Site Generators

- Jekyll
- Hexo
- Hugo

Jekyll

Jekyll can be installed by following the Installation Instructions hosted on the official website. So if you are on Ubuntu Linux,

```
sudo apt-get install ruby-full build-essential zlib1g-dev
echo '# Install Ruby Gems to ~/gems' >> ~/.bashrc
echo 'export GEM_HOME="$HOME/gems"' >> ~/.bashrc
echo 'export PATH="$HOME/gems/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
gem install jekyll bundler
```

After running the above to install Jekyll, we just need to pick a project or theme to start our Jekyll server with. Check out GitHub or Google for some Jekyll themes, and clone one. Keep in mind to store this repository into a location on your host where you'd like to store the root of your blog, since we will use this repository to start our Jekyll server it will store all of our configuration files.

```
git clone https://github.com/streetturtle/jekyll-clean-dark
cd jekyll-clean-dark

# Build the site with the contents in the current directory
jekyll build

# Serve the site on a webserver and detach the process from this shell
jekyll serve --detach
```

Running `jekyll build --watch` on an active shell will check for any changes to the sites root directory and build them out to the published site.

When creating a new post, `bundle exec jekyll post "Name"` can be ran to create a draft post. The same command has various other uses that will help in the early stages of a blog -

```
Subcommands:
  docs
  import
  build, b              Build your site
  clean                 Clean the site (removes site output and metadata file) without
building.
  doctor, hyde          Search site and print specific deprecation warnings
```

```
  help                Show the help message, optionally for a given subcommand.
  new                 Creates a new Jekyll site scaffold in PATH
  new-theme           Creates a new Jekyll theme scaffold
  serve, server, s    Serve your site locally
  draft               Creates a new draft post with the given NAME
  post                Creates a new post with the given NAME
  publish             Moves a draft into the _posts directory and sets the date
  unpublish           Moves a post back into the _drafts directory
  page                Creates a new page with the given NAME
```

Specifically, page, (un)publish, post, draft, serve, new, and build are the commands we will use heavily.

When generating a new post using `bundle exec jekyll post "Name"`, you might notice at the top of the new post generated in `./_posts/` there is a block describing the page to Jekyll. This is an important block and can be used to customize how the page is displayed based on the arguments provided. For example, below is a default new post `testpost`

```
---
layout: post
title: testpost
date: 2019-09-01 12:00
description: A test page
tags:
- test
---
# Test
```

The above block does nothing but describe our page to Jekyll, up to our first header that is actually output to our post `# Test`. The layout is described in a corresponding file stored in the `./_layouts/` directory.

If we wanted to add a custom table of contents, for example, when running github.com/allejo/jekyll-toc Jekyll theme we can simply add an argument to our header and it will create a table of contents with anchored links to each topic header automatically, just by adding `toc: true` below.

You can also customize it by styling `.toc` class in `**theme.scss**`

```
layout: post
title: testpost
date: 2019-09-01 12:00
description: A test page
```

```
tags:
- test
toc: true
```

Now if we use markdown carefully Jekyll will automatically create a nice table of content based on the content of our post, and the structure / sizes of the headers for each topic within. (The above solution is based on github.com/allejo/jekyll-toc)

To display raw code, we'll need to use some Liquid syntax -

```
{% highlight md %}
{% raw %}
code
{% endraw %}
{% endhighlight %}
```

Here, we should also be sure to define the langue for syntax highlighting with `{% highlight md %}`

When trying out new themes, some images or symbols may not work correctly. In chasing these down, we will need to use a bit of HTML, CSS, Liquid, and Mardown. For example, I noticed a symbol or image that was used for bullet points in a list of tags was broken on my specific theme, appearing as an empty box instead. To track this down, `tree -L 2` was useful in learning the layout of this unfamiliar project quickly. Eventually, through viewing the files related to tags within my theme, I found that the sidebar itself was an `include` in the below statement of `./_layouts/post.html` -

```
<div class="col-md-3 hidden-xs">
  {% include sidebar.html %}
</div>
```

So, this pointed me to check out the `./_includes/` directory, where I found the below file - `./_includes/sidebar.html`

```
<div class="sidebar ">
  <h2>Recent Posts</h2>
  <ul>
    {% for post in site.posts limit:5 %}
    <li><a href="{{ post.url | relative_url }}">{{ post.title }}</a></li>
    {% endfor %}
  </ul>
</div>
```

```
<div class="sidebar">
  <h2>Tags</h2>
  <ul class="sideBarTags">
    {% assign tags = (site.tags | sort:0) %}
    {% for tag in tags %}
    <li>
      <a href="{{ '/tag/' | append: tag[0] | relative_url }}" data-toggle="tooltip" data-
placement="right" title="{{ tag[1].size }}">
        <span>{{tag[0] }}</span></a></li>
    {% endfor %}
  </ul>
</div>
```

The file above pointed me to the CSS classes below associated with each of the tags (`<ul>`) shown by the sidebar

```
<div class="sidebar">
  <h2>Tags</h2>
  <ul class="sideBarTags">
```

I knew these would be stored in the `./assets/` directory within the root of our Jekyll project, where I found `./assets/css/themes.scss` contained the below CSS statement `content: '\f02b';` - This was the symbol in the sidebar of my theme that was causing issues

```
//*********************************************************
//                    Sidebar
//*********************************************************
.sidebar li {
  margin-top: .7em;
  line-height: 1em;
}
ul.sideBarTags {
  list-style: none;
  li:before {
    content: '\f02b';
    font-family: 'FontAwesome';
    float: left;
    margin-left: -0.8em;
  }
}
```

By changing it, and also tweaking some other settings below, I was able to improve the look of the sidebar.

```
//*************************************************************
//                    Sidebar
//*************************************************************
.sidebar li {
  margin-top: .7em;
  line-height: 1em;
}
ul.sideBarTags {
  list-style: none;
  li:before {
    content: '-';
    font-family: 'FontAwesome';
    float: left;
    margin-left: -0.8em;
  }
}
```

Hexo

Hexo is a static site generator that converts Markdown syntax into prebuilt dynamic themes. Hexo's use of Node.js, HTML, and CSS / YAML makes for a simple, scalable website that is easy to maintain, build upon, or migrate.

Check out the book on Systemd Services for a quick example of running a hexo blog as a system service that starts automatically on reboot and can be controlled via `systemctl` and `journalctl`

Installing Hexo

Installing Hexo is done with npm, and requires Node.js. To meet these requirements, we first need to install both of these tools. Luckily, there are scripts and commands to help us do so.

```
# Install NVM to prep for Node
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
# Close and reopen your terminal to start using nvm or run the following to use it now:
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm
bash_completion

# Install Node.js with NVM
nvm install stable
```

Now that we have everything we need, all that's left to do is install Hexo -

```
# Install Hexo with NPM
npm install -g hexo-cli
```

Creating Hexo Sites

Creating a site with Hexo requires that we first have an empty directory that we want to build our site within. Create this directory, then run `hexo init` to generate the files we will use to build our site.

```
mkdir /site/
hexo init /site/
```

Our site is generated within `/site/`, and within it we can find new files and folders Hexo generated when we ran the `hexo init` command on the directory. The basic structure of a hexo blog can be seen below -

```
├── _config.yml
├── db.json
├── node_modules
│   ├── JSONStream
│   ├── a-sync-waterfall
│   ├── abbrev
│   ├── accepts
│   │
│   ...More...
├── package.json
├── public
│   ├── 2019
│   ├── archives
│   ├── css
│   ├── fancybox
│   ├── images
│   ├── index.html
│   ├── js
│   ├── lib
│   └── tags
├── scaffolds
│   ├── draft.md
│   ├── page.md
│   └── post.md
├── source
│   ├── _drafts
│   └── _posts
└── themes
    ├── cactus
    └── landscape
```

The `_config.yml` file contains all of the settings for our site, and its important to step through them carefully to make the changes that may need to be made in order for the site to work properly.

The `source` directory contains all of our drafts, posts, and pages. This blog has no additional pages, so none are seen here, but they would be represented as additional directories within the `source` directory, titled with the page name. These page directories contain only the index.md file that is the content.

`scaffolds` are the defaults used when generating a draft, post, or page using Hexo. These are useful to edit when attempting to configure your changes to initialize with some settings or information, which speeds up the process of adding new content.

The `public` directory is generated by Hexo, and we don't need to worry much about it.

So, if we wanted to migrate our site from Hexo, the only files we'd need to worry about keeping are the Markdown files within `source`. Markdown is widely used across many tools and applications, so finding a new way to use these posts and pages is likely and easy to manage.

Installing themes

Hexo uses prebuilt themes for generating Markdown into webpages. I chose Cactus Dark for this site, and made changes where I seen fit.

Themes are installed by simply navigating to the root directory of your site, and cloning the repository into the `themes` directory. See the commands below for an example of installing this theme to a new Hexo site (without any modifications you may see here).

```
cd /site/
git clone https://github.com/probberechts/hexo-theme-cactus.git themes/cactus
```

Now within your root directory, modify the `_config.yml` to point to the cactus theme we just added. The default theme and value within the `_config.yml` is landscape, that value points to the theme directory we cloned above.

Its important to note that changes to the theme will need to take place within the `/site/theme/cactus/` directory, which may contain a large set of files that are at first unfamilliar to you. After some time poking around your own small site, spend some time looking through the directories and files within your themes directories. They will often provide good examples to use on your own.

Hexo Commands

Also found by running `hexo -h`, see the below help text for a list of commands when using Hexo

```
Usage: hexo <command>
```

```
Commands:
  clean     Remove generated files and cache.
  config    Get or set configurations.
  deploy    Deploy your website.
  generate  Generate static files.
  help      Get help on a command.
  init      Create a new Hexo folder.
  list      List the information of the site
  migrate   Migrate your site from other system to Hexo.
  new       Create a new post.
  publish   Moves a draft post from _drafts to _posts folder.
  render    Render files with renderer plugins.
  server    Start the server.
  version   Display version information.

Global Options:
  --config  Specify config file instead of using _config.yml
  --cwd     Specify the CWD
  --debug   Display all verbose messages in the terminal
  --draft   Display draft posts
  --safe    Disable all plugins and scripts
  --silent  Hide output on console
```

For more help, you can use 'hexo help [command]' for the detailed information or you can check the Hexo Documentation

Git for Hexo

Even though GitHub Pages will host Hexo blogs for free, I choose to run mine on a VPS that I maintain myself out of personal interest. For that reason, my approach to using Git with Hexo is slightly different than the usual.

When running `hexo init` I noticed all it was doing was cloning a starter hexo repository and running `npm install` within, which kicked me off with a github repository that I had nothing to do with, generated theme files I didn't really need or want, and made moving my own site to a private repository a bit confusing. Initially, I thought the `hexo init` command was doing something to prepare the services on my system, and not just cloning a starters template to get me going.

>

> This section assumes you have a running Hexo site that you want to track with Git and clone onto another system. Everyone starts somewhere, and running `hexo init` is a great place to start. Once you have your own hexo site setup, you can follow these steps to track it with Git.

After noticing this, it was quite easy to setup a private github repository that could be cloned onto any host just as quickly as `hexo init`, though we will need to run *three commands* instead of one. If you haven't already, run `sudo rm -rf .git*` from within the root directory of your hexo blog. This will remove Git from your directory so we can set it up with our own repository later.

First, grab the Hexo gitignore and create it within your hexo root directory.

```
.DS_Store
node_modules/
tmp/
*.log
.idea/
yarn.lock
package-lock.json
.nyc_output/
```

Then, we want to initialize a new git repository within our directory with `git init` and head over to GitHub to create your private repository. Once you've created a private repo, add your remote URL with `git remote add origin https://github.com/username/repo`. This is already nearly the end of the process.

Depending on your setup, you may not need to modify your themes layout, css, and various other settings. I have done all of these things, and made tracking such changes a bit more confusing, so I'll go over my process for keeping up to date with my theme's updates on Git while preserving the changes I've made myself.

Add your theme as a submodule within your `hexoroot/themes/` directory with `git submodule add https://github.com/probberechts/hexo-theme-cactus themes/hexo-theme-cactus`. This will allow git to clone the updated contents of this repository when you clone using the `--recursive` flag.

Add the rest of your files to git and make your initial commit. Head over to your new server and you can get an exact copy of your website running with a few simple commands -

```
git clone --recursive https://github.com/username/repo
cd repo && npm install
hexo server
```

Now, provided you've already configured NGINX on the new host to point to the appropriate location, you can visit your IP or domain in a browser and see a your full blog has been easily copied across the web in three commands.

Markdown Guide

Below we can see the basic syntax used when writing raw markdown pages.

```
Headings
# Heading level 1
OR
Heading level 1
===============

## Heading level 2
OR
Heading level 2
---------------

### Heading level 3
#### Heading level 4
##### Heading level 5
###### Heading level 6

Italics Text
Italicized text is the *cat's meow*.
Italicized text is the _cat's meow_.
A*cat*meow

Bold text
I just love **bold text**.
I just love __bold text__.
Love**is**bold

Italics and Bold text -
This text is ***really important***.
This text is ___really important___.
This text is __*really important*__.
This text is **_really important_**.
```

```
Quotes
> Dorothy followed her through many of the beautiful rooms in her castle.
>
>> The Nested Blockquote

Blockquotes can contain elements
> #### The quarterly results look great!
>
> - Revenue was off the chart.
> - Profits were higher than ever.
>
>  *Everything* is going according to **plan**.



Lists

1. First item
2. Second item
3. Third item
4. Fourth item

OR

- First item
- Second item
- Third item
    - Indented item
    - Indented item
- Fourth item

Inline `code`

Horizonal Rules

***

---

_____________________
```

```
Links
My favorite link is [Duck Duck Go](https://duckduckgo.com)
<https://www.markdownguide.org>
<fake@example.com>
I love supporting the **[EFF](https://eff.org)**.
This is the *[Markdown Guide](https://www.markdownguide.org)*.

Images
![Tux, the Linux mascot](/assets/images/tux.png)
```

The examples above were taken from The Official Markdown Guide

nginx.conf

Below is a basic `nginx.conf` that serves as an example of passing local traffic to the port running Hexo.

```
# A simple nginx.conf showing how to pass traffic to a docker container
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;
    return 301 https://www.domain.com$request_uri;

  }


  # SSL - domain.com
```

```
  server {
    server_name domain.com www.domain.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;

    # Pass to container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }

}
```

Troubleshooting layouts

Sometimes a layout may either need to be adjusted to your needs or corrected to fix some issue, which is easily done by modifying the CSS at `hexoblogroot/themes/landscape/layout/` where `laylout` is a hexo theme I have installed on my hexo server.

For me, the files in this directory are seen below

```
hexoblogroot/landscape/layout/
.
├── _colors
│   ├── classic.styl
│   ├── dark.styl
│   ├── light.styl
│   └── white.styl
├── _extend.styl
├── _fonts.styl
├── _highlight   # Note that this directory controls syntax highlighting :)
├── _mixins.styl
├── _partial
│   ├── archive.styl
│   ├── article.styl
│   ├── categories.styl
```

```
│   ├── comments.styl
│   ├── footer.styl
│   ├── header.styl
│   ├── index.styl
│   ├── pagination.styl
│   ├── post
│   │   ├── actions_desktop.styl
│   │   └── actions_mobile.styl
│   ├── search.styl
│   ├── tags.styl
│   └── tooltip.styl
├── _util.styl
├── _variables.styl
├── rtl.styl
└── style.styl
```

These files contain the CSS that will be applied to the generated output of the static site generator after parsing your markdown. If you visit your site and notice that some element is not interactable, its probably being overlapped by another element. To check if this is the case, right click-> inspect element and hover over the HTML options at the bottom of your screen until the issue is highlighted. Then you'll notice some CSS is made available to you that describes how the element is being displayed. This is the same CSS in the files above, and if you make any live edits to the site using the inspector that you want to remain persistant on the site, apply those changes in the files above.

For me, I had an issue that was only seen on the desktop version of my sie. So, we look within `hexoblogroot/themes/landscape/layout/_partial/post/actions_desktop.styl` and see the issue within the first block -

```
header-post
  position: fixed
  top: 2rem
  right: 0
  display: inline-block
  float: right
  z-index: 100
```

The issue for me was `z-index` was causing the element to lay overtop of the text within a post on my website. The Z index represents the 'depth' of the element in 3D space, so increasing this over that of another element would overlay it. So, to fix this we just remove `z-index` and save the file! The changes are applied instantly when saving if the hexo server is kept running.

Generating favicons

For many websites you'll notice icons are consistent between all devices and locations, whether the icon is on someones desktop on their phone or PC the same image is adjusted to suit the appearance. these are favicons, and can be generated easily at realfavicongenerator. After generating them, theres a few things you'll need to do -

Insert the snippet generated by the favicon generator into the `<head>` of your website. This will direct all platforms to their respective image / icon.

> Before uploading, you'll notice an option to specify where you'll place your favicon files on your webserver. For hexo, I chose `/images/`, and unzipped my generated favicon package to `hexoblogroot/themes/landscape/source/images` after removing the default icons there came with my theme.

```
<link rel="apple-touch-icon" sizes="180x180" href="/images/apple-touch-icon.png">
<link rel="icon" type="image/png" sizes="32x32" href="/images/favicon-32x32.png">
<link rel="icon" type="image/png" sizes="16x16" href="/images/favicon-16x16.png">
<link rel="manifest" href="/images/site.webmanifest">
<link rel="mask-icon" href="/images/safari-pinned-tab.svg" color="#5bbad5">
<link rel="shortcut icon" href="/images/favicon.ico">
<meta name="msapplication-TileColor" content="#da532c">
<meta name="msapplication-config" content="/images/browserconfig.xml">
<meta name="theme-color" content="#ffffff">
```

In a hexo blog, the `<head>` is located at `hexoblogroot/themes/landscape/layout/_partial/head.ejs`

Hugo

Official Documentation

Hugo Docker Image that is not maintained by Hugo, but it is recommended by the official documentation

Nice blog post on using Hugo with Docker nodinrogers - Hugo in Docker

Site Setup

```
sudo apt install docker-compose
mkdir /Code/Docker/hugo && cd /Code/Docker/hugo
vim docker-compose.yml
```

Paste the following into the `docker-compose.yml` -

```
server:
  image: klakegg/hugo:0.93.2
  command: server
  volumes:
    - ".:/src"
  ports:
    - "1313:1313"
```

Now we try to start the container, but there's an error; This is because I had not created a site previously, so hugo has nothing to serve.

```
docker-compose up
Starting hugo_server_1 ... done
Attaching to hugo_server_1
server_1  | Error: Unable to locate config file or config directory. Perhaps you need to
create a new site.
server_1  |        Run `hugo help new` for details.
server_1  |
hugo_server_1 exited with code 255
```

I don't have hugo installed locally, and one of the main attractions to Docker for me is avoiding installing service dependencies to my local system. So, I will use the following command to access

the `hugo` CLI using Docker, which will generate a new site for us. Note that the `/src/site` directory is relative to the container, and not our local system. This command will bind our CWD to the container's `/src` directory (`$(pwd):/src`), and then generate the new site at `/src/site` on the container. Since this is a bound directory, the generated site files will be available on our local system.

```
docker run --rm -it -v $(pwd):/src klakegg/hugo:0.93.2 new site /src/site

Congratulations! Your new Hugo site is created in /src/site.

Just a few more steps and you're ready to go:

1. Download a theme into the same-named folder.
   Choose a theme from https://themes.gohugo.io/ or
   create your own with the "hugo new theme <THEMENAME>" command.
2. Perhaps you want to add some content. You can add single files
   with "hugo new <SECTIONNAME>/<FILENAME>.<FORMAT>".
3. Start the built-in live server via "hugo server".

Visit https://gohugo.io/ for quickstart guide and full documentation.
```

```
kapper@xps:~/Code/Docker/hugo/site$ tree -L 2
.
├── archetypes
│   └── default.md
├── config.toml
├── content
├── data
├── layouts
├── public
│   ├── categories
│   ├── index.xml
│   ├── sitemap.xml
│   └── tags
├── resources
│   └── _gen
├── static
└── themes
```

But when we run `docker-compose up` we still get the same error.

```
docker-compose
up



Starting hugo_server_1 ... done
Attaching to hugo_server_1
server_1  | Error: Unable to locate config file or config directory. Perhaps you need to
create a new site.
server_1  |        Run `hugo help new` for details.
server_1  |
hugo_server_1 exited with code 255
```

This is because the hugo container expects the `/src` volume to be bound to the directory that contains our site. We previously generated our site within the `site` subdirectory, so we need to modify our `docker-compose.yml` to reflect this under the `volumes` section -

```
server:
  image: klakegg/hugo:0.93.2
  command: server
  volumes:
    - "./site:/src"
  ports:
    - "1313:1313"
```

Now we can start our hugo site, and visit it in our browser at `localhost:1313`, assuming you are using the default ports above.

```
docker-compose up


Recreating hugo_server_1 ... done
Attaching to hugo_server_1
server_1  | Start building sites …
server_1  | hugo v0.93.2-643B5AE9 linux/amd64 BuildDate=2022-03-04T12:21:49Z
VendorInfo=gohugoio
server_1  | WARN 2022/06/03 14:58:20 found no layout file for "HTML" for kind "home": You
should create a template file which matches Hugo Layouts Lookup Rules for this combination.
server_1  | WARN 2022/06/03 14:58:20 found no layout file for "HTML" for kind "taxonomy": You
should create a template file which matches Hugo Layouts Lookup Rules for this combination.
server_1  | WARN 2022/06/03 14:58:20 found no layout file for "HTML" for kind "taxonomy": You
should create a template file which matches Hugo Layouts Lookup Rules for this combination.
server_1  |
```

```
server_1  |                   | EN
server_1  | -------------------+-----
server_1  |   Pages            |  3
server_1  |   Paginator pages  |  0
server_1  |   Non-page files   |  0
server_1  |   Static files     |  0
server_1  |   Processed images |  0
server_1  |   Aliases          |  0
server_1  |   Sitemaps         |  1
server_1  |   Cleaned          |  0
server_1  |
server_1  | Built in 1 ms
server_1  | Watching for changes in /src/{archetypes,content,data,layouts,static}
server_1  | Watching for config changes in /src/config.toml
server_1  | Environment: "DEV"
server_1  | Serving pages from memory
server_1  | Running in Fast Render Mode. For full rebuilds on change: hugo server --
disableFastRender
server_1  | Web Server is available at http://localhost:1313/ (bind address 0.0.0.0)
server_1  | Press Ctrl+C to stop
```

At this point, the `-d` flag will be useful to us since we no longer need the debug output from the container. If you plan to have the container up for a while and won't be using this output, detach the container from your session -

```
docker-compose up -d


Starting hugo_server_1 ... done
```

Git Repository

You can track your entire site as a Git repository, if you want to. To to this, I would run the following commands, where the root of the repository ends up being the directory that contains the `docker-compose.yml` file. This bundles everything together so we can easily deploy the site on a new server if needed.

```
git init


Initialized empty Git repository in /home/kapper/Code/Docker/hugo/.git/
```

```
git status

On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        docker-compose.yml
        site/

nothing added to commit but untracked files present (use "git add" to track)
```

This output produces a warning telling us that we have an embedded repository, because I ran `git clone git@github.com:StaticMania/portio-hugo.git site/themes/portio` before initializing the repository. I wanted to track some test theme in the initial commit, but my hugo site is still not configured to use any theme. This way I can always restore the original configurations if I ever need them, or I could even just reference them on Github by looking in the repository history.

```
git add .
warning: adding embedded git repository: site/themes/portio
hint: You've added another git repository inside your current repository.
hint: Clones of the outer repository will not contain the contents of
hint: the embedded repository and will not know how to obtain it.
hint: If you meant to add a submodule, use:
hint:
hint:   git submodule add <url> site/themes/portio
hint:
hint: If you added this path by mistake, you can remove it from the
hint: index with:
hint:
hint:   git rm --cached site/themes/portio
hint:
hint: See "git help submodule" for more information.
```

The above output indicates that we could run `git submodule add git@github.com:StaticMania/portio-hugo.git site/themes/portio` to track the Portio theme as a submodule of our repository. For me, I won't - simply because Portio was the theme I used for my first attempt at configuring a Hugo theme. I plan to shop around for a theme that I like, and when I settle on one I will likely add that as a submodule.

Now, before we install our theme we could make a commit to track the site as it was newly generated by Hugo. Simply run `git commit` and type your initial commit message, then continue to work. You don't need to push to a remote until you're ready. I'll probably just track this project locally for a while and push it to a repo only when I go to deploy it. This way I will retain my repository history and have access to useful git commands to restore files or view diffs.

Themes

To start, let's look at the Portio Hugo theme on GitHub. This repo provides these install instructions which we will use to configure the theme below

```
cd themes/
pwd
~/Code/Docker/hugo/site/themes
```

```
git clone git@github.com:StaticMania/portio-hugo.git portio
fatal: could not create work tree dir 'portio': Permission denied
```

To fix the above error, change ownership of these files to your current user -

```
kapper@xps:~/Code/Docker/hugo/site/themes$ cd ../../
kapper@xps:~/Code/Docker/hugo$ sudo chown -R kapper:kapper site/
kapper@xps:~/Code/Docker/hugo$ ll
total 16
drwxrwxr-x  3 kapper kapper 4096 Jun  3 11:29 ./
drwxrwxr-x  3 kapper kapper 4096 Jun  3 10:13 ../
-rw-rw-r--  1 kapper kapper  114 Jun  3 10:58 docker-compose.yml
drwxr-xr-x 10 kapper kapper 4096 Jun  3 10:57 site/
```

```
git clone git@github.com:StaticMania/portio-hugo.git portio

Cloning into 'portio'...
remote: Enumerating objects: 1377, done.
remote: Total 1377 (delta 0), reused 0 (delta 0), pack-reused 1377
Receiving objects: 100% (1377/1377), 8.63 MiB | 16.73 MiB/s, done.
Resolving deltas: 100% (502/502), done.
```

```
cp -r site/themes/portio-hugo/exampleSite/* site/
```

Now we can restart our docker container and see the theme in the browser!

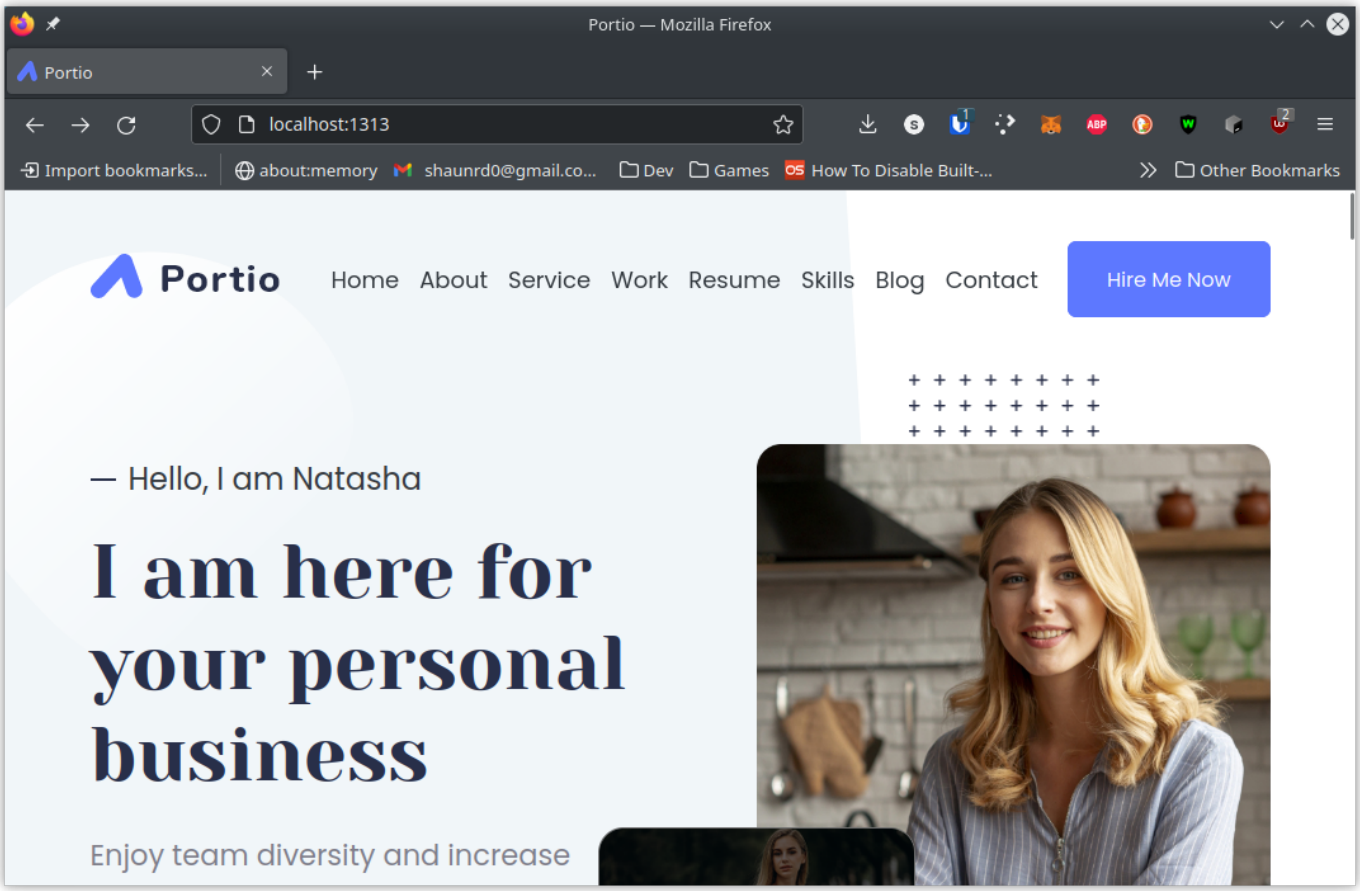
```
docker-compose down && docker-compose up -d`
```

Unfortunately at this point the theme did not render correctly for me. This was because I was targeting the `klakegg/hugo:0.93.2` docker image in my `docker-compose.yml`, and this theme requires Hugo Extended to render SASS/SCSS styling. So, we modify the `docker-compose.yml` to contain the below `ext-ubuntu` image instead -

```
server:
  image: klakegg/hugo:ext-ubuntu
  command: server
  volumes:
    - "./site/:/src"
  ports:
    - "1313:1313"
```

And when we restart the container with the following command once more, everything is working and the theme is rendered correctly!

```
docker-compose down && docker-compose up -d`
```



Creating Posts

In your `config.toml`, set the following option -

```
contentDir = 'content/'
```

Then we can create our `posts` directory, and run `hugo new content/posts/post-name.md` to create a new post.