Deploy a website

1.
Install **Jekyll** by typing the following command in the terminal:
gem install jekyll
Wait for the gem to finish installing before moving onto the next exercise. A successful install will display the following:
Successfully installed jekyll-3.1.3
Parsing documentation for jekyll-3.1.3
Installing ri documentation for jekyll-3.1.3
Done installing documentation for jekyll
1 gem installed

To generate a new website, 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.

jekyll **new** personal-website
You can use Jekyll to view your site *locally*.
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 //previously cd to personal-website
This command **starts a local server that will server the files to your computer.** The serve command will also come in handy when you want to preview changes you make to your site.
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, which you can learn more about here.

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 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.

In order **to publish your site using GitHub Pages**, you'll need to **create a repository (repo) on GitHub**.
A GitHub repository is an online, central storage place where you can store files and all the versions of those files. We'll use the repo you create to store the contents of your website.
Your repo's name *must* also follow GitHub Pages' naming convention, otherwise your site will not publish at all.
Specifically, the **repo's name must be in the following format: your-user-name.github.io**
In the example above, you would replace your-user-name with your actual GitHub user name.

mkdir personal-website
cd personal-website
use **git init** in your personal-website folder

Next, Git needs to know what repo will store your site's content.
In this case, the repo will be the one you created on GitHub earlier.
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
In the example above, https://github.com/your-user-name/your-user-name.github.io.git is the remote URL that refers to the repository you created

on GitHub earlier. Again, you would replace your-user-name with your actual GitHub username.
Note: Make sure that your remote's URL is typed correctly. Otherwise, you risk a failed deploy.

git remote add origin https://github.com/bitaemi/bitaemi.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

Confirm that the remote was succesfully added, by typing the following:
git remote -v
This command lists all the Git remotes and their corresponding URLs.

origin https://github.com/bitaemi/bitaemi.github.io.git (fetch)
origin https://github.com/bitaemi/bitaemi.github.io.git (push)

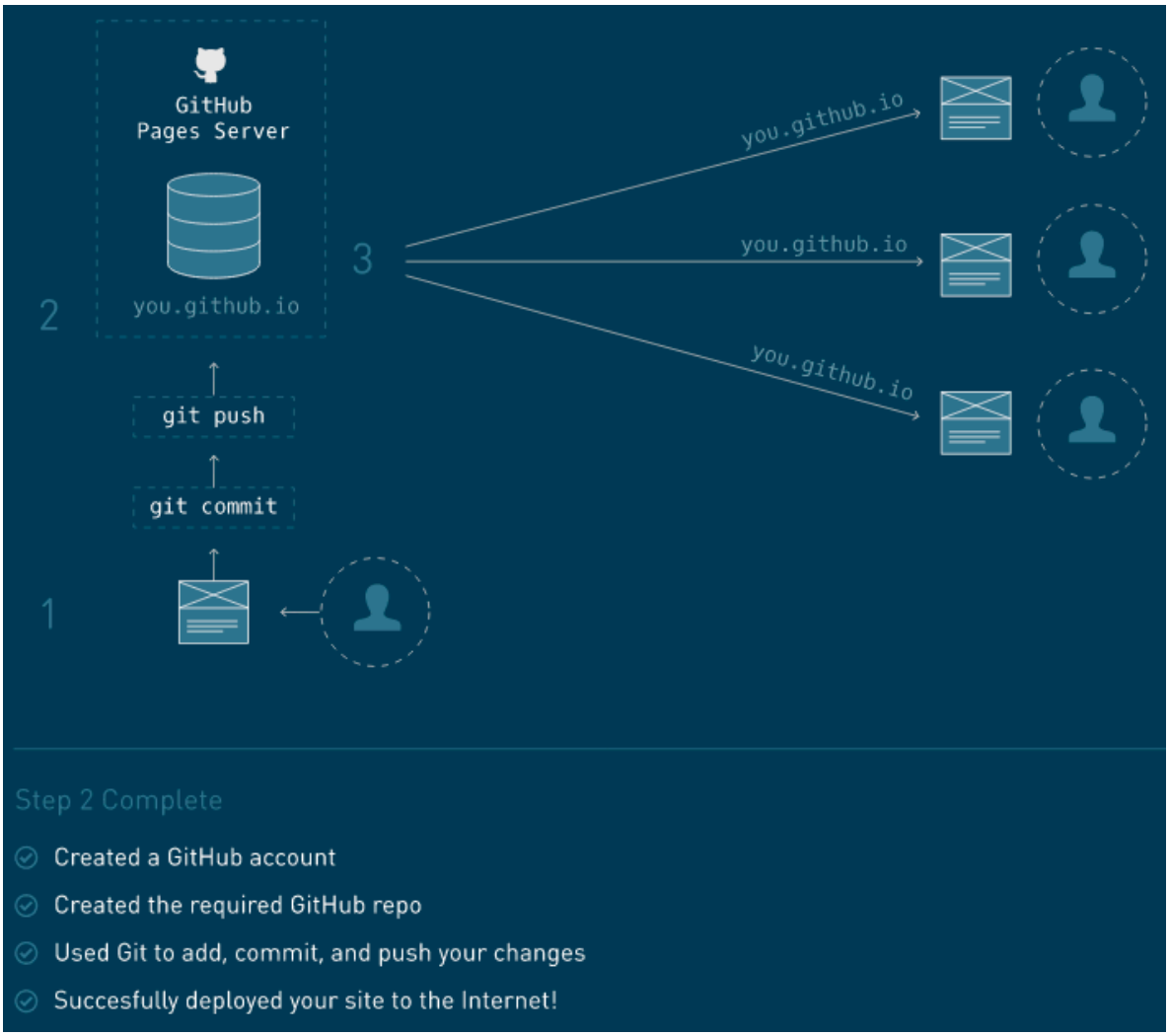
Add all of your site's contents using the following Git command:
git add .
.
Save your changes using Git's commit command and the following commit message:
git commit -m "Save my work"

[master (root-commit) ea04d63] Save my work
 20 files changed, 819 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 _config.yml
 create mode 100644 _includes/footer.html
 create mode 100644 _includes/head.html
 create mode 100644 _includes/header.html
 create mode 100644 _includes/icon-github.html
 create mode 100644 _includes/icon-github.svg
 create mode 100644 _includes/icon-twitter.html
 create mode 100644 _includes/icon-twitter.svg
 create mode 100644 _layouts/default.html
 create mode 100644 _layouts/page.html
 create mode 100644 _layouts/post.html
 create mode 100644 _posts/2017-07-25-welcome-to-jekyll.markdown
 create mode 100644 _sass/_base.scss
 create mode 100644 _sass/_layout.scss
 create mode 100644 _sass/_syntax-highlighting.scss
 create mode 100644 about.md
 create mode 100644 css/main.scss
 create mode 100644 feed.xml

create mode 100644 index.html

It's time to deploy your site!
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
Wait until the terminal stops outputting information. **Note:** You will be prompted to enter your GitHub credentials before you can push (i.e. username and password).
As you input your password, you'll notice that no text will be *visible* as you type — this is how all terminals work (as an added security measure). Be aware that although your password is not visible, the terminal is still accepting your input. Simply hit "Enter" on your keyboard after you are done typing your invisible password. If you typed it incorrectly, simply try again.
Finally, if you have two-factor authentication enabled for your GitHub account, this exercise will not pass for you. In that case, check out this resource.

$ git push -u origin master
Username for 'https://github.com': bitaemi
Password for 'https://bitaemi@github.com':
Counting objects: 27, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (25/25), done.
Writing objects: 100% (27/27), 9.33 KiB | 0 bytes/s, done.
Total 27 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/bitaemi/bitaemi.github.io.git
 * [new branch]   master -> master
Branch master set up to track remote branch master from origin.



The URL for your GitHub Pages site is: your-user-name.github.io, where your-user-name is your actual GitHub username.

Let's review what you accomplished in this unit:

1. Created a GitHub account
2. Created the required GitHub repo
3. Used Git to add, commit, and push your website to the repo
4. Succesfully used GitHub Pages to deploy your site to the Internet!

Note that your site's URL matches your repo's name — a GitHub Pages requirement. But what if you want to change the URL to a custom URL?

In the next unit, you'll learn how to keep your site hosted on GitHub, but change the URL to something besides the GitHub Pages default.

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.

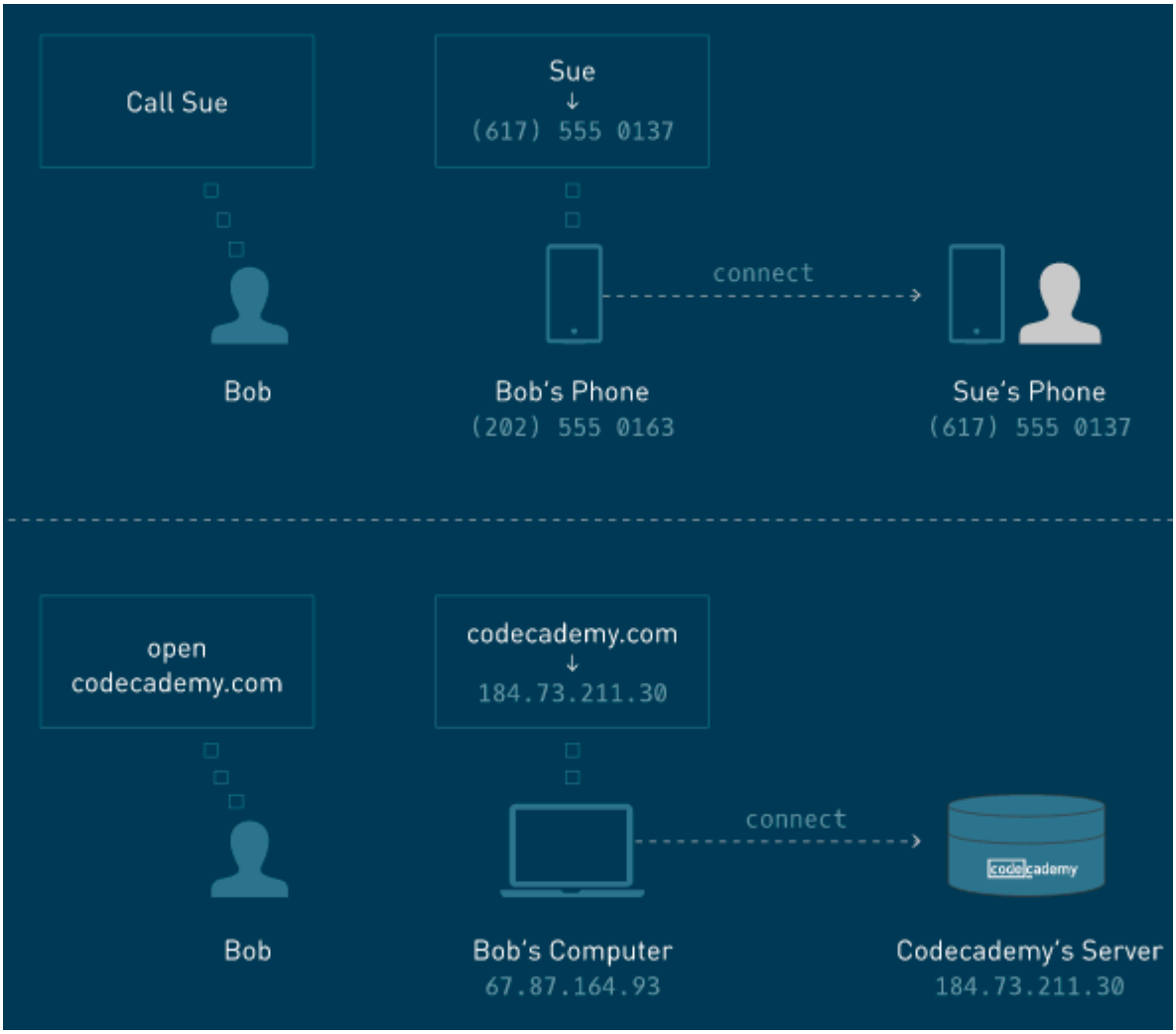
Before you choose a custom domain name, it's important to first understand what **domain names** actually are.

Domain names are human-friendly names that identify servers on the Internet. A global system known as the **Domain Name System** (DNS) is used for storing which domain names correspond to which servers.

For example, Codecademy's domain name is www.codecademy.com. When you

type the domain name into your browser, **your computer asks the DNS to identify which servers should receive the request in order to load our website.**

Note: This unit is optional. We'll walk you through the steps required to purchase a custom domain name and assign it to your GitHub Pages site. If you do not want to purchase a domain name, you won't have to. However, feel free to follow along to see what steps are required to do so (for future reference).



We're going to use Amazon Web Services (AWS) 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.

If you don't already have an account, create one at https://aws.amazon.com.

AWS uses a thorough verification process. You'll have to confirm the following:

1. Contact information
2. Payment information (*no* purchase is necessary to open an account, but a debit or credit card is)
3. Identity verification (a PIN entered via a phone call)
4. Support Plan (there is a free plan)

Remember, creating an AWS account is free — there are no required purchases. The only purchase that you'll (optionally) make will be the purchase

of your custom domain name.

Route 53 allows you to search the availability of a domain name you have in mind. It also offers many suffixes, like .com, .io, .me, and .pizza. If the domain name you want is unavailable as a .com for example, you can try using a different suffix.
The suffixes of domain names are known as **top-level domains (TLDs**).
Different TLDs cost different annual prices.
Note: .com domains are the most popular and are therefore generally unavailable (or expensive).
your new domain name doesn't work yet — you can't visit it in your web browser. We have to connect it to your GitHub Pages website first.
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
3. Open GitHub and access the repo you created earlier titled your-user-name.github.io.

4. Click the "New file" button.

5. Name the new file **CNAME**. Do not add a file name extension.

6. In the file, on line 1, type the **custom domain name** you just purchased in the following format:

yourcustomdomain.com
You may have purchased a domain name with a TLD other than .com. In that case, make sure to use that TLD when creating the **CNAME** file.
5. Commit the new file.
6. Under the title of the repo, click on "Settings." Scroll down to the section titled "GitHub Pages" and confirm that there is a message *similar* to the following:
Your site is published at http://yourcustomdomain.com.
7. Try navigating to your website in your browser using your new domain name. It still doesn't work! Let's finish setting it up in the next few exercises.

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.

Access Route 53 once again. On the left side of the page, click on the title that says "Hosted Zones." Notice that you have a Hosted Zone for your new domain name. Click on it to open it.
Domain names are associated with the correct DNS records by setting the domain name's *name servers*.
After a domain name is typed into a browser, the computer first retrieves the name servers that correspond to that domain name. The name servers are important because they're responsible with providing the computer with other important information (in the form of DNS records) associated with the domain name.
Setting your domain's name servers is important. The DNS is a global system, which means that anyone can create DNS records. 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.

1. 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.

2. On the left hand side, under "Domains," click on "Registered domains." Then, click on your domain name.

3. On the right hand side of the page, locate the section titled "Name Servers." Notice that these are the same name servers that your Hosted Zone's NS record contained. Route 53 did this for you automatically.