GitHub

Version control software solutions include Mercurial, Perforce, Subversion, and CVS. We'll be using GitHub for this course, which is based on Git.

You should already have a conceptual grasp of version control concepts, as introduced during the lecture session.

Setting Up GitHub

Firstly, set up a GitHub account:

1. go to https://github.com;
2. click "Sign up" to create a new account;
3. follow the required steps.

Be aware that we are making use of free GitHub accounts. All code is publicly viewable. **Do not** add sensitive information to your project files (i.e. passwords in your source code) and only add code and assets that you are comfortable publishing on the world wide web.

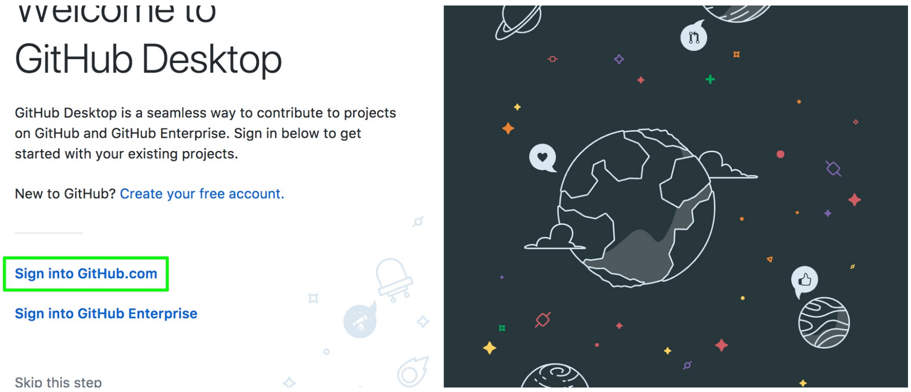
Setting Up GitHub Desktop

GitHub Desktop provides a graphic interface for Git-based projects -- which means you can avoid typing-in commands via the terminal. It connects to the github.com server, enabling one to work with a web-hosted repository. In other words: the *master* repository resides on the server, and you clone it to your computer in order to work on it. Any changes you make can then be synchronised back up to the server.

You'll need to download the GitHub Desktop software. It's free and is available for both Mac and Windows.

1. Go to https://desktop.github.com;
2. grab the download for your platform (Mac/Windows);
3. run the GitHub installer.

Upon opening the application for the first time, the software presents a welcome screen; click **Sign into GitHub.com**:

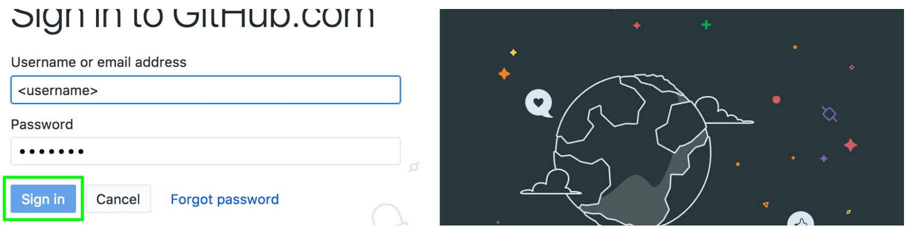


Welcome to GitHub Desktop

GitHub Desktop is a seamless way to contribute to projects on GitHub and GitHub Enterprise. Sign in below to get started with your existing projects.

New to GitHub? Create your free account.

Next, enter your github.com account details; then click **Sign in**:

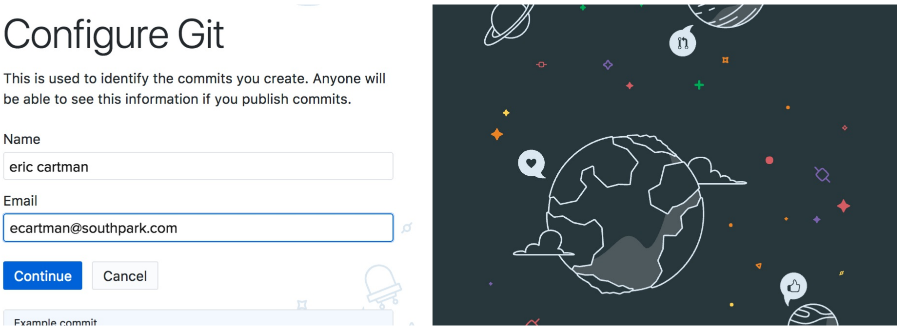


Sign in to GitHub.com

Username or email address: <username>

Password: •••••••

Your name and an associated email address are attached to every commit you make to a Git repository (more on this later). Add these details in the 'configure' step:



Configure Git

This is used to identify the commits you create. Anyone will be able to see this information if you publish commits.

Name: eric cartman

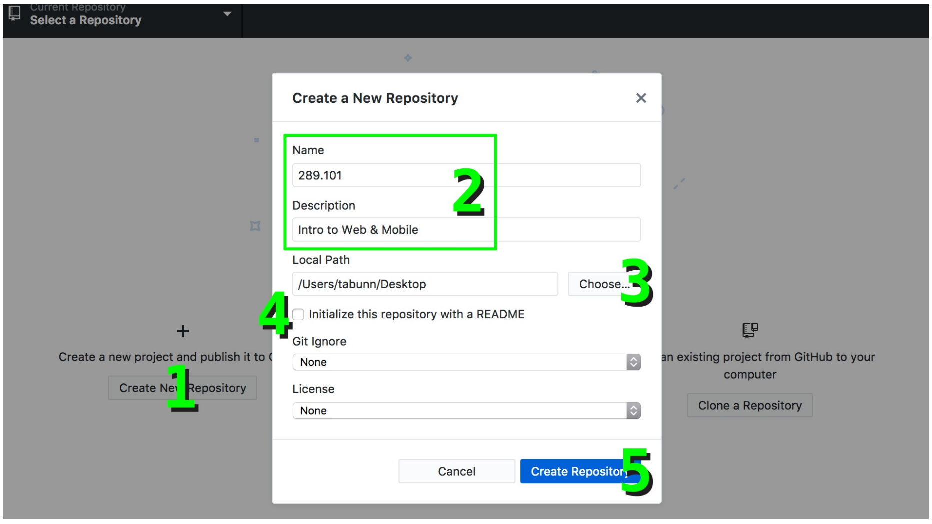
Email: ecartman@southpark.com

There will be one more step requesting anonymised usage data. I'll leave that decision to you.

Creating a New Repository

Now that GitHub Desktop has been configured, you can create a new repository.

1. Click **Create New Repository**;
2. *name* it `289.101`, and enter `Intro to Web & Mobile` in *Description* field;
3. set the path to wherever it is you prefer to work (the Desktop is probably a sensible choice);
4. there is no need to initialise a README, gitignore, or license file (you can always do that later if you wish);
5. now click **Create Repository**.



Name: 289.101

Description: Intro to Web & Mobile

Local Path: /Users/tabunn/Desktop

□ Initialize this repository with a README

Git Ignore: None

License: None

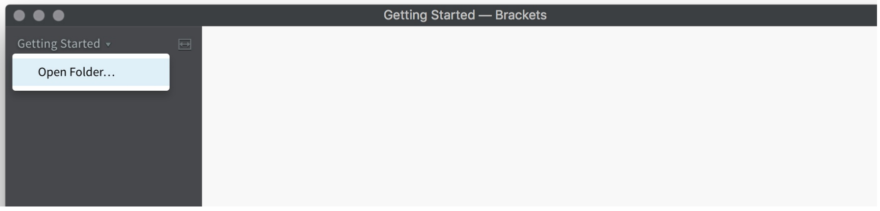
This step will create a new directory/folder titled `289.101` on your desktop.

Note that the terms *directory* and *folder* are used interchangeably -- but real coders use *directory* ;P

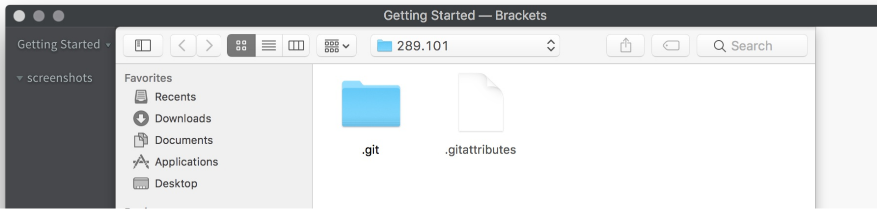
Adding Files to Your Repository

To add files to the repo, you will need to save them in the `289.101` directory. For the sake of convenience, it's best to point your editor to your working project.

Open the Brackets editor; then, from the *Getting Started* drop-down, select **Open Folder...**



Locate and select the `289.101` directory (on your desktop?). Note: you will see files/folders that the OS Finder/File Explorer may ordinarily hide from you. **Do not delete these** -- or anything that begins with `.git`



Brackets will now list the *289.101* directory and all of its contents in the left panel. However, you won't see the `.git` directory. Brackets is smart enough to know that you should *never* edit anything in there!

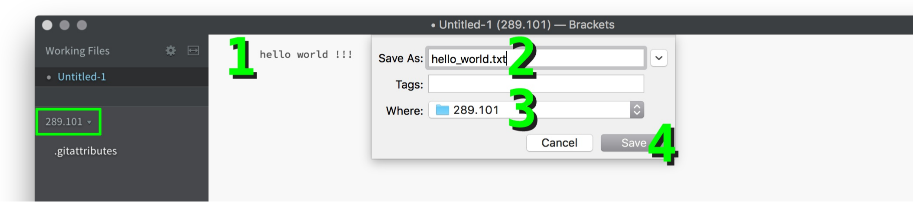
Create a new file in Brackets (**File > New**), then:

1. add a `hello world !!!` line;

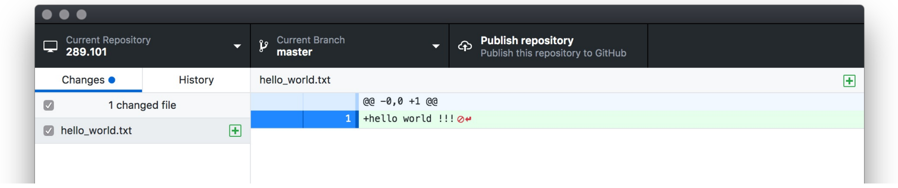
2. save it as *hello_world.txt* (.txt is the file extension for plain-text files);

3. ensure it is being saved in the correct location (this should default to *289.101*);

4. then click **Save**.



Switch back to the GitHub Desktop application. You should see the new file listed under the Changes panel:



```
@@ -0,0 +1 @@
1 +hello world !!!
```

Committing Changes

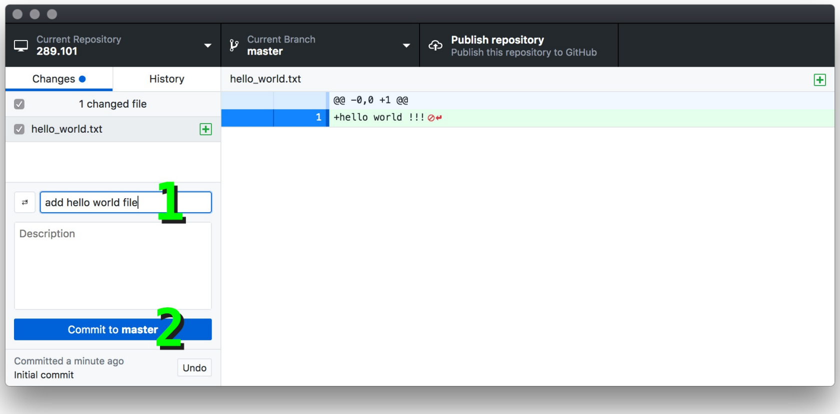
Although you have made changes to your project, you've yet to commit them. You generally don't commit after every line of code you edit, but rather after each successfully completed 'task'. Your lecturer/tutor will provide some advice on how best to manage your commits (based on tasks, cards, WIP commits, etc.).

To commit your latest changes:

1. enter a commit summary -- in this case: `add hello_world file`;

2. and then click **Commit to master**.

We won't bother with commit *Description*s. By convention, the summary is not supposed to exceed 50 characters. The description, however, provides a more detailed explanation.

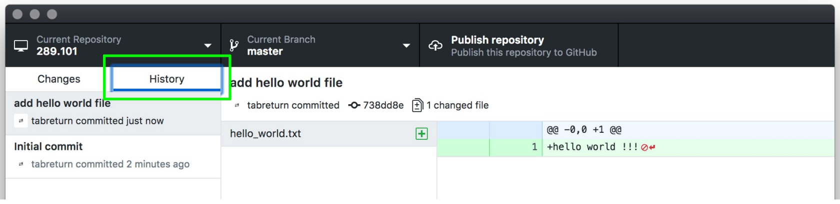


```
@@ -0,0 +1 @@
1 +hello world !!!
```

Summary: add hello world file

Committed a minute ago
Initial commit

If you wish to view a list of your commits and their accompanying changes, use the History tab:



add hello world file
tabreturn committed just now

Initial commit
tabreturn committed 2 minutes ago

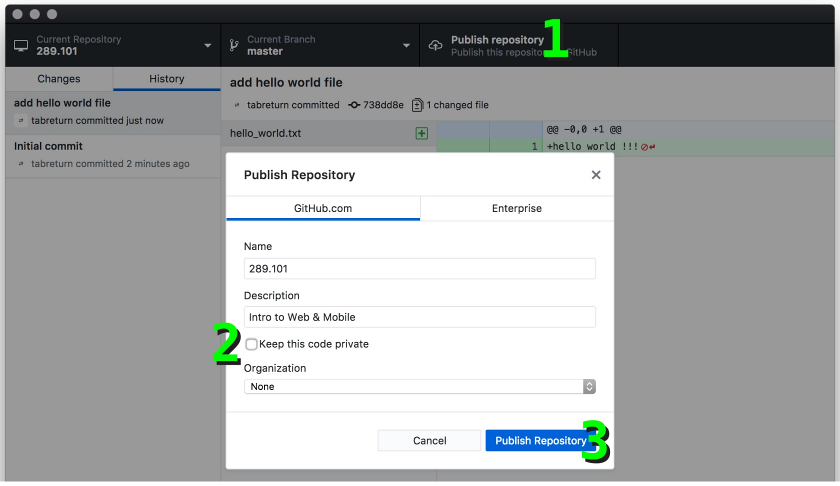
add hello world file
tabreturn committed 738dd8e 1 changed file

```
@@ -0,0 +1 @@
1 +hello world !!!
```

Pushing/Publishing Changes

You can make as many commits as you like (more on that soon), but until you *push* the commits to your server, these changes exist on your computer only. However, you have yet even to *publish* your working repo -- meaning that your first push will also create the repo on GitHub's server as well as push any commits.

1. Click **Publish repository**;
2. ensure the *Name* and *Description* are correct and uncheck the *Keep this code private* option (you'll need to be a paying GitHub user for this feature);
3. then click **Publish Repository**.



add hello world file
tabreturn committed just now

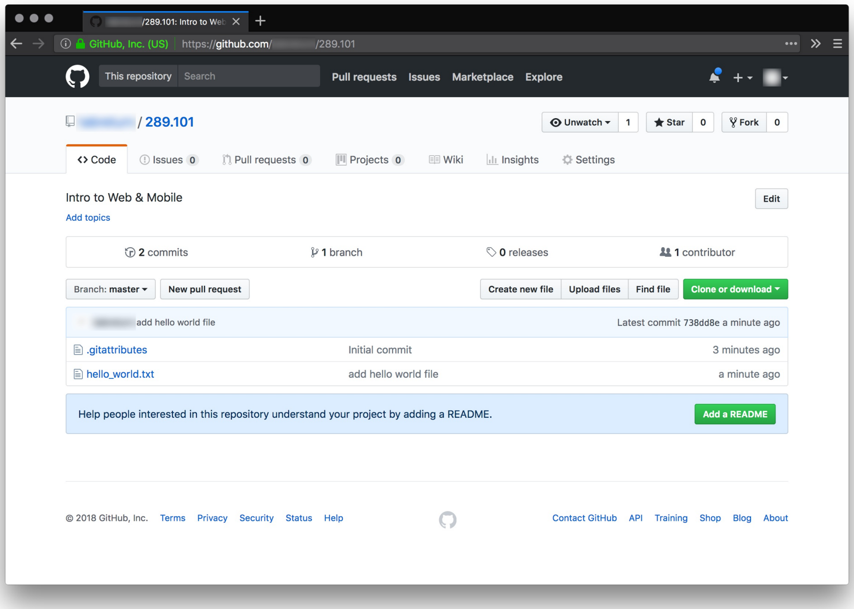
Initial commit
tabreturn committed 2 minutes ago

add hello world file
tabreturn committed 738dd8e 1 changed file

```
@@ -0,0 +1 @@
1 +hello world !!!
```

Name: 289.101
Description: Intro to Web & Mobile
□ Keep this code private
Organization: None

You can verify that your commits have reached the server by accessing the newly published repository on the github.com website:



[redacted]/289.101: Intro to Web

GitHub, Inc. (US) https://github.com/[redacted]/289.101

[redacted] / 289.101

Intro to Web & Mobile

2 commits · 1 branch · 0 releases · 1 contributor

[redacted] add hello world file — Latest commit 738dd8e a minute ago

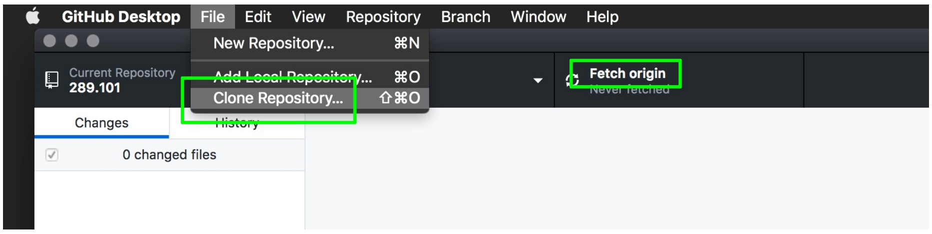
.gitattributes	Initial commit	3 minutes ago
hello_world.txt	add hello world file	a minute ago

Help people interested in this repository understand your project by adding a README.

© 2018 GitHub, Inc.

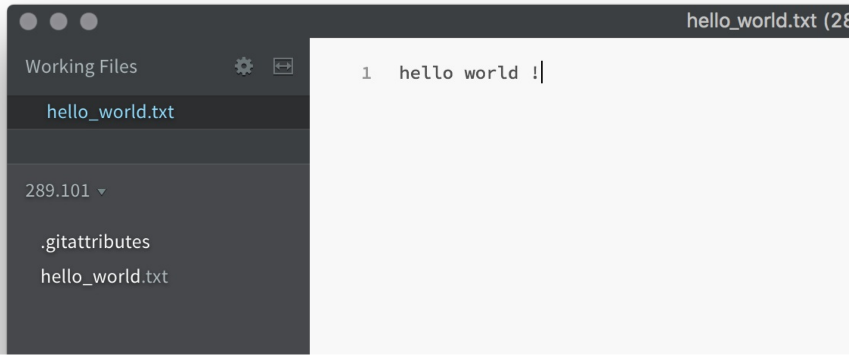
If you're working at college (or wherever else), always ensure that you commit all of your changes and **push them before you leave**. On that point: it's not necessary to push after every commit -- instead, you can pile up a whole bunch of commits and then do a single push when you have finished working. This way when you pull from another device (say, your computer at home), you will receive the latest version of what you last uploaded.

Use the **Clone Repository...** option to get an existing repo up and running on a computer. Use **Fetch origin** to pull the latest changes to your device.



More Commits

To get a better grasp of this whole commit thing, edit to your txt file using Brackets; remove two exclamation marks:

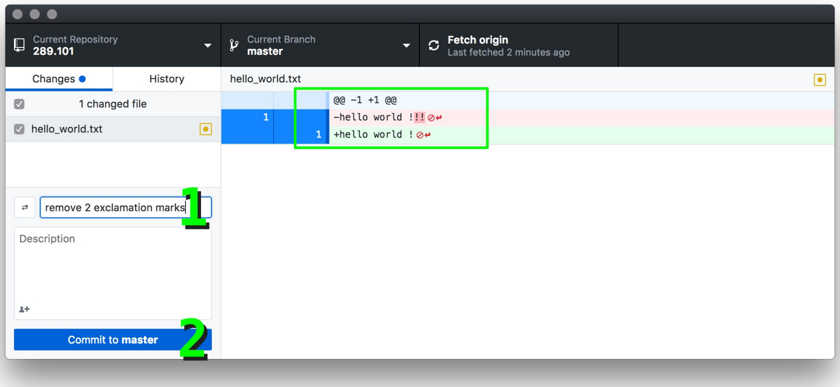


```
hello world !
```

Save the file and switch back to GitHub Desktop. Deletions are indicated in red, and insertions in green, along with the corresponding line numbers.

1. Add a commit summary;

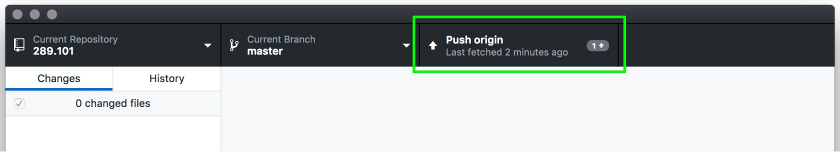
2. and click **Commit to master**.



```
@@ -1 +1 @@
-hello world !!!
+hello world !
```

remove 2 exclamation marks

You can continue to add as many commits as you see fit during your work session, but be sure to push the changes when you shut-down/log-out of the computer:



Styles (CSS) and Colours

This section steps you through a series of tasks involving HTML style attributes, colours, and CSS (Cascading Style Sheets).

Begin by creating a new file in your "289.101" directory named "styles.html". Add the basic document structure and a red, bold paragraph:

```
<!DOCTYPE html>

<html>

  <head>
    <title> Styles </title>
  </head>

  <body>
    <p style="color:red;"> <b>important text</b> </p>
  </body>

</html>
```

Now add another two red paragraphs:

```
  ...

  <body>
    <p style="color:red;"> <b>important text</b> </p>
    <p style="color:red;"> <b>more important text</b> </p>
    <p style="color:red;"> <b>and some more important text</b> </p>
  </body>
```

Looking at this code, it's clear that important text is bold and red. However, the capricious client of this fictitious website has now decided it must be italic and teal instead. Here goes:

```
  ...

  <body>
    <p style="color:#008080;"> <i>important text</i> </p>
    <p style="color:#008080;"> <i>more important text</i> </p>
    <p style="color:#008080;"> <i>and some more important text</i> </p>
  </body>
```

Okay -- so now the client wants to change it again! Time for a better approach! You have used **inline styles** up to this point, which are generally considered poor form. Inline styles are actually CSS code embedded within an attribute. We'll retrofit the code to use CSS class selectors instead:

```
<!DOCTYPE html>
```

```
<html>

  <head>
    <title> Styles </title>

    <style>
      .important {
        color: #008080;
        font-style: italic;
      }
    </style>
  </head>

  <body>
    <p class="important"> important text </p>
    <p class="important"> more important text </p>
    <p class="important"> and some more important text </p>
  </body>

</html>
```

Note how the `style="color:#008080;"` has been replaced with `class="important"`, and how the `.important { ... }` selector in your new `<style>` section corresponds to this. Now you can change the styling of all the 'important' text in one place. Your lecturer/tutor will explain the CSS selector/property/value syntax. If you need further information on this, refer to the following documentation:

- https://developer.mozilla.org/Learn/CSS/Introduction_to_CSS/Syntax
- https://developer.mozilla.org/Web/CSS/color_value

CSS comments are placed within a `/*` and `*/` and can span multiple lines. It's easiest to type these using the number pad! Here is an example of single- and multi-line comments:

```
    ...
    <style>
      /* this is a class selector */
      .important {
        color: #008080;
        font-style: italic;
      }
```

```
  /* look at me!
  I'm a multi-line comment */
...
```

You should also be aware of how CSS handles conflicts. For example, where you have two of the same selector, the one closest to the bottom of your code 'wins' -- in this case the important text will be orange and italic:

```
...
<style>
  .important {
    color: #008080;
    font-style: italic;
  }

  ...

  .important {
    color: orange;
  }
</style>
...
```

CSS also provides *tag selectors* for when you wish to affect elements based on their tag name (note that, unlike class selectors, they are not preceded with a `'.'` character):

```
...
<style>
  .important {
    color: #008080;
    font-style: italic;
  }

  h1 {
    background-color: orange;
  }
</style>
...
<h1> Some Heading with an Orange Background </h1>
...
<h1> Another Heading with an Orange Background </h1>
```

More *specific* selectors override less specific ones (for example, class selectors are more specific than tag selectors) while inline styles override everything.

CSS provides all sorts of features to control the visual appearance and layout of a web-page. You'll be covering many of these in the weeks to come.

Save your file.

Links & Paths

As HTML documents contain nothing much other than text and tags (even CSS is usually placed in a separate file), any images or links to other pages must be referenced using a file path. There are two types of paths: *absolute* and *relative*.

For now, create a new HTML document named "links.html". You'll need to add the basic document structure, of course -- i.e. `doctype`, `html`, `head`, `body`. Ensure that you save it in your 289.101 directory; the idea being that this repo will serve as a reference for when you forget how to do something.

Absolute Paths

Here is a hyperlink to Google as you would code it in an HTML document:

```
<a href="http://www.google.com"> Google </a>
```

Note how the `a` tag is used to wrap anything you wish to serve as the clickable link, and the `href` attribute for its destination.

Absolute paths will work regardless of where the document is deployed (provided a user's device is connected to the internet). It's easy to identify absolute links as they begin with the protocol (`http://` or `https://`). For example:

- `http://www.google.com`
- `https://github.com/about`
- `https://assets-cdn.github.com/images/modules/about/about-header.jpg`

Add the above as links in your "links.html" web-page. Save and test.

If you miss the workshop or need more information on hyperlinks, refer to this documentation:

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Creating_hyperlinks

Relative Paths

Relative paths require that the referenced documents are part of your website, i.e. that they sit somewhere inside your main/root website directory. The relative position of the file to the HTML document is critical, and you must ensure that you include any file extensions, for example:

- `<a href="about.html> About </a>`
- `<a href="countries/france.html> France </a>`

Slashes (`/`) and full-stop pairs (`..`) are used to navigate through child and parent directories. Take this arrangement as an example:

```
my_website
│  about.html
│  contact.html
│
└──countries
   │  france.html
   │  germany.html
   .
```

To link from the "about.html" file to the "germany.html" file, the code would read as:

```
<a href="countries/germany.html> Germany </a>
```

Make a new directory in your repo named "images"; inside it, create a new HTML document named "images.html":

```
289.101
│  ...
│  ...
│
└──images
   │  images.html
   .
```

In the next section, you will add an image to this webpage using a relative path.

Images

Images are inserted using the `img` tag. Note that the image tag is *self-closing* (has no closing tag), so it's best to write it with a trailing slash:

```
<img />
```

Download the workshop files from Stream and add the `cat.png` file to the images directory:

```
289.101
│  ...
│  ...
│
└──images
   │  images.html
   │  cat.png
   .
```

The `src` attribute is used to reference the image file that you wish to display:

```
<img src="cat.png" />
```

Correctly speaking, you should always add an `alt` attribute to your image tag which provides a short description about the image:

```
<img src="cat.png" alt="cute kitten" />
```

Why do you think the `alt` attribute is necessary?

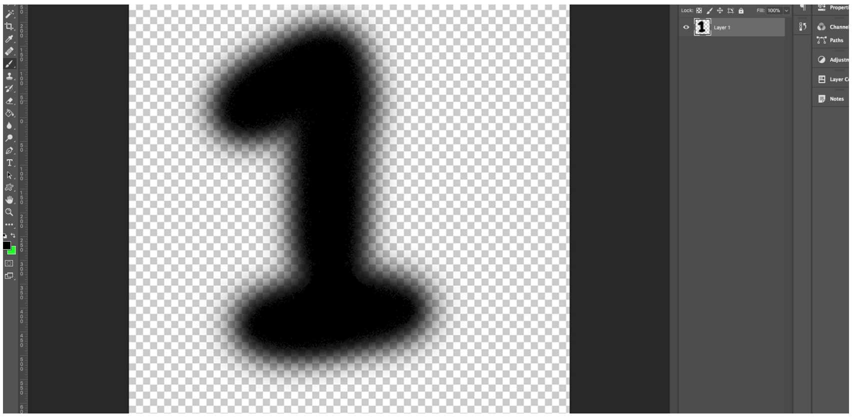
Add the above code to your "images.html" web-page. Save and test that it works.

Putting It All Together

Create a new directory named "mini_website". Within this, create a new "1.html" document (with some basic document structure code).

Then, using Photoshop:

1. create a new 500px by 500px RGB image with a transparent background;
2. using a soft-round brush, draw a crude "1";



3. select **File > Export > Save for Web (Legacy)**; your lecturer/tutor will now explain the various formats and options;
4. save the image as "1.png" in a new "img" sub-directory of "mini_website";

Your directory structure should now look like this:

```
mini_website
│  1.html
│
└──img
   │  1.png
   .
```

Organising things like this is good practice as having all of your HTML and image files bundled into a single directory can be messy to work with, especially as your website grows larger and more complex.

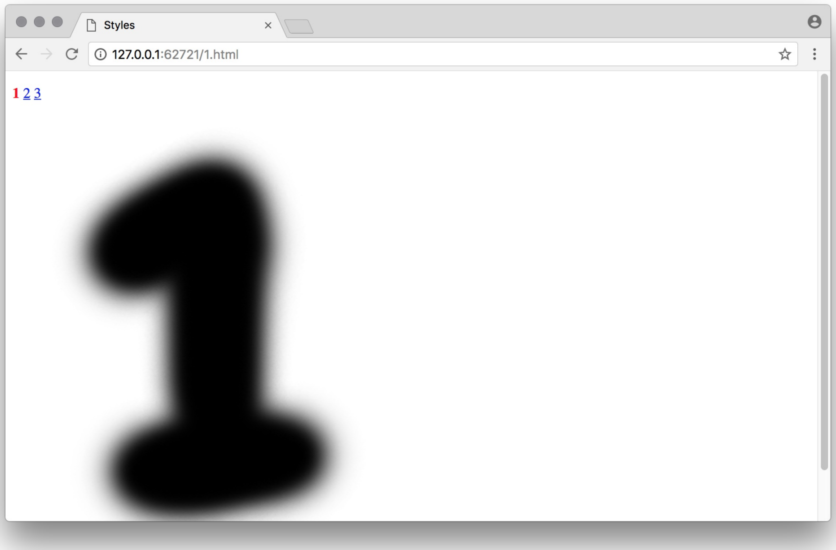
Now insert the image into the 1.html document:

```
    ...
    <img src="img/1.png" />
    ...
```

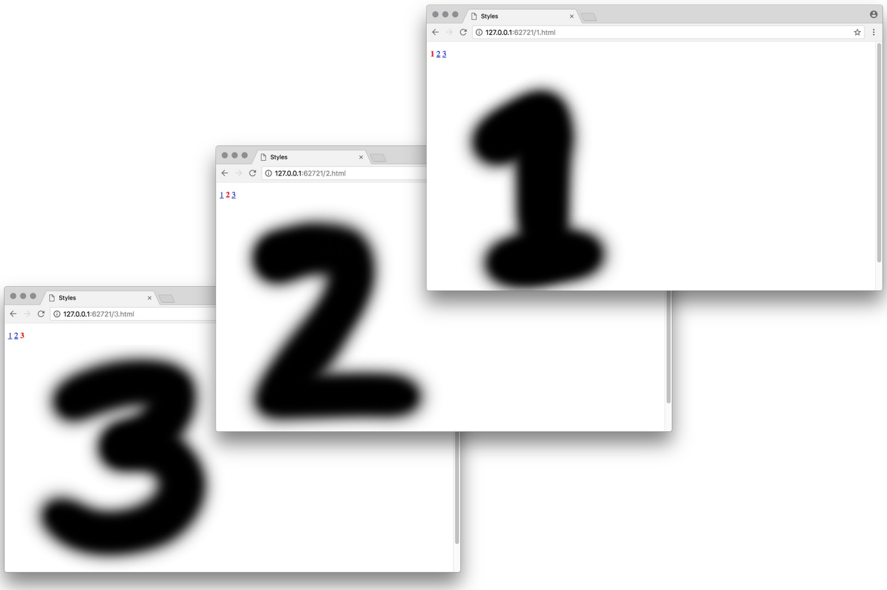
Add 3 hyperlinks -- one link each for a "1.html", "2.html", and "3.html" web-page. However, the link for the page you are currently viewing should be differentiated -- this helps the user situate him/herself. In this case make the 1 link red and bold using a class:

```
  ...

    <style>
      .selected {
        color: red;
        font-weight: bold;
        text-decoration: none;
      }
    </style>

  ...

  <a class="selected" href="1.html"> 1 </a>
  ...
```

Your web-page should look like this:



Now complete the website by creating web-pages for 2 and 3, then linking them all together.



end