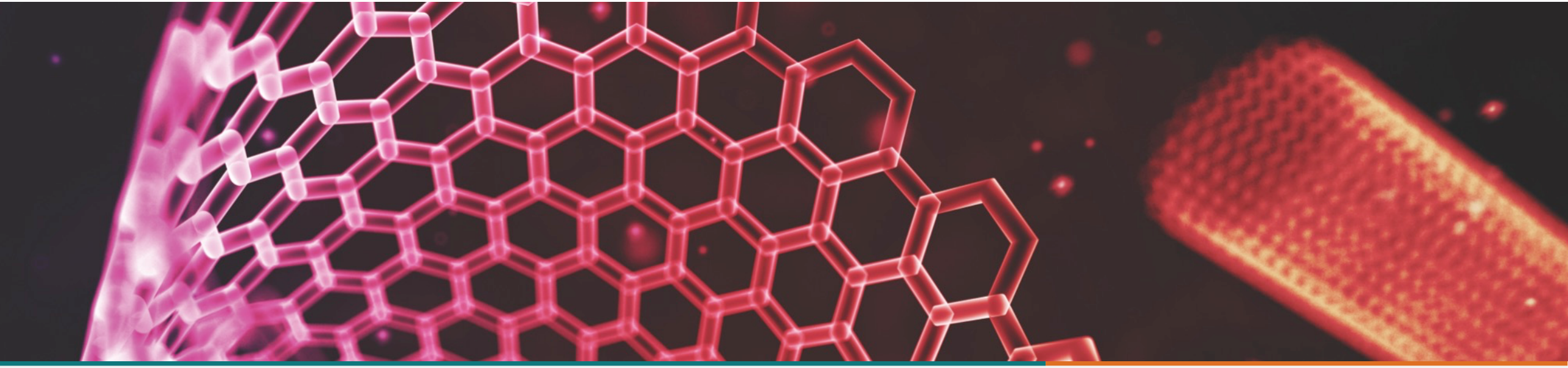
Patrick Hill

Adjunct Professor

Computer Science Department

Patrick.Hill@stevens.edu

Collaborative Programming



How Do We Code Collaboratively?

Learning to code in collaboration with others is a very important skill to learn for your life as a developer.

We can code collaboratively by using a combination of version control software, ticketing systems, and workflows to distribute work amongst our team members

What Is Version Control?

Version control is a system that records sets of changes over time. It allows you to keep track of a history of your files in a very detailed manner.

Version control allows you to see a detailed log of all your changes, as well as roll changes back. It allows you a great deal of control over the state of your files; you can make many, many versions of your code and change between the versions with ease!

- This allows for easy feature development.

What Is Git?

Git is an extremely popular version control software that is commonly used both personally and professionally. It is the most widely used version control software at the current point in time.

It was originally created by Linus Torvalds, who happens to also be the creator of the Linux Kernel.

Git is a distributed version control system: everyone will make a copy of a codebase to work off of on their local machine, and changes on one machine will not affect changes on another machine.

How Can We Use Git?

Git is a command line program, so we will use it from our terminal (much like node)

- https://git-scm.com/

Most of our commands are simple, and we will use Git to work collaboratively.

We will, in general, use Git by setting up a centralized repository that we will publish code to and we will synchronize changes in our local repositories.

For most of our work, we will be using GitHub to store an online copy of our repository.

All groups are required to use a GitHub or similar online repo for the final project. This is so I can keep track of how much work is being done and by whom.

Distributing Work

The easiest way to work in a team is to distribute work.

Very often, work is distributed in one of the following forms:

1. Feature based; each team member owns a portion of the features and does all the work for that area. This includes server routes, data code, HTML, CSS, and frontend JavaScript
2. Architectural ownership; each team member takes a region of the code to own. One user will work on authentication, one will work on routes, one will work on CSS, one on JS, etc.
3. Ticket based; a series of tickets are created regarding each issue, bug, feature, etc., and developers claim tickets to work on.

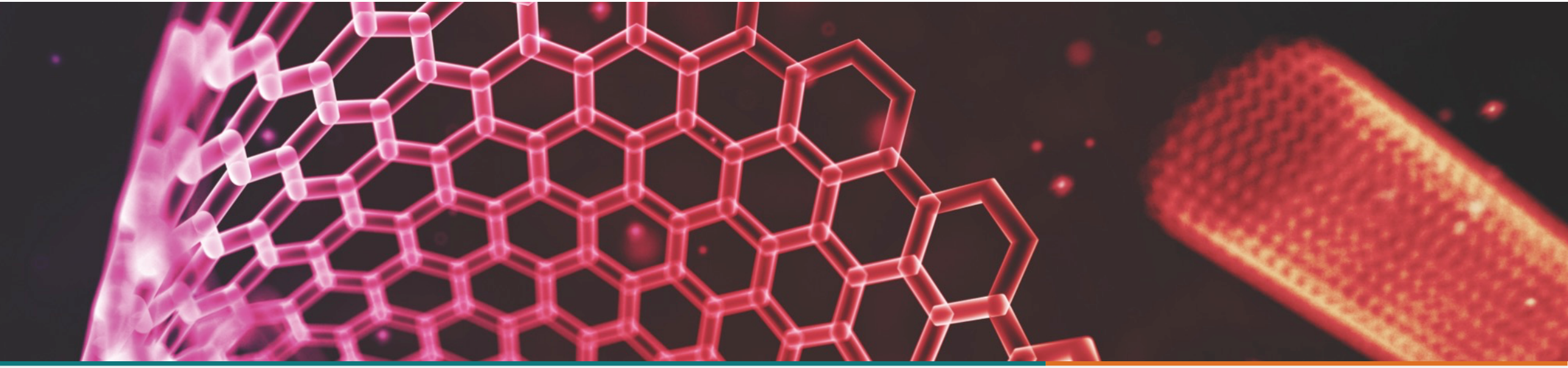
Using a Ticket System

Ticketing systems are a popular way to keep track of issues and features. Project managers often use this software to manage the team workload. You may find this useful for your projects.

Some popular ticket systems/project management apps:

- Asana
  - http://asana.com/
- Trello
  - http://trello.com/
- GitHub issues
  - https://developer.github.com/v3/issues/

Git



Git Terms

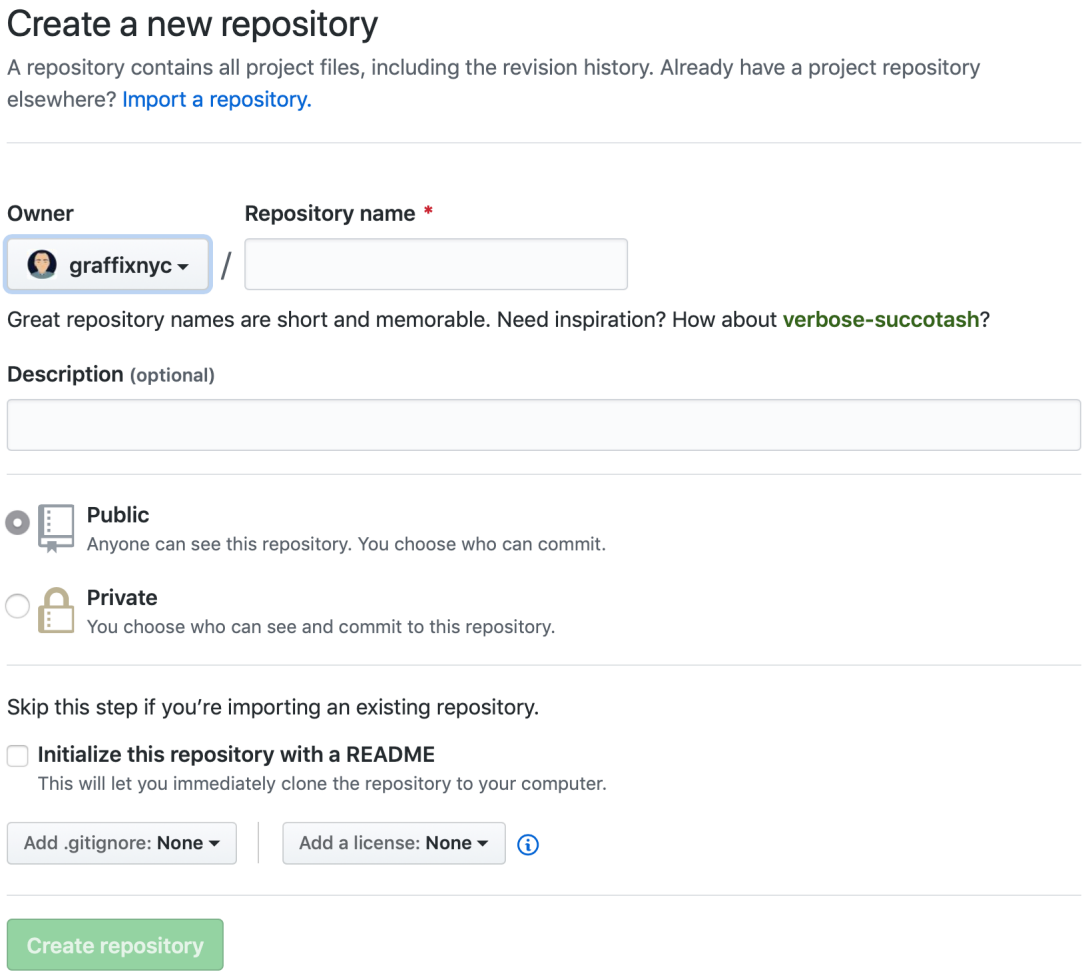
Term	Meaning
Repository	A repository is a location that stores the information about the project's file and folder structure, as well as its history.
Branch	A branch is a pointer to a certain chain of file change histories; you can have many branches but will always have at least one. Traditionally, the original branch is called master.
Commit	A commit is a snapshot of your repository at a point in time.
Remote	A reference to a repository stored outside of your current local machine; i.e., the repository on GitHub.
Push	Pushing is the act of taking your commits and uploading them to a remote repository.
Pull	Pulling is the act of taking commits from a remote repository and bringing the changes down to your local repository.
Merging	Merging is the act of bringing one set of changes from one branch to another and creating a new version of the code with both histories.
Pull Request	A pull request is a request to bring a series of changes from one branch to another

Creating a Repository on GitHub

The first step to using Git is making a repository. A repository is a data structure that stores information about the files and folders of a project, as well as the history of the structure and changes.

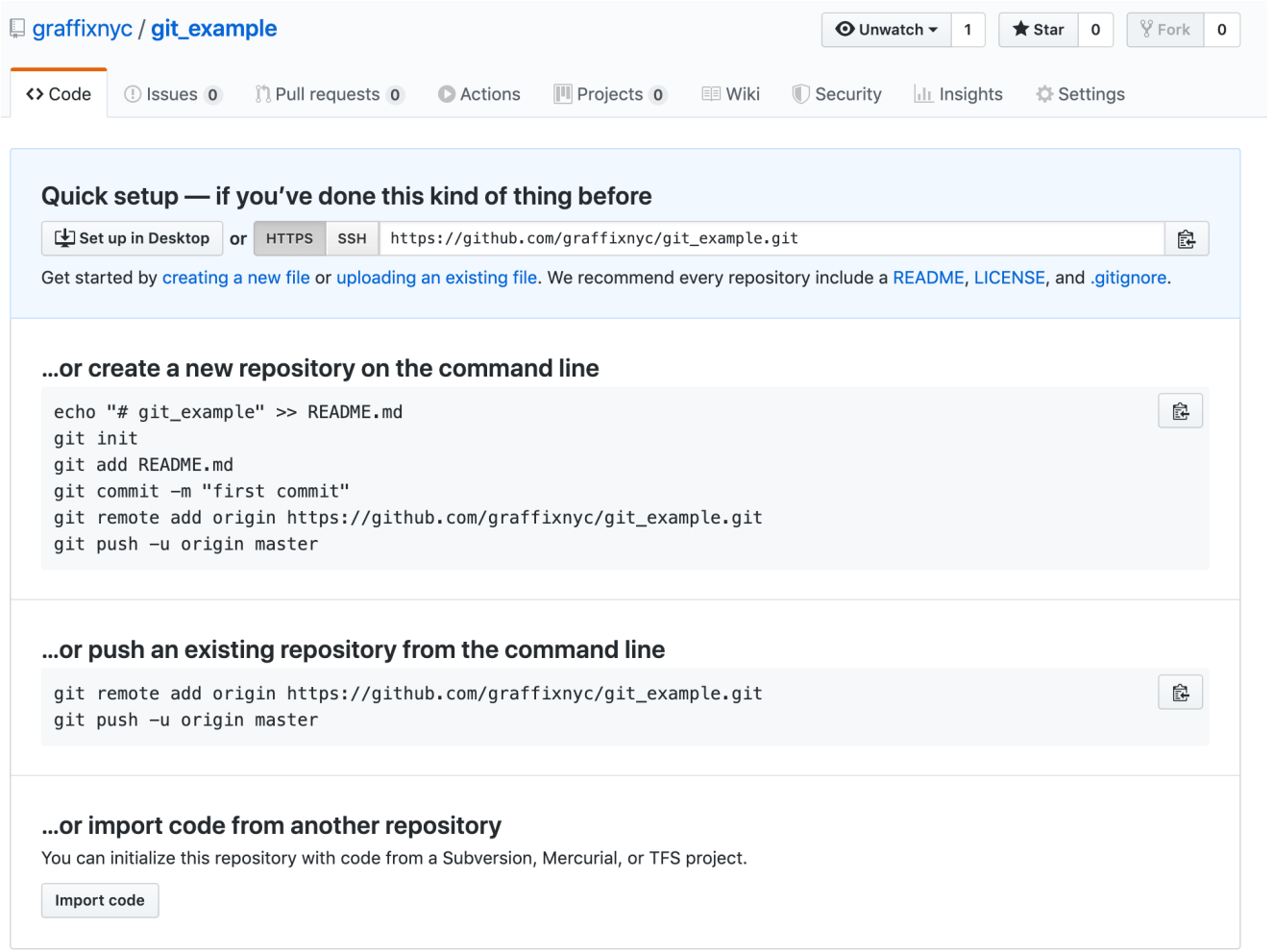
The easiest way to make a repository is to start on GitHub, with the following settings:

- https://github.com/new



Creating a Repository on GitHub

Once you give the repo a name and click “Create repository” the following page with instructions is displayed showing the URL to the remote repo and other information



Setting Up Your Local Repository and Pushing Changes

Once we have created our online repository, we need to create our local copy.

1. Navigate to the folder where you will store your project in the terminal
2. Type in the command ***`git init`***
3. Create a README.MD file using the command: ***`touch README.MD`***
4. Now, we add the file to the staging area using the command: ***`git add README.MD`***
   - If we want to add multiple files and directories at once we can use the command `git add .`
5. Then we commit the file using the following command: ***`git commit -m "initial commit"`***
6. Now we link the remote repository to your local repo: ***`git remote add origin https://github.com/username/reponame.git`***
7. Now, we push our local file to the remote repository: ***`git push -u origin master`***

Pulling Changes

We can pull down changes in the same manner, by issuing a ***`git pull REPOSITORY_NAME BRANCH_NAME`*** command. If there are changes, you will automatically pull and merge these changes into the branch you are currently in.

For example: ***`git pull origin master`***

An Easy Workflow

Let us pretend for a moment that we want to make a series of updates to our readme file. A common workflow is:

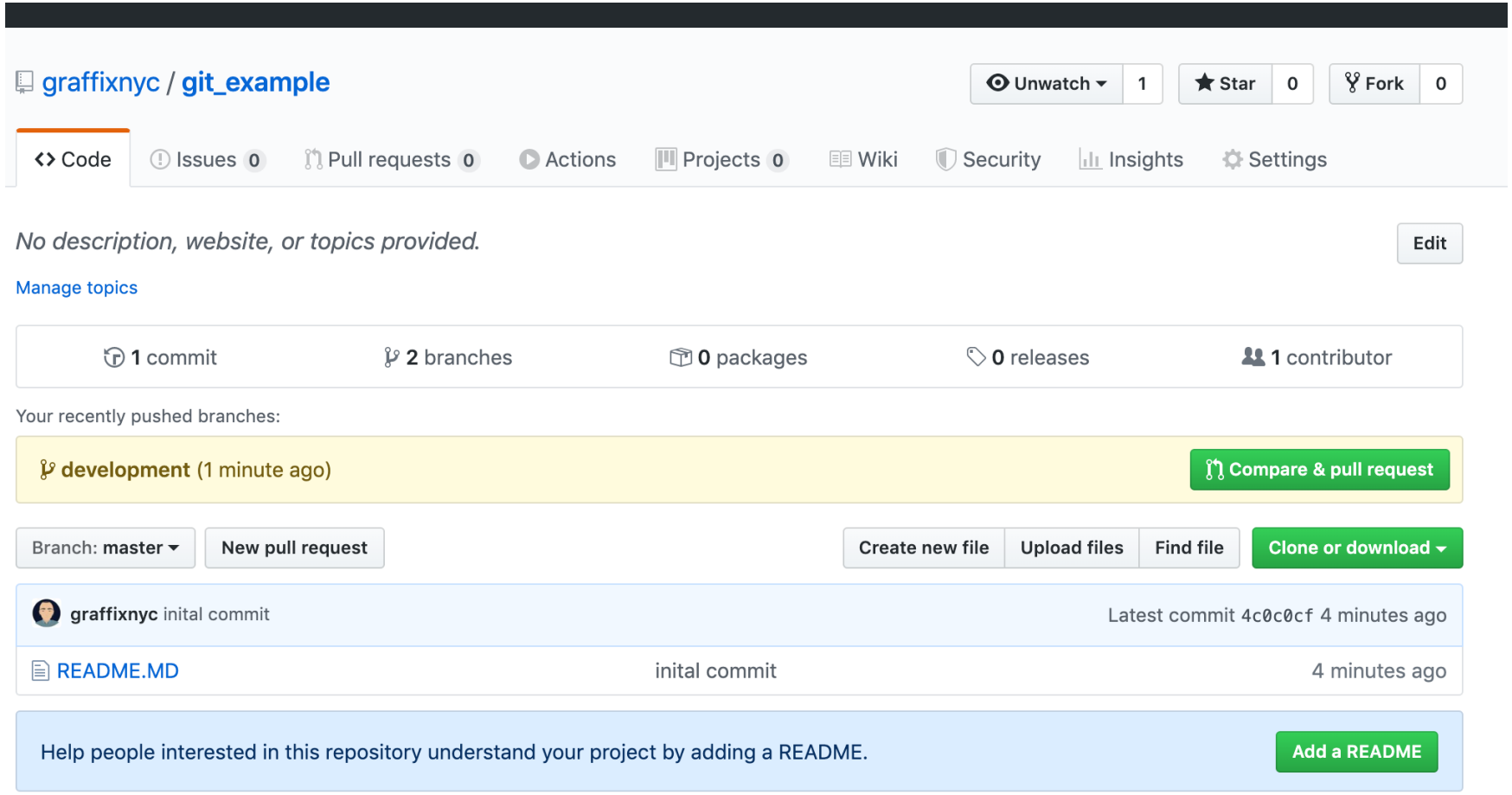
- Make a new branch devoted to all changes to the readme, such as:
  `git checkout -b development`

At this point, we have created a new branch using the `git checkout -b development` command; this creates a new branch named *development*; if we already had that branch and were just moving to the branch, we would omit the *–b* flag.

- Make relevant updates to the codebase
- Make as many commits as needed until satisfied with the result and add them using the `git add` command
- As you commit, push your changes up to a remote branch; the first time you push a new branch, it will be created online!
- When done with the feature, issue a pull request for the code to be reviewed.
- When the code is reviewed and accepted, merge it into the main branch.

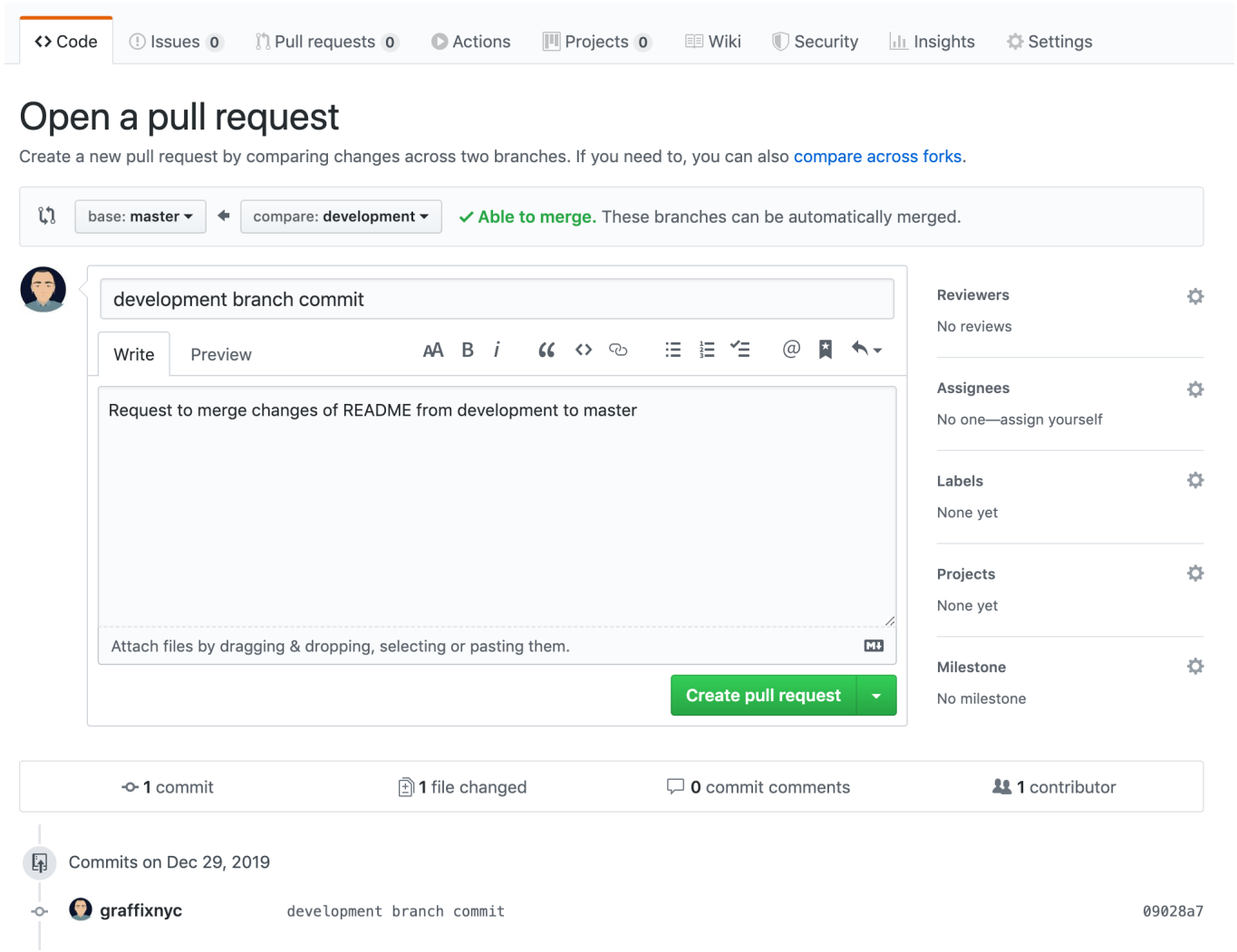
Making a Pull Request

On GitHub, we can create a pull request based on the branch. This will issue a request to have this changes merged into a different branch (in our case, *master*).



Making a Pull Request

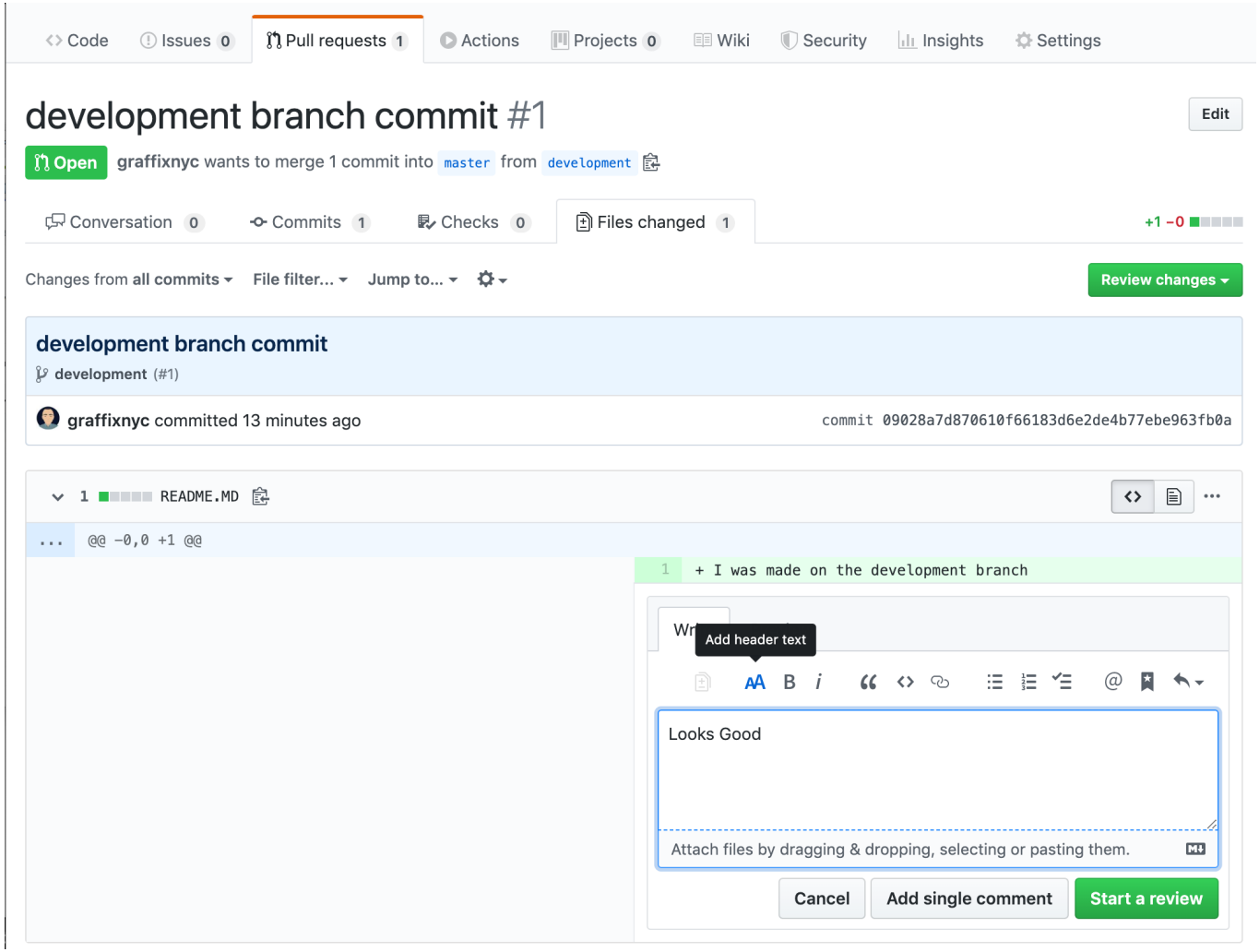
Pull requests should have a description of the set of changes, and when created will show the changes in that branch.



Reviewing a Pull Request

All pull requests should be reviewed by a different developer (and it's also good to have it reviewed by more than one developer); by clicking the *files changed* tab on the pull request page, we can leave comments to the developer about their pull request.

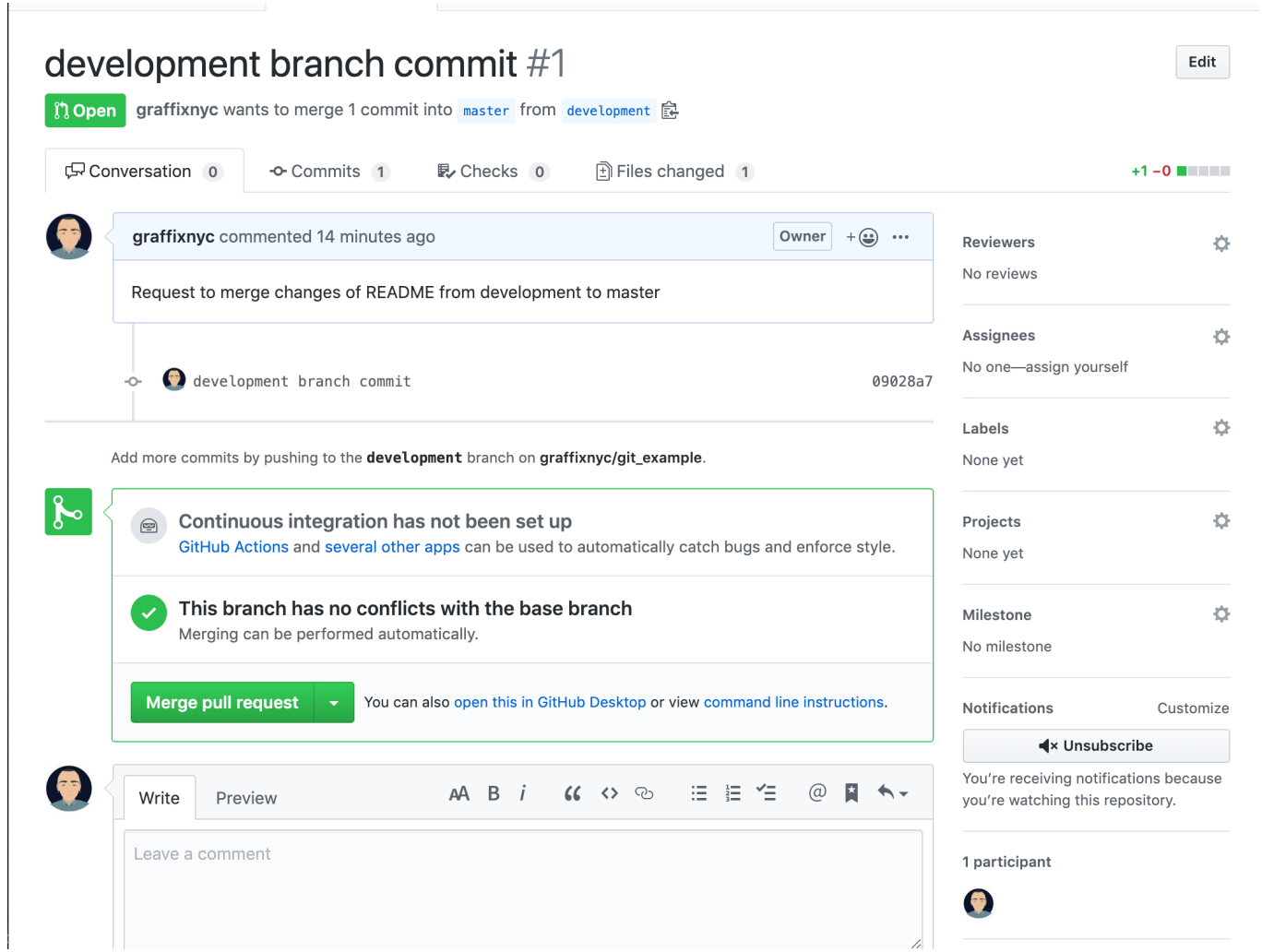
This is very useful for identifying inefficiencies, logical errors, bugs, etc. It also forces multiple developers to look at the approach to solving a problem or making a feature, so that more than one developer is familiar with each portion of the code in a project.



Merging the Pull Request

If a pull request can be safely merged in without conflicts occurring, GitHub will allow you to merge in the changes when all developers are satisfied with the changes.

This will add the new commit data to the master branch of the online repository.



Pulling Changes

At this point, the only repository with the new commit on the *master* branch is the online version; even the repository that we developed on does not have those changes on the master branch(only your local development branch does).

We need to routinely pull from the *master* branch to stay up to date:

```
git pull origin master
```

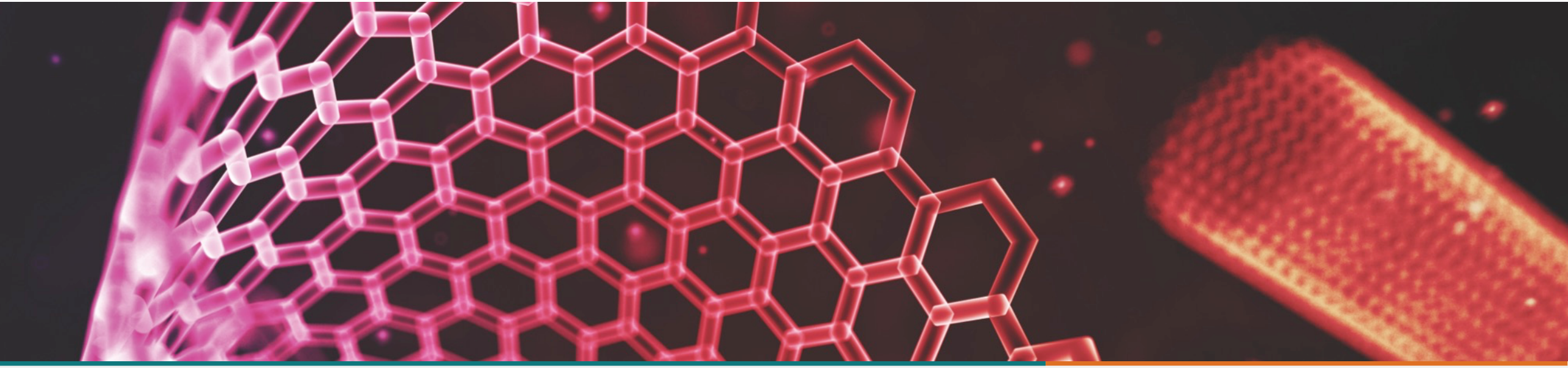
Avoiding Issues

Many issues can occur when using version control, due to the nature of many people editing the same sets of files.

There are a few easy tricks to avoiding most common issues with Git:

- Never develop on the master branch; do all development in your own feature branches, and issue pull requests.
- Pull master into your own feature branches commonly; merge errors will occur that you will need to resolve by hand, but you will ultimately have to resolve these issues far less than if you were all working on the master branch
- Isolate your work into small chunks; do not wait to do a whole feature before you commit. Commit often, as you accomplish small, incremental changes.
- Make new feature branches off of master; master should always be the most up-to-date working code; it is prudent when starting a new feature to get an updated version of the master branch and make a new branch from that up-to-date master branch.
- Pull often; this is so important, that we're listing it twice. Pull often!

Introduction to HTML



Creating an HTML Document

HTML (Hyper Text Markup Language) is a markup language; it is a way of describing content. A file written in HTML is referred to as an HTML document.

Our first HTML documents will exist on our desktops, rather than on a server.

- HTML documents are simply text files that are formed following the HTML standard.
- HTML is composed of a series of tags to describe the content.
- An HTML document is a text document that describes a **web page**.

Your browser will interpret this document and render it!

What's in an HTML Document?

- An HTML has a series of elements
  - Open tag plus attributes and properties
  - Nested elements
- Some very important
  - HTML Doctype
  - HTML Element
  - Head Element
  - Body Element
- Elements can be identified by an ID: ID can only be used once per document
- Elements can identify a group by their class
- Elements are described in the document and rendered in the DOM

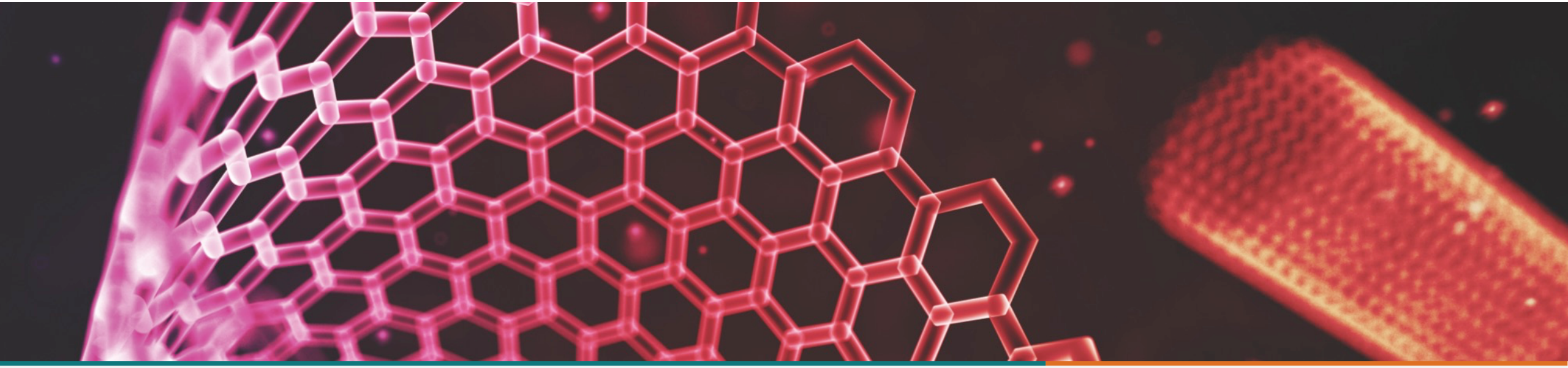
Starting an HTML Document

All HTML documents start with the following barebones structure. Content to be visible on your page will go inside the body tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <!-- Content Goes here -->
</body>
</html>
```

HTML Skeleton File

HTML Elements



What is an HTML Element?

An HTML element usually consists of a **start** tag and an **end** tag, with the content inserted in between:

*<tagname>*Content goes here...*</tagname>*

The HTML **element** is everything from the start tag to the end tag:

* *My first paragraph.* *

Some elements are called **block-level** elements and some elements are called **inline** elements

Block Elements vs Inline Elements

Block-Level Elements

- A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

Block level elements in HTML:

<address>	<article>	<aside>	<blockquote>	<canvas>
<dd>		<dl>	<dt>	<fieldset>
<figcaption>	<figure>	<footer>	<form>	<h1>-<h6>
<header>	<hr>		<main>	<nav>
<noscript>		<p>	<pre>	<section>
<table>	<tfoot>		<video>	

Block Elements vs Inline Elements

Inline Elements

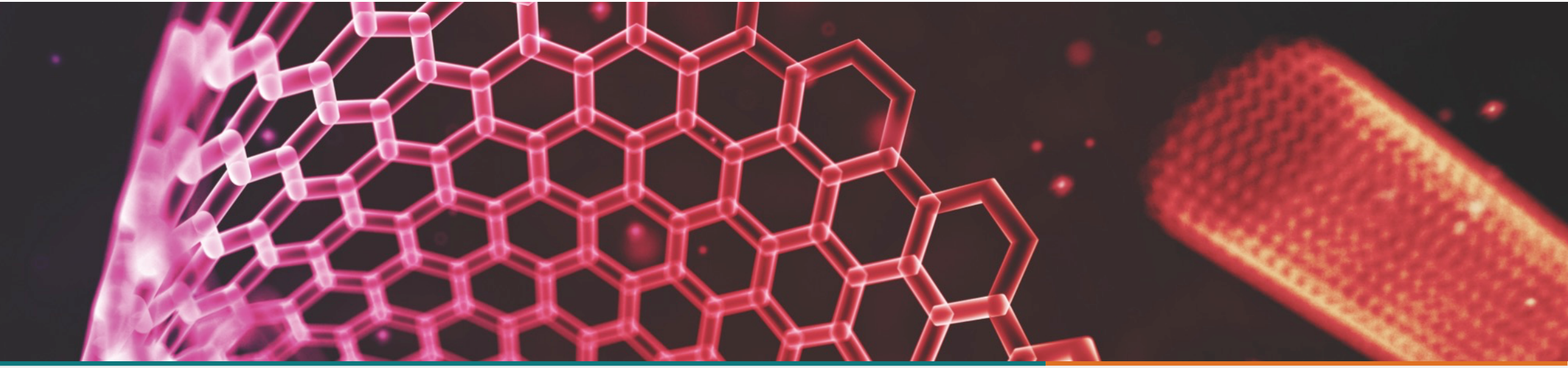
- An inline element does not start on a new line and only takes up as much width as necessary.

  Inline elements in HTML:

<a>	<abbr>	<acronym>	<b>	<bdo>
<big>	 	<button>	<cite>	<code>
<dfn>		<i>	<img>	<input>
<kbd>	<label>	<map>	<object>	<output>
<q>	<samp>	<script>	<select>	<small>
		<sub>	<sup>	<textarea>
<time>	<tt>	<var>		

More about block-level vs inline elements: https://coursework.vschool.io/html-block-vs-inline/

Common HTML Elements



Common HTML Elements: Document Structure

Opening Tag	Closing Tag	Description
<html>	</html>	Opens and closes an HTML document
<head>	</head>	The first of two main sections of an HTML document. The <head> section is used to provide information about the document for use primarily by search engines and browsers.
<title>	</title>	The title of document. This element is nested inside the <head> section. In HTML5, this is the only required tag other than the DOCTYPE declaration.
<body>	</body>	The second of two main sections of an HTML document. The <body> section contains all the content of the web page.

Common HTML Elements: Content (Container)

Opening Tag	Closing Tag	Description
<h1> to <h6>	</h1> to </h6>	Headings. H1 is the main heading, H2 is secondary, etc.
		Paragraph
		A container for a *block* of content
		A container for *in-line* content, such as content inside a paragraph.
		Gives the contained text emphasis (usually as *italics*).
		Makes the contained text **bold**.
		Link
		Ordered (numbered) list
		Unordered (bulleted) list
		List item, must be nested inside a list element such as a or
<!--	-->	Comment. Anything between these tags is not displayed on the screen. This is useful for making notes to yourself or to others who may view the source code of the web page.

Common HTML Elements: HTML 5 Semantic Tags

Opening Tag	Closing Tag	Description
<article>	<article>	Represents a self-contained composition in a document, page, application, or site, which is intended to be independently distributable or reusable (e.g., in syndication). Examples include: a forum post, a magazine or newspaper article, or a blog entry.
<aside>	</aside>	Represents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.
<details>	</details>	creates a disclosure widget in which information is visible only when the widget is toggled into an "open" state. A summary or label can be provided using the <summary> element.
<figcaption>	</figcaption>	Figure Caption element represents a caption or legend describing the rest of the contents of its parent <figure> element.
<figure>	</figure>	Represents self-contained content, potentially with an optional caption, which is specified using the (<figcaption>) element. The figure, its caption, and its contents are referenced as a single unit.
<footer>	</footer>	Represents a footer for its nearest sectioning content or sectioning root element. A footer typically contains information about the author of the section, copyright data or links to related documents.
<header>	</header>	Represents introductory content, typically a group of introductory or navigational aids. It may contain some heading elements but also a logo, a search form, an author name, and other elements.

Common HTML Elements: HTML 5 Semantic Tags

Opening Tag	Closing Tag	Description
<main>	</main>	Represents the dominant content of the <body> of a document. The main content area consists of content that is directly related to or expands upon the central topic of a document, or the central functionality of an application.
<mark>	</mark>	Represents text which is marked or highlighted for reference or notation purposes, due to the marked passage's relevance or importance in the enclosing context.
<nav>	</nav>	Represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are menus, tables of contents, and indexes.
<section>	</section>	Represents a standalone section — which doesn't have a more specific semantic element to represent it — contained within an HTML document. Typically, but not always, sections have a heading.
<summary>	</summary>	Specifies a summary, caption, or legend for a <details> element's disclosure box. Clicking the <summary> element toggles the state of the parent <details> element open and closed.
<time>	</time>	represents a specific period in time. It may include the datetime attribute to translate dates into machine-readable format, allowing for better search engine results or custom features such as reminders.

Common HTML Elements: Empty (Non-Container)

Opening Tag	Description
 	Inserts a line break
<embed>	Embeds external content at the specified point in the document. This content is provided by an external application or other source of interactive content such as a browser plug-in.
<hr>	Inserts a horizontal rule (line)
<img>	Inserts an image into a web page.
<input>	Is used to create interactive controls for web-based forms in order to accept data from the user
<link>	Specifies relationships between the current document and an external resource. This element is most commonly used to link to stylesheets
<meta>	Represents metadata that cannot be represented by other HTML meta-related elements
<source>	Specifies multiple media resources for the <picture>, the <audio> element, or the <video> element.
<track>	Is used as a child of the media elements <audio> and <video>. It lets you specify timed text tracks (or time-based data)
<wbr>	Represents a word break opportunity—a position within text where the browser may optionally break a line, though its line-breaking rules would not otherwise create a break at that location.

Common HTML Elements: Tables

Opening Tag	Closing Tag	Sample Attributes	Description
<table>	</table>		Adds a table
<tr>	</tr>		Table row (start & end).
<th>	</th>		When creating a table to display data, use this tag to differentiate the first row or column of cells as heading cells for all the other cells in the same column or row. Browsers typically display this element bold and centered within the table cell. The ***scope*** attribute defines whether this is a row header or column header.
<td>	</td>	scope="row" scope="col"	Table data cell.
	colspan="number"		Use with <th> or <td> elements. Spans cells across multiple columns.
	rowspan="number"		Use with <th> or <td> elements. Spans cells across multiple columns.

For even more elements available check out: https://developer.mozilla.org/en-US/docs/Web/HTML/Element

Common HTML Elements: Form Elements

Opening Tag	Closing Tag	Description
<form>	</form>	Defines a form that is used to collect user input
<input>	none	Allows the user to input data. It can be displayed in several ways, depending on the type attribute (more about that next slide).
<select>	</select>	Defines a drop-down list.
<textarea>	</textarea>	Defines a multiline input field
<button>	</button>	Defines a clickable button.

Common HTML Elements: Form Input Types

We can give inputs different types to handle different types of input: `<input type="VALUE">`

Type	Description
type="text"	Defines a one-line text input field
type="password"	Defines a password field
type="submit"	Defines a button for submitting form data to a form-handler
type="reset"	Defines a reset button that will reset all form values to their default values
type="radio"	Defines a radio button
type="checkbox"	Defines a checkbox
type="button"	Defines a button
type="color"	Is used for input fields that should contain a color.
type="date"	Is used for input fields that should contain a date.
type="datetime-local"	Specifies a date and time input field, with no time zone.
type="email"	Is used for input fields that should contain an e-mail address.

Common HTML Elements: Form Input Types

We can give inputs different types to handle different types of input: `<input type="VALUE">`

Type	Description
type="file"	Defines a file-select field and a "Browse" button for file uploads.
type="month"	Allows the user to select a month and year.
type="number"	Defines a numeric input field
type="range"	Defines a control for entering a number whose exact value is not important (like a slider control)
type="search"	Is used for search fields (a search field behaves like a regular text field).
type="tel"	Is used for input fields that should contain a telephone number.
type="time"	Allows the user to select a time (no time zone)
type="url"	Is used for input fields that should contain a URL address.
type="week"	Allows the user to select a week and year.

Improving Our HTML Documents

This week we're going to create a meaningful document about different types of coffee and explain how to make it marked up in a sensible way.

We will learn about:

- Formatting text
- Organizing our data
- Lists
- Tabular data

Reusability and Repetition

For web programming, it's very necessary to think of everything based in terms of reusable components. There's a lot of repetition in web programming, where you're displaying many different instances of similar data

- Every tweet has all the same info
- Every blog post has a title, time, body, etc.
- Product descriptions all have prices, titles, etc.

Because of this, we're going to look at our programming in terms of:

- Is this reusable? If so, how?
- How can I make this component accessible?
    - More on this when we get to forms in a few weeks.

The Browser is Only Half the Battle

The web isn't just accessible via a browser. As modern web developers, we must care about:

- Screen readers
- Search Engine Crawlers / Other AI

There is a growing movement to make the web more accessible

- Leveraging HTML's strengths
  - Navs in the nav
  - Labels in forms
  - Using headings properly
  - Attributes to help screen readers
- Making designs accessible
- Tables for tabular data only
- Make sure you can navigate via keyboard

Separating Style and Content

Before we can think in terms of organizing our data meaningfully, we need to understand what HTML does not accomplish; the way your document looks.

- **Elements are used to describe your data; CSS is used to style your data.**
- While browsers give many native styles to elements by default, elements are not inherently used for styling. This is why tags for bolding and italicizing text, or changing fonts, were deprecated in HTML5.
- There needs to be a clear separation between style and content; any overlap is a happy coincidence.

While writing HTML, thinking in terms of content first, then styling often leads to more logical, and easier to style documents.

**You will get points off your labs if you use tags like <b>, <i>, **

Types of Text

Across the web, text is used to portray many different types of things.

- Headings / Titles in your content (**h1**, **h2**, **h3**, **h4**, **h5**, **h6**)
- Regular paragraph (**p**)
- Generic groups of text / adding custom definitions or functionality to text (**span**)
- Emphasized text (**em**)
- Important text (**strong**)
- Addresses (**addr**)
- Citations (**cite**)
- Abbreviations (**abbr**)
- Quotes (**blockquote**)

Using the right kind of element to describe text is very important for SEO, non-browser accessibility, and readable code.

- Even without different styles, those tags help readers understand their document.

The Layout of Your Content

There are many elements that describe the layout of your content

- How to navigate content / your document (**nav**)
- Grouping your content into sections that have something to do with each other (**main**, **section**)
- Denoting a header for content or your document (**header**)
- Denoting a footer for content or your document (**footer**)
- Grouping content into a self-contained article (**article**)
- Stating that certain content is secondary (**aside**)
- Grouping divisions of content (**div**)

List Data

Lots of data you'll see and create is some form of a list

- Unordered lists state that the order of the items in the list don't matter (**ul**)
- Ordered lists state that the order of the items in the list have some sort of meaning (**ol**)
- Each entry in a list is a list item(**li**)

You'll very often find see nested lists:

```
<ul>
    <li>Item 1</li>
    <li>Item 2
        <ul>
            <li>Subitem</li>
        </ul>
    </li>
</ul>
```

Tabular Data

Data is also often presented in a table format. Each table has:

- A table element (**table**)
  - A table header (**thead**) (optional)
    - A table row (**tr**)
      - Multiple table header cells (**th**)
  - A table footer (**tfoot**) (optional)
    - A table row (**tr**)
    - Multiple data table cells (**td**)
  - A table body
    - Multiple table rows (**tr**)
    - Multiple table data cells (**td**)

Meaningfully Grouping a News Article

A news article is easily represented properly in HTML.

- Article
  - Header
  - By line (Sub header)
  - Body paragraphs
  - Footer with comment form

Meaningfully Grouping a Recipe

Just because the tag is ‘article’ doesn’t mean it just has to be news! The article is “an article of content”.

- Article
  - Header: title of recipe, possibly details like cooking skill required
  - A list of ingredients
  - Body paragraphs explaining how to cook recipe
  - An aside with nutritional information

Referencing Assets

Relative: When you specify a path as a relative location, the browser attempts to find these assets relative to your current location.

- When you are at http://localhost/blogs/ and use a relative path of ***my_image.jpg*** and ***styles/background.png***, your browser will attempt to find the resources at **http://localhost/blogs/my_image.jpg** and **http://localhost/blogs/styles/background.png** respectively.

Root Relative: Similar to relative, you can have root relative paths; these will be relative locations based on the root of your host (it will not take the current path into account)

- When you are at http://localhost/blogs/ and use a root relative path of ***/images/my_image.jpeg*** your browser will attempt to find the resources at **http://localhost/images/my_image.jpg**

Absolute: You can reference elements by an entire URL (protocol, host, path, etc.) and your browser will look for these directly

- When you are at http://localhost/blogs/ and use an absolute path of **http://localhost/images/my_image.jpeg** your browser will use that URL to locate the image

Referencing External Assets

Images

- You reference images as normal HTML elements. The image tag is the img

Stylesheets

- In the head of your document, you can specify CSS stylesheets to apply to your document using the link tag; these will be loaded in order of tag appearance

Scripts

- You reference scripts using the script tag.
- Your script files should almost always be placed right before your closing body tag; this will allow your browser to render the page and then add function to it, rather than locking the page up to perform JavaScript tasks while the page is still loading. Scripts will be referenced in the order you include them.

Validating HTML

For this course, the validity of your HTML is highly important.

Having valid HTML means your browser does not have to guess how to fix it, which can lead to drastically wrong web pages and pages that cannot be made sense of.

The w3 website has an easy to use validation service that tells you issues and proposed solutions:

- https://validator.w3.org/#validate_by_input

You should view the source of your page, copy, and paste it all into the HTML validator's 'direct input' section before submitting HTML in this class.

You should strive to write as perfect HTML as possible.

You will get points off your labs if you have ANY invalid HTML

Attributes and Properties

Elements can have many classes and properties attached to them to further describe them. The difference between the two of those are nuanced and deals with the state of the page.

- This is an example of how browsers had to adapt to a set of standards that were not always fully thought out.

Attributes appear in key-value fashion when writing HTML:

- `<a href="http://google.com">Go To Google</a>`

The ***href*** attribute is set to Google's home page

Elements are parsed and, as they are changed, keep track of the set of properties. Some properties come from their attributes; others come from user input.

Classes and IDs

Elements will often be described with classes and IDs to signify them in some way. Many elements can share a class, and each element can have many classes

```
<div class="panel panel-default"></div>
```

Has two classes, panel and panel-default

```
<div class="panel panel-danger"></div>
```

Has two classes, panel and panel-danger

However, only one element can have a particular ID:

```
<div id="about-me"></div>
```

Classes and IDs are most often used to style elements and target elements with JavaScript to add functionality.

Questions?