**Jekyll** is a simple static site generator. Using Jekyll is a very common way of generating a "ready-to-publish static website" within seconds.

Jekyll is a Ruby gem (also known as a [RubyGem](http://guides.rubygems.org/what-is-a-gem/)) and can be installed from the command line.

Install Jekyll by typing the following command in the terminal:

gem install jekyll

To do so, we'll use Jekyll's new command and specify a directory name. The directory will contain all of your site's default content that can be customized later. For example, to generate a website in a directory called my-portfolio-site, we can type:

jekyll new my-portfolio-site

On the web, a server hosts your site's files and makes your website available for everyone to see.

However, viewing a website locally means that you're viewing the site on your own computer (hence the term "locally" or "local"). The site is not, however, available on the public Internet. Instead, your computer is acting as the server that hosts your site.

You can view your site locally by using Jekyll's serve command, like so:

jekyll serve

By default, the address for the local server that *Jekyll's* serve command starts is http://localhost:4000/.

The website that Jekyll generates differs from a website that you'd create on your own. It offers a standard directory structure, as well as components that help speed up development.

It's important to understand Jekyll's default directory structure and contents of your site:

1. **\_config.yml** - This is a configuration file that contains many values that need to be edited only once. These values are used across your site, for example, your site's title, your e-mail, and more. Note that this is a **.yml** file.
2. **\_includes/** - This directory contains all the partials (code templates that keep you from repeating your code over and over) that your site uses to load common components, like the header and the footer.
3. **\_posts/** - This directory is where [blog posts](https://en.wikipedia.org/wiki/Blog) are stored. New blog posts can be added and will be rendered with the site's styling, as long as the file name follows Jekyll's standard naming convention.
4. **\_layouts/** - This directory contains templates that are used to style certain types of posts within the site. For example, new blog posts will use the HTML layout defined in post.html.

**GitHub Pages** are public webpages that are hosted and published through GitHub. GitHub Pages offers extensive integration and support for Jekyll. By using both, you'll benefit from:

* Easy setup
* Troubleshooting your site
* Updating and maintaining your site

To specify the repo using Git, we'll have to add the remote and label it as the origin.

1. The remote is the URL of the repo that will store your site's contents.
2. The origin is an alias for the remote. You can think of an alias as an abbreviation or a substitute name. This means that instead of having to always type the lengthy remote URL over and over again, you can simply refer to it as origin later on.

In the terminal, you can add the remote with the following command:

git remote add origin https://github.com/your-user-name/your-user-name.github.io.git

**Important:** If you accidentally make a mistake when adding the remote URL, you can start over and remove the remote with the following command:

git remote rm origin

In this case, we want to push *all* of your site's content to the repo. This means we will do the following two things (in order):

**1.**Add all of your site's contents using the following Git command:

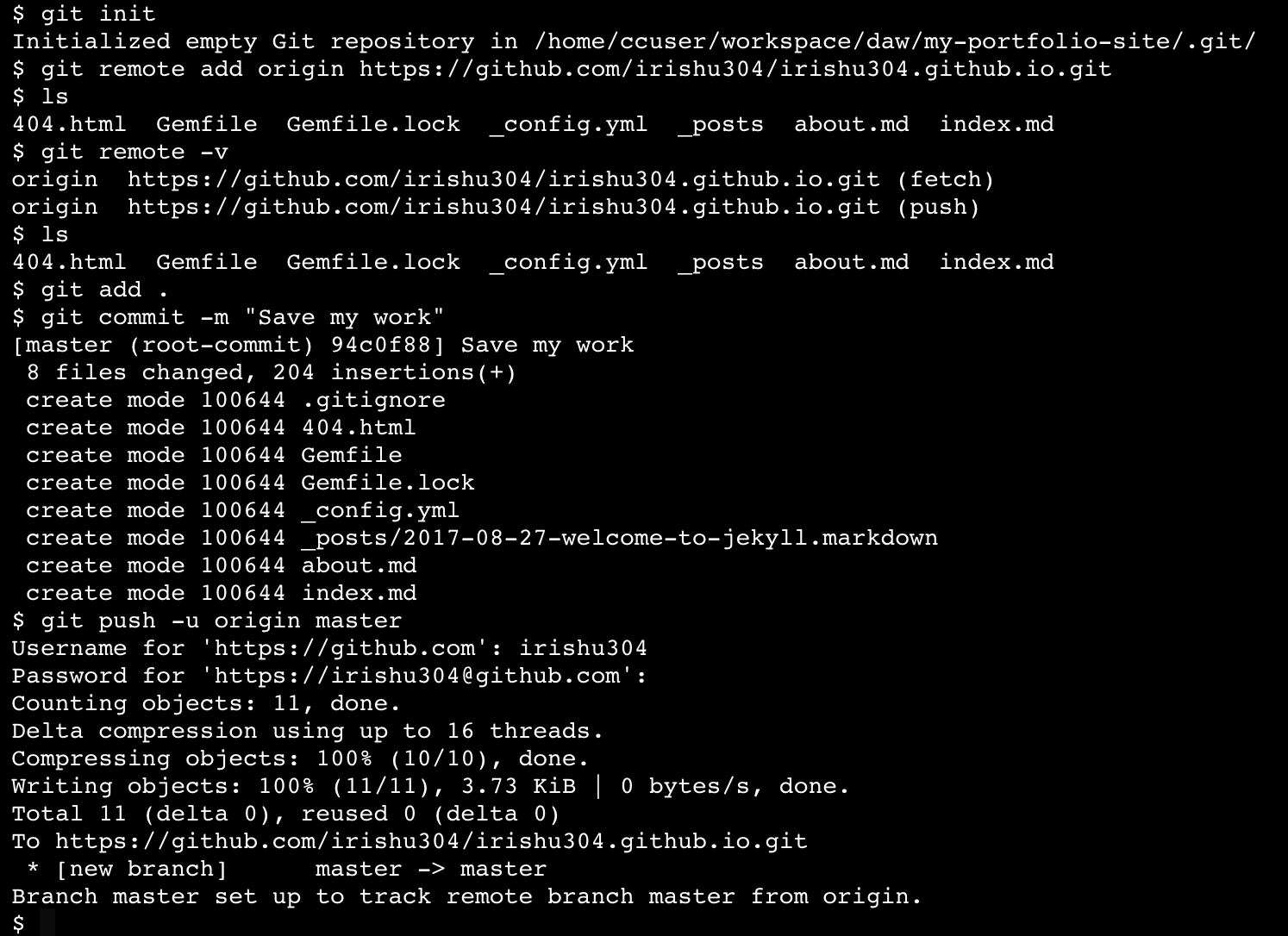
git add .

**2.**Save your changes using Git's commit command and the following commit message:

git commit -m "Save my work"

Once again, we'll use Git to help deploy your site. This time, we'll use Git's push command and push the contents of your site up to your repo using the following command:

git push -u origin master



In the last unit, you deployed your site and GitHub Pages assigned your site a default URL, or domain name.

In this unit, you'll purchase your own custom domain name and assign it to your GitHub Pages website. At the end of the unit, you'll be able to access your site using both your new domain name and your default GitHub Pages domain name.

We're going to use [Amazon Web Services (AWS)](https://aws.amazon.com/) to purchase your custom domain.

AWS is an industry standard suite of web infrastructure services used frequently by developers. The specific service we're going to use to purchase your domain name is called Route 53.

AWS offers many services used for web development like servers, databases, and networking configuration. There are two steps required:

1. Inform GitHub of the new domain name we'll be using (the one you purchased)
2. Set up DNS records in Route 53 that direct to GitHub

The new CNAME file in your repo informs GitHub that you're assigning a new custom domain name to your GitHub Pages site.

Next, we have to let the rest of the Internet know that we want to associate the custom domain name with your GitHub Pages site.

We can do this by creating DNS records, which are globally accessible records that map domain names to servers.

The DNS records are created inside of a Hosted Zone in Route 53. A Hosted Zone is essentially a group of DNS records for a single domain.

Domain names are associated with the correct DNS records by setting the domain name's *name servers*.

We must verify that the DNS records we create were actually created by the owner of the domain name (in this case, you).

By doing this, the owner of a domain name ensures that only they have exclusive control over their domain's DNS records.

Notice that the Hosted Zone for your domain name already has an NS (Name server) record. This record contains four values. These are the Hosted Zone's unique name servers. Take note of these values and copy them down somewhere.

We're going to start by creating an A record, which stands for Address record.

An A record directs a domain name to an IP address. This record will associate our new custom domain name with Github's servers.

When setting up a website, it's also conventional to also set up a www subdomain. www stands for world wide web.

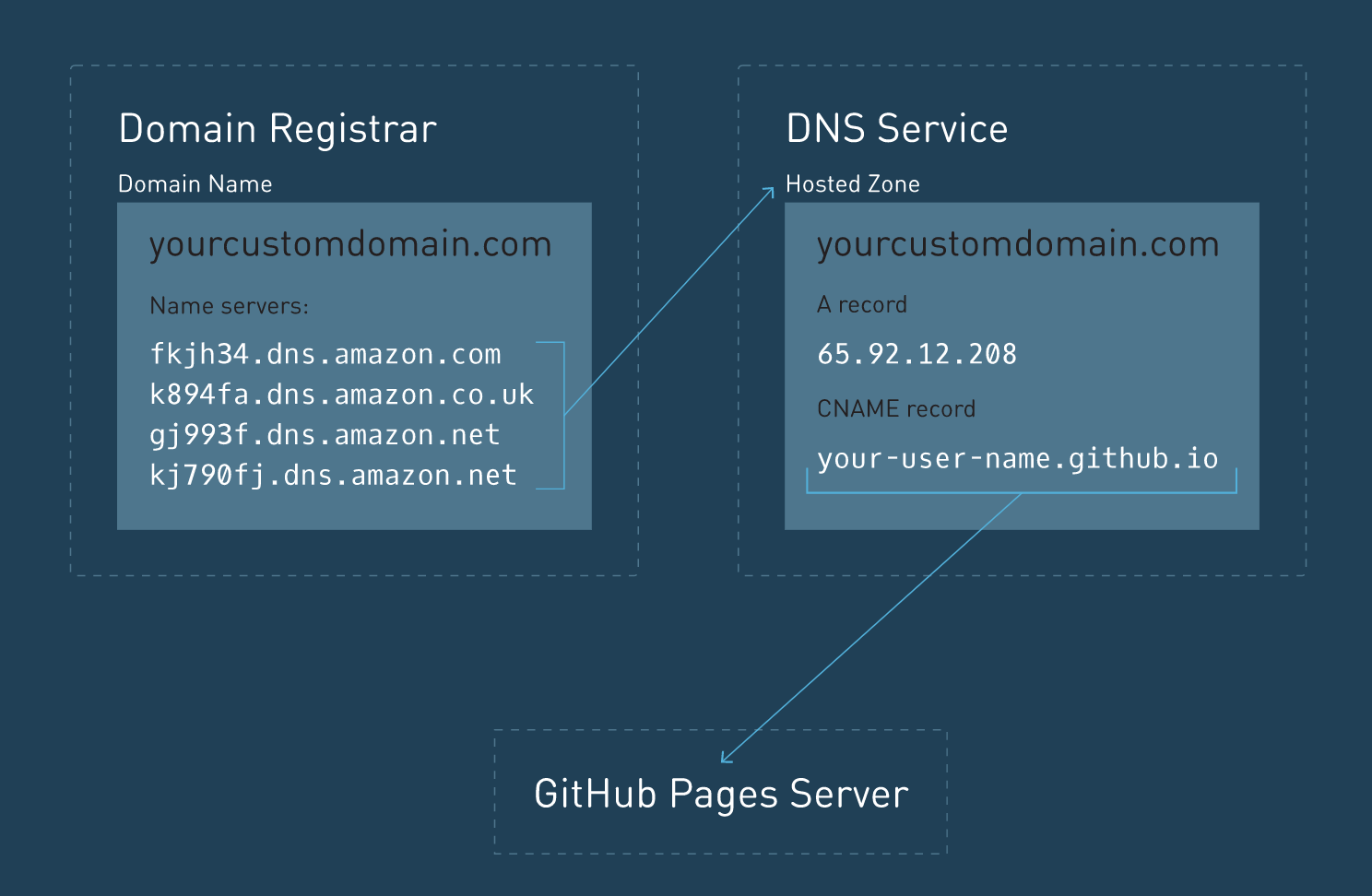
Subdomains are part of a main (or root) domain. For example, www.yourcustomdomain.com is a subdomain of the yourcustomdomain.com root domain.

We can set up a subdomain using a CNAME record, which stands for Canonical Name.

A CNAME record specifies that a domain name will be used as an alias, or substitute, for the true (canonical) domain name.

In Route 53, your domain name's Hosted Zone contains the following:

1. The NS (Name Server) record for your domain name. When a domain name is typed into a browser, the DNS looks to these name servers to help direct the request.
2. The A (or Alias) record. This record is used to direct requests of your domain name to GitHub's servers using their IP addresses.
3. The CNAME (or Canonical name) record. This record specifies what custom domain will point to your true (canonical) domain.



1. You purchased a custom domain name through a Domain Registrar, which in this case, is Route 53.
2. Four unique name servers were assigned to your custom domain name after your purchase.
3. To assign your custom domain name to your web site, you had to set up a Hosted Zone with multiple DNS records for your custom domain name. The Hosted Zone was set up within Route 53.
4. Inside of the Hosted Zone, the NS record was created automatically for you by Route 53. However, you created the A record and the CNAME record.
5. This setup allows you to visit your personal website with your new custom domain name, even though it's hosted on GitHub!