Tarbell

Release 0.9-beta7

July 20, 2015

Contents

Tarbell makes it simple to put your work on the web, whether you're a team of one or a dozen. With Tarbell, you can collaboratively build beautiful websites and publish them with ease.

Tarbell makes use of familiar, flexible tools to take the magic (and frustration) out of publishing to the web. Google spreadsheets handle content management, so changes to your stories are easy to make without touching a line of code. Step-by-step prompts help you set up and configure your project, so that publishing it is a breeze.

New here? Read the Tarbell tutorial or install and configure Tarbell to get started.

Need help? File a ticket tagged "support" in Github.

Tarbell is named after Ida Tarbell, a distinguished muckraking journalist whose 1904 The History of the Standard Oil Company is a masterpiece of investigative reporting. Read more about her on Wikipedia.

Requirements

Tarbell requires Python 2.7 and Git (v1.5.2+). Tarbell does not currently support Python 3.

Tarbell does not work on Windows machines.

Anatomy of a Tarbell project

Tarbell projects are made up of four pieces:

- The core Tarbell library, installed when you run *pip install tarbell*
- A Tarbell blueprint directory
- Your project template files
- A Google spreadsheet (optional)

Using Tarbell

3.1 Installation

3.1.1 Install Tarbell with *pip install tarbell*

```
pip install tarbell
```

Note: Tarbell requires Python 2.7 and Git (v1.5.2+). Tarbell does not currently support Python 3.

Tarbell does not work on Windows machines.

To install on Ubuntu (tested with Ubuntu 14.04 LTS), install these dependencies first:

```
apt-get install build-essential git python-pip python-dev libncurses5-dev
```

To install with Fedora 21, install these dependencies from Edward Borasky's Tarbell docker image (untested).

```
yum install gcc git libyaml-devel make patch python-devel python-pip readline-devel tar
```

3.1.2 Configure Tarbell with *tarbell configure*

The `tarbell configure` command will set up your Tarbell settings

```
tarbell configure
```

Please consider setting up Google spreadsheet access for collaborative data editing and Amazon S3 settings for easy publishing.

3.1.3 Configure Google spreadsheet access (optional)

In order to allow Tarbell to create new Google Spreadsheets, you'll need to download a client_secrets.json file to access the Google Drive API. You can share this file with collaborators and within your organization, but do not share this file anywhere public.

Log in to the Google API Developer Console and create a new project:

Now click the "APIs & auth" tab. (Click on the "APIs" tab below that if it doesn't open automatically.) Enable Google Drive API.

Click the "Credentials" tab (right below "APIs") to create a client ID:

Select "installed app":

Now, go to the "consent screen" settings and give your project a name:

Whew! Now you can download the `client_secrets.json` file:

The file you download will be called something like `client_secret_longstringofrandomlettersandnumbers.apps.go`

Rename it to *client_secrets.json*.

Now, you do one of the following:

- Copy `client_secrets.json` to `~/.tarbell/client_secrets.json`.
- Specify the `client_secrets.json` download location when running `tarbell configure`. (By default, Tarbell will attempt to find this file in your `Downloads` directory.)

The first time a Tarbell command needs access to a Google spreadsheet (say, while you're running *tarbell configure*), you'll be prompted to authenticate

```
Go to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive&

Enter verification code:
```

Follow the link:

You should receive a confirmation code:

Enter it. If it works, you'll see:

    Authentication successful.

Now you can access and create Google spreadsheets to use with Tarbell projects.

Note: You need to visit the authentication page from the same machine that you are configuring Tarbell in order to avoid an OAuth Error. If you are using a remote machine, consider using the Lynx terminal browser. Alternatively, you can *pre-authenticate <remote-configuration.rst>*.

3.1.4 Configure Amazon S3

Generate keys for your Amazon S3 account. Add them during the Amazon S3 section of installation.

To generate keys, log into your AWS Console, click your name and select "Security Credentials".
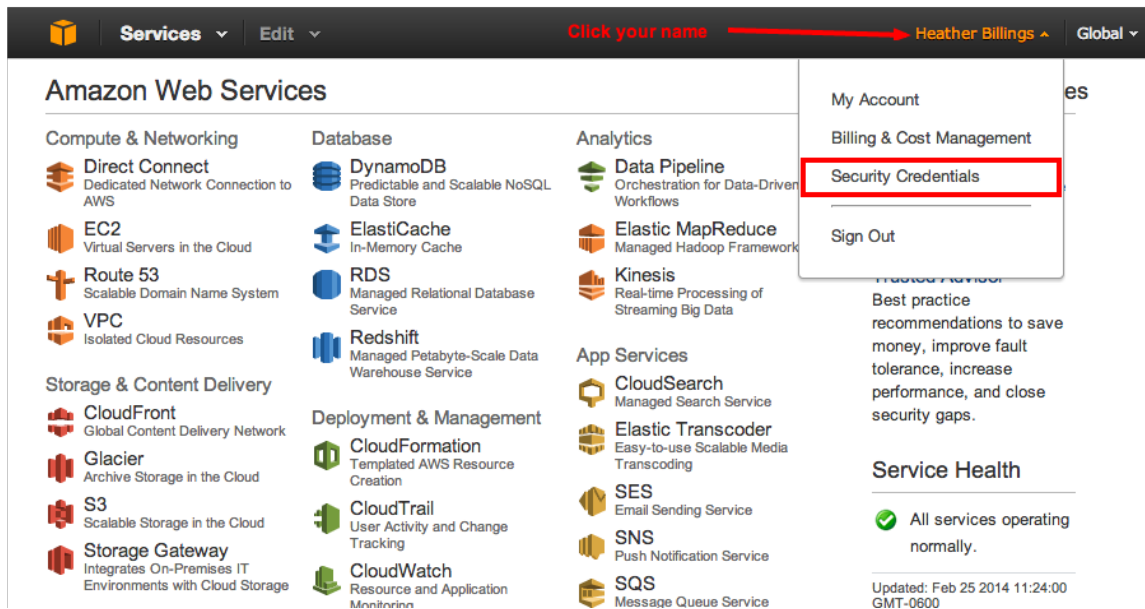
Don't worry about IAM users right now.
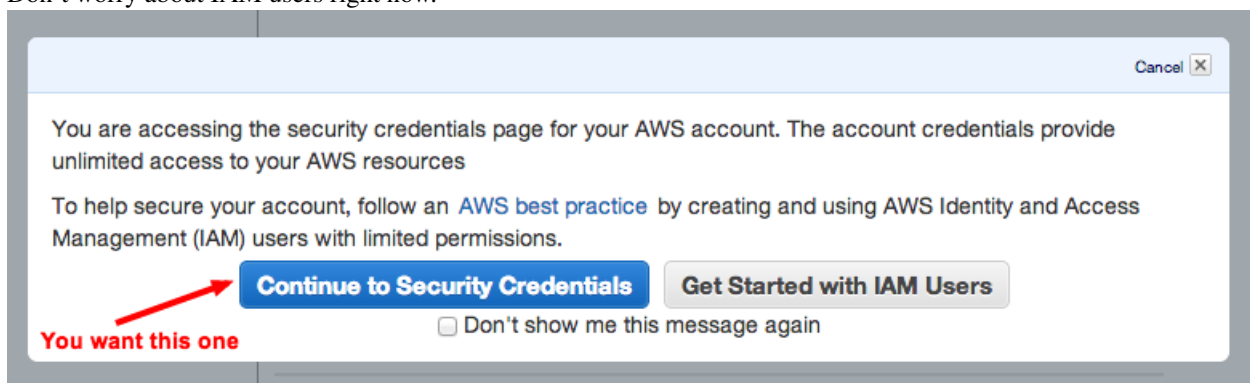
You should see a list of different sections. Click the section that reads, "Access Keys (Access Key ID and Secret Access Key)" and then the button, "Create New Access Key." Note that if you have existing keys, you can currently retrieve its Access Key ID and Secret Access Key from the legacy Security Credentials page (linked to in this section), but that Amazon plans to remove the ability to see this information soon.

Woohoo, now you can download your keys! You MUST do this now – Amazon only lets you download the keys on this screen. If you accidentally close the prompt, you can always delete the keys you just generated and generate a new pair.

Now you need to tell Tarbell what your AWS keys are. Run *tarbell configure*. After it checks to see if Google is configured, you'll get this prompt:

```
Would you like to set up Amazon S3? [Y/n] y

Please enter your default Amazon Access Key ID: (leave blank to skip)

Please enter your default Amazon Secret Access Key: (leave blank to skip)

What is your default staging bucket? (e.g. apps.beta.myorg.com, leave blank to skip)
```

If you don't already have a staging or production bucket, you can create one by going to the S3 management console and clicking "Create bucket."
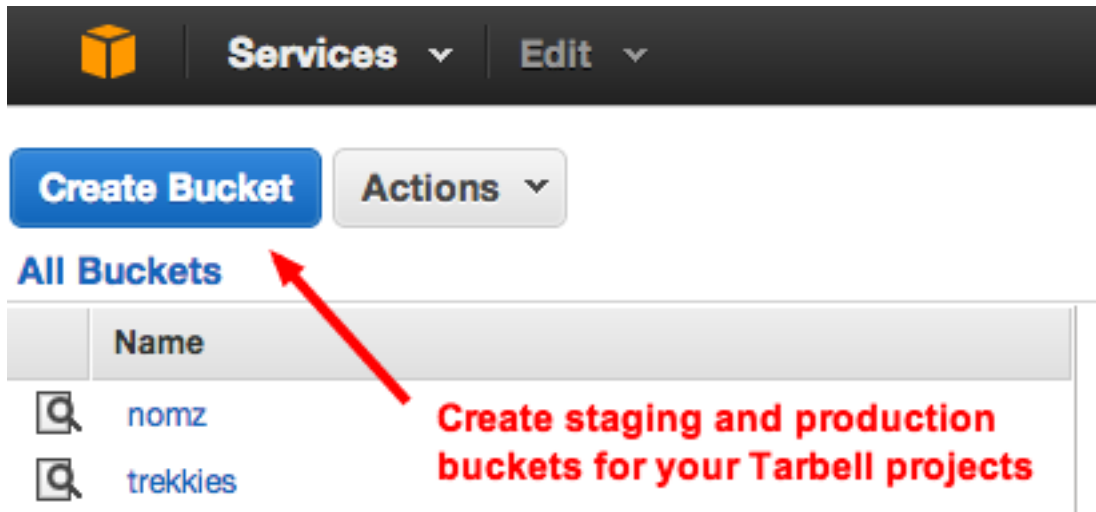
Just remember that when you name a bucket, it must be unique to AWS, not just your account. Like usernames, bucket names are shared across the entire Amazon system. (Which is silly, but that's how it is.)
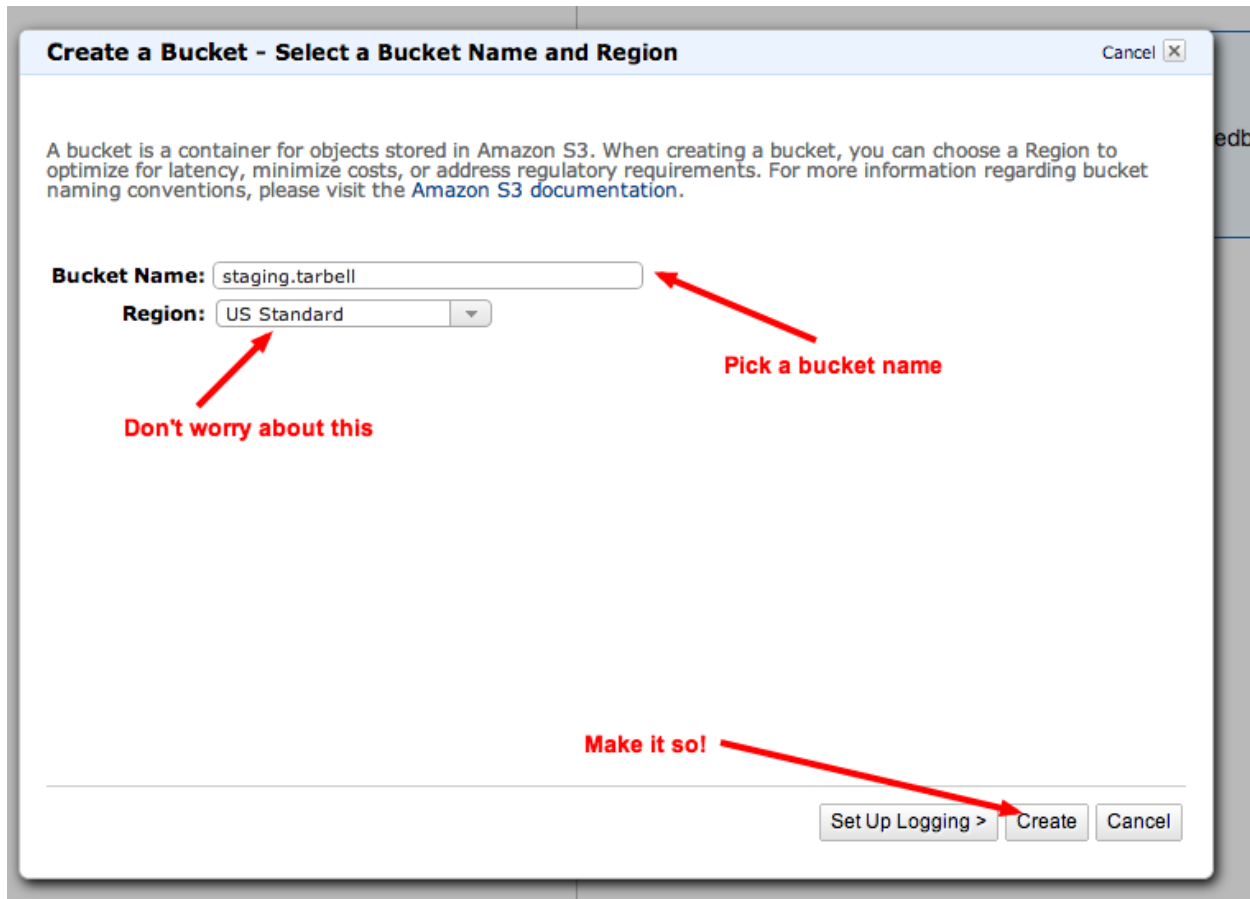
Once you've added production and staging buckets to your configuration, you will get this message:

```
Would you like to add bucket credentials? [y/N]
```

If there are additional buckets in your S3 account that you want to use with Tarbell, enter their names here. Otherwise, skip this.

3.1.5 Set a default project path

This is where your Tarbell projects will live. This path will be used by *tarbell list* and *tarbell switch*.

3.1.6 Using Tarbell with virtualenv

Note: If you've never heard of virtualenvs or know you're not using one with Tarbell, skip this section.

Virtual environments (virtualenv) are useful for developers and advanced users managing many Python packages. Tarbell can be installed globally or within a virtualenv.

If you'll be working on Tarbell itself, extending its functionality or otherwise manipulating the guts of the system, then it might make sense to install it inside a virtualenv.

Here are some things to keep in mind if you use a virtualenv:

- The Tarbell settings file (`~/.tarbell/settings.yaml`) is global, meaning all Tarbell projects - whether inside a virtualenv or not - share the same settings. This includes the path that Tarbell expects to find all your projects (i.e., where Tarbell will look when you run `tarbell list` and `tarbell switch`.)

- The `client_secrets.json` file used to authenticate to Google is also global, so you may run into problems using multiple Google accounts to access spreadsheets.

3.2 Tutorial

Let's build a website about celebrated Chicago journalist Ethel Payne!

You'll need to have access to a command prompt (an application on your computer that allows you to execute text-based commands). Search for the Terminal application. All of the commands we'll show you here will need to be typed into the command prompt.

First you need to install and configure Tarbell. Make sure to set up a Google spreadsheet. Go ahead. We'll wait.

3.2.1 Set up a new project

Once Tarbell is configured, create a new project by typing this command into your prompt:

```
tarbell newproject
```

You'll need to answer a few questions. What will its name be, where on your computer will it live? And a few others about its data sources and management. Anytme you see something in brackets, you are being prompted with a default answer, and can just hit enter if you wish to accept it. The process will go something like this:

```
tarbell newproject

What is the project's short directory name? (e.g. my_project) ethelpayne

Where would you like to create this project? [/Users/davideads/tarbell/ethelpayne]

What is the project's full title? (e.g. My awesome project) Ethel Payne: A life in journalism
```

You will be asked to choose which blueprint you wish to be the foundation of your project. Blueprints can be expanded upon but are very useful for setting the basic configurations necessary for the kinds of projects they have been designed for. See this page for more about blueprints, and go ahead and pick one:

```
Pick a Tarbell blueprint:

  [1] Basic Bootstrap 3 template
      https://github.com/newsapps/tarbell-template

[2] Searchable map template
      https://github.com/eads/tarbell-map-template
```

```
Which template would you like to use? [1] 1

Cloning into '_blueprint'...

Checking connectivity... done
```

At this point Tarbell will prompt you to make a Google spreadsheet, and if you didn't do this during the install and configure part of the tutorial, it's a good idea to do it now. Tarbell will look for the Google settings you'll need:

```
client_secrets found. Would you like to create a Google spreadsheet? [Y/n] y

What Google account should have access to this this spreadsheet? (Use a full email address, such as y

Success! View the spreadsheet at https://docs.google.com/spreadsheet/your_key_will_go_here
```

Now Tarbell installs your project's files and creates a git repo for your code:

```
Copying configuration file

- Creating tarbell.py project configuration file

- Done copying configuration file

Copying html files...
Copying index.html to /Users/davideads/tarbell/ethelpayne

Initial commit

[master (root-commit) 2bf96fb] Created ethelpayne from https://github.com/newsapps/tarbell-template
5 files changed, 58 insertions(+)
create mode 100644 .gitignore
create mode 100644 .gitmodules
create mode 160000 _blueprint
create mode 100644 index.html
create mode 100644 tarbell_config.py
```

At this point you might have the option to work with some project management tools, and you will definitely be prompted to use Github for sharing your code:

```
-- Calling newproject hooks --
--- Calling create_repo
Want to create a Github repo for this project [Y/n]? n
Not creating Github repo...

All done! To preview your new project, type:

tarbell switch ethelpayne

or

cd /Users/davideads/tarbell/ethelpayne
tarbell serve
```

And if all has gone well? You will see this message:

```
You got this!
```

Well, you heard the machine, you got this. Run the switch command to fire up a preview server:

```
tarbell switch ethelpayne
```

```
Switching to ethelpayne
Edit this project's templates at /Users/davideads/tarbell/ethelpayne
Running preview server...

Press ctrl-c to stop the server
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

Now visit http://127.0.0.1:5000/ in a browser.

You can also run your project by changing to the proejct directory and running the command:

```
tarbell serve.
```

You're ready to start editing your template.

3.2.2 Structure your project

It's a good idea to organize your project's files according to convention. That way everyone knows where to find things. Make directories named css, data, images and js in your project root directory (i.e., not in _blueprint) to keep your project uncluttered. See the section for creating and installing projects for more detail about best practices when creating your projects.

3.2.3 Add content

In a browser, open the Google spreadsheet that you created during the project set up. This is where our website's content will live. Let's look at the values worksheet (visible in the tabs in the bottom left) first. You should see something like this:

	A	B	C
1	*key*	value	_notes
2			— Optional variables —
3	google_analytics_id		Identifier for Google analytics
4	opengraph_image		OpenGraph image (Facebook, Pinterest, Google Plus), use full url
5	opengraph_descriptio		OpenGraph description (Facebook, Pinterest, Google Plus)
6	twitter_description		Text for twitter
7			— Example variables —
8	headline	Test headline	Show key->value mapping – print in template with {{ headline }}
9			
10			
11			

Keys and values are a common idea in programming: each key is shorthand for a corresponding value. Each of the values in the *values* column is available to your site when you use the matching *key* in your template.

Note: Header fields that start with underscores, like *_notes* does here, will not be made available to your template.

Open your project's index.html page and, assuming you chose the Basic Boostrap template (option 1), you should be able to find this line

```
<h1>{{ headline }}</h1>
```

Note: To start creating pages, you'll need a text editor. (TextWrangler is a decent option.)

Look at your page in the browser again and notice the headline matches what's in your Google spreadsheet under the *value* column with the *key* "headline". Try changing that value in the spreadsheet to "Ethel Payne, Chicago journalist".

Reload the server at http://127.0.0.1:5000 in your web browser to see your changes!

You can add as many keys and values as you like. We'll add a few.

8	*headline*	Ethel Payne, Chicago journalist
9	*quote*	I stick to my firm, unshakeable belief that the black press is an advocacy press, and that I, as a part of that press, can't afford the luxury of being unbiased ... when it come to issues that really affect my people, and I plead guilty, because I think that I am an instrument of change.
10	*quote_source*	Ethel Payne: A life in journalism

Now we need to reference these variables in the template. Go back to index.html and add

```
<blockquote>{{ quote }}</blockquote>
<p>from {{ quote_source }}</p>
```

Reload your site and look at the results!

Note: Tarbell uses Jinja2 for templating. Read the excellent documentation to learn more about using Jinja.

3.2.4 Displaying data

Sometimes you need to display structured data. Helpfully, the Google spreadsheet you created has some data like this under the *data* worksheet. The best way to display this data in Tarbell is by using a for loop (using Jinja2 syntax)

```
{% for row in data %}
  <p>
    <strong>{{ row.column1 }}:</strong>
    {{ row.column2 }}
  </p>
{% endfor %}
```

You should see the following when you reload your page:

row1, column1: row1, column2 **row2, column1**: row2, column2

Let's take a closer look at what's going on here:

```
{% for row in data %}
```

This reads in every row in the *data* worksheet. If we called our worksheet "birthdates," we could access that data by doing:

```
{% for row in birthdates %}
```

You'll notice that we no longer have columns labeled "key" and "value." Instead, we access the column we want by name. To understand this better, let's add some data about some famous ladies who might have been friends of Ida Tarbell had they known one another:

	A	B	C	D	E
1	column1	column2	name	born	died
2	row1, column1	row1, column2	Grace Hopper	12/9/1906	1/1/1992
3	row2, column1	row2, column2	Ethel Payne	8/14/1911	5/28/1991
4					

Now let's edit our index.html again to display this information:

```
{% for row in data %}
  <h2>{{ row.name }}</h2>
  <strong>{{ row.born|format_date }} – {{ row.died|format_date }}</strong>
  <p>{{ row.name }} was known for her work in {{ row.known_for }}.</p>
{% endfor %}
```

Your page should now look like this:

Grace Hopper

Dec. 09, 1906 - Jan. 01, 1992

Grace Hopper was known for her work in mathematics and computer programming.

Ethel Payne

Aug. 14, 1911 - May. 28, 1991

Ethel Payne was known for her work in civil rights journalism.

3.2.5 Adding CSS

Out of the box, Tarbell gives you Bootstrap 3 CSS. Chances are, you'll want to extend this to add your own CSS to your project.

To this point, we've ignored the _blueprint directory in your project. Now's the time to dive in! You may have noticed this line up at the top of your index.html file:

```
{% extends "_base.html" %}
```

The _base.html file is where all of the CSS, JavaScript and other goodies live. By "extending" _base.html, index.html has access to all of the things that live in the base. You can read more about how template inheritance works here.

Note: Filenames prefaced with an underscore will be ignored when you publish your project. Our naming convention is to use underscores for "partial" templates that represent small pieces of the page, like navigation and footers.

There are two CSS blocks at the top of the page:

```
{% block library_css %}
<link rel="stylesheet" type="text/css" href="http://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/
<link rel="stylesheet" type="text/css" href="css/base.css" />
{% endblock library_css %}
```

```
{% block css %}{% endblock %}
```

The first block includes Bootstrap 3's CSS and your project's default base.css stylesheet. Don't worry about it right now. The second block is what you'll want to extend.

Note: You'll only need to touch the library_css block if you need to do something like override the version of Bootstrap included here. Otherwise, for adding project-wide styles, edit the base.css file.

In your project root (i.e., not in _blueprint), create a css folder, if you haven't done so already. Inside that, create a new style.css file and add some CSS rules:

```
h2 { font-family: Georgia, serif; }
strong { color: #c7254e; }
```

Now switch back over to your index.html and add the css block. Do this on line 2, after the base extension:

```
{% extends "_base.html" %}

{% block css %}
<link rel="stylesheet" href="css/style.css">
{% endblock %}

{% block content %}
```

Your text should now be styled!

3.2.6 Using Javascript

You can include JavaScript on your page much the way you would include CSS. By default, these are the blocks available in _base.html:

```
{% block library_scripts %}
<script src="http://cdnjs.cloudflare.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.1.1/js/bootstrap.min.js"></scr
{% endblock library_scripts %}

{% block scripts %}{% endblock %}
```

The *library_scripts* block contains the default Bootstrap Javascript and jQuery. You probably don't need to mess with this.

The *scripts* block can be extended in your templates. You'll want to create a *js* directory in your project root to hold all of your Javascript files. Then you can include them in your index.html (or other templates):

```
{% block scripts %}
<script src="js/app.js"></script>
{% endblock %}
```

3.2.7 Using `{{ super() }}`

Sometimes, you want to extend a CSS or Javascript block without overriding the stuff that's in the base. You can do that with the super() template tag. This will look at all of the things in the base version of the block, and add your new content to it rather than override it. For instance:

```
{% block library_scripts %}
{{ super() }}
<script src="js/app.js"></script>
{% endblock library_scripts %}
```

This will yield this on the rendered page:

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.1.1/js/bootstrap.min.js"></scr
<script src="js/app.js"></script>
```

Without `super()`, you would merely end up with:

```
<script type="text/javascript" src="js/app.js"></script>
```

3.2.8 Overriding default templates

While the Tarbell `base.html` template (see more on templates) contains some very handy things, you may find you need to override some of the provided templates. One of the most common cases in which this occurs is the navigation.

In the `_blueprint/base.html` template, you can see that the nav is included just before the content starts:

```
{% block nav %}
  {% include "_nav.html" %}
{% endblock nav %}

{% block content %}{% endblock content %}
```

To override the default nav, create a new `_nav.html` file in your project root (at the same level as `index.html`, not within the `_blueprint` directory). Type in a message to yourself:

```
Ida Tarbell would be proud of this website!
```

Reload your test page. Bingo!

Now, such a message probably isn't very helpful to your users, so to create a more functional nav, copy the code out of `_blueprint/nav.html`, paste it into `_nav.html`, and rejigger the code as desired. It's all Bootstrap 3, so you might find it helpful to view the Bootstrap navbar docs.

3.2.9 Putting it all together: Leaflet maps

With Tarbell, you can use a Google spreadsheet to provide any kind of data to your page: text, image URLs, and even latitude/longitude data that can power a map. We're going to show you how to use Tarbell to store geodata for a map.

Set up the spreadsheet

First, we need to create that Google spreadsheet to power the map. Go to the spreadsheet you created when you started your project, and edit the `data` tab to contain columns named `city`, `latitude` and `longitude`. Then, select the visible cells with your mouse, and choose Format -> Number -> Plain text. (This will prevent Google from automatically converting your lat/longs to dates.) Enter the following data:

Chicago 41.838299 -87.706953 Detroit 42.3314 -83.0458 Minneapolis 44.9833 -93.2667

It should look like this:

	A	B	C	D
1	city	latitude	longitude	
2	Chicago	41.838299	-87.706953	
3	Detroit	42.3314	-83.0458	
4	Minneapolis	44.9833	-93.2667	
5				

values ▾ data ▾ keyed_data ▾

Make a holder for the map

Now, we're going to create what's called a "partial." This is an HTML file which gets parsed and added to another HTML file when Tarbell compiles your pages. (`_nav.html` is a partial, as is `_footer.html`.) We'll call it `_map.html`. Open it up and add a div that will contain your map:

```
<div id="map"></div>
```

Note: Partials are always prefaced with an underscore _. This tells Tarbell to refrain from compiling them as independent pages. Otherwise, your project would end up with pages like *yoursite.com/footer.html*. Anything you write in a partial could also be written directly on a page, but using a partial makes it easier to reuse code. For instance, we may want to use our map on every page on our site, but using a partial means we only store the code in one file, making it easy to update and maintain.

Give the map a height

We'll need to set a height and width for this map in the `style.css` CSS file created earlier with the following rule:

```
#map { height: 600px; width: 600px; }
```

Include map assets on the index page

On the `index.html` page, include the partial like so::

```
{% include "_map.html" %}
```

We'll need to set a height for this map in the CSS file created earlier called style.css with the following rule:

```
#map { height: 180px; }
```

Include the Leaflet CSS and your new stylesheet (if you haven't already) before the content block:

```
{% block css %}
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
<link rel="stylesheet" href="css/style.css" />
{% endblock %}
```

Then, after the content block, add the Leaflet Javascript library and a new file you will create:

```
{% block scripts %}
<script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js"></script>
<script type="text/javascript" src="js/map.js"></script>
{% endblock %}
```

You'll need to create a `js` directory in your project root. Within that, create a file named `map.js` (note that we included that in the block above). This is where we'll write a little bit of Javascript to make the map work.

Write some Javascript

Note: Your Javascript should be enclosed in a `$(document).ready(function(){` call, which ensures that everything enclosed within will be loaded after the page has loaded. For more information, see this page.

Now that everything else is set up, open your `map.js` file. First, let's access the data you put in your spreadsheet and convert it to JSON in one fell swoop with a very handy Jinja filter:

```
var map_data = {{ data|tojson }}
```

This will turn the columns from the workbook called `data` into something that looks like this:

```
.. image:: img/map_data.png
```

We can reference our city data in the rest of our Javascript now. So let's make the map!

When we include `leaflet.js` on the page, it will create a Javascript object named `L` that allows us to access all the Leaflet functionality we need. We need to store that object in a variable that references the div in `_map.html` with the ID `map`, which will contain out map. Note that we refernce the div ID by wrapping the ID name, `map`, in quotes. We'll set the latitude and longitude to that of the first city (from the `map_data` variable), and then the zoom level of the tile (the lower the number, the farther out the map will be zoomed to start):

```
var map = L.map('map').setView([41.838299, -87.706953], 6);
```

This tells Leaflet to create a map and set the center of it to Chicago, with a default zoom level of 6.

Note: Leaflet map objects give you a great deal of control over your map's appearance and behavior. The most basic settings are made via the `setView` method, which controls latitude, longitude and zoom levels. Leaflet exposes many methods and properties to manage the state of your map, though, and we definitely encourage you to check out their docs and continue experimenting at the end of this tutorial.

Next we'll give Leaflet the URL of a tileset to `addTo` the map object we created. We will also set the max and min zoom levels for the tiles. We'll use Open Street Map's tileset:

```
L.tileLayer(
  'http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    attribution: 'Map data &copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors',
    maxZoom: 16,
    minZoom: 5
}).addTo(map);
```

Note: For more information about what this code does, see our docs on creating Leaflet maps

---
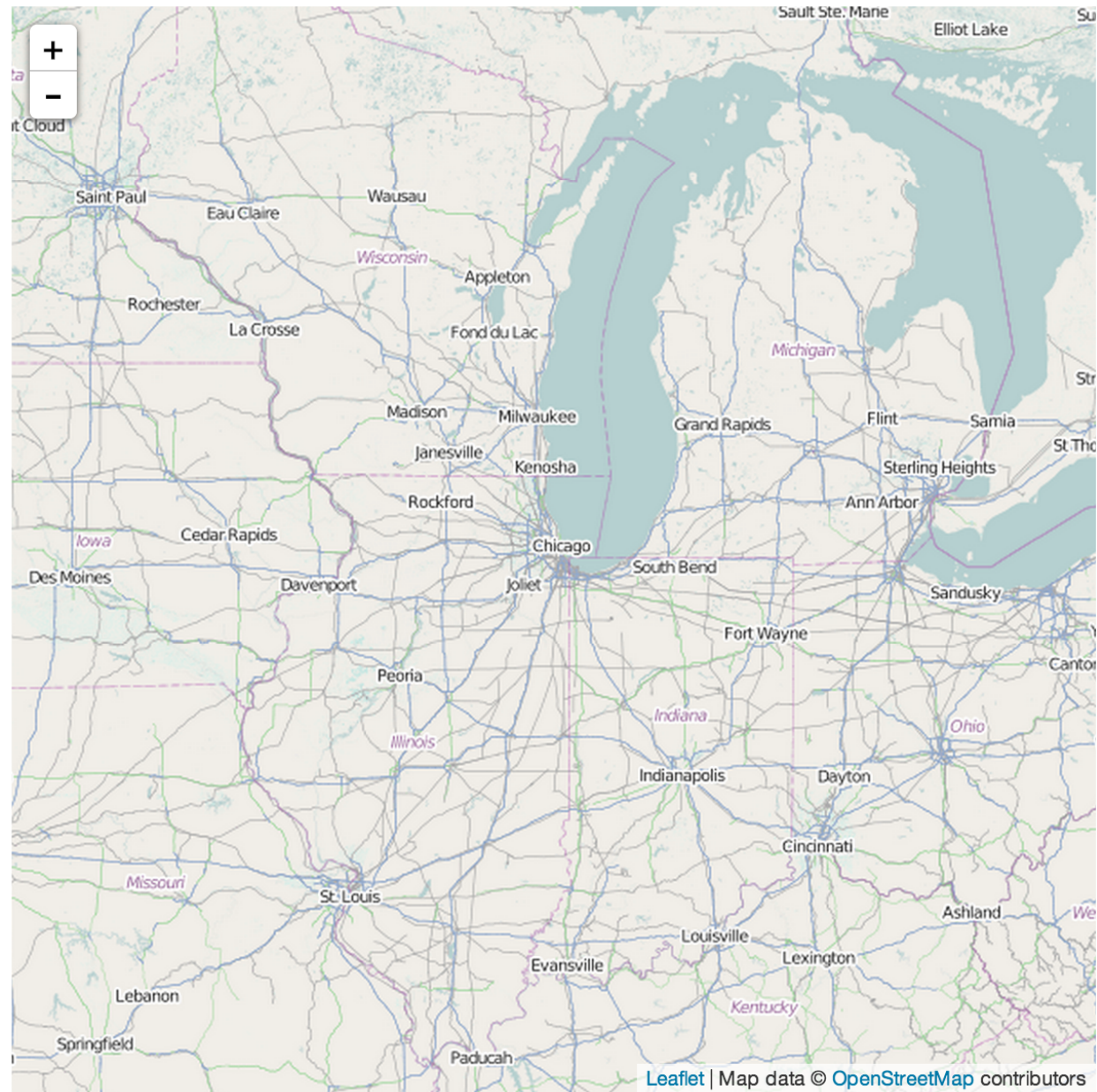
So now we have a map, but it would be really helpful to display some information on it. Let's add a marker for Chicago, by adding the lat/lon from the spreadsheet and then attaching it to the map:

```
var chicagoMarker = L.marker([map_data[0].latitude, map_data[0].longitude]);
chicagoMarker.addTo(map);
```
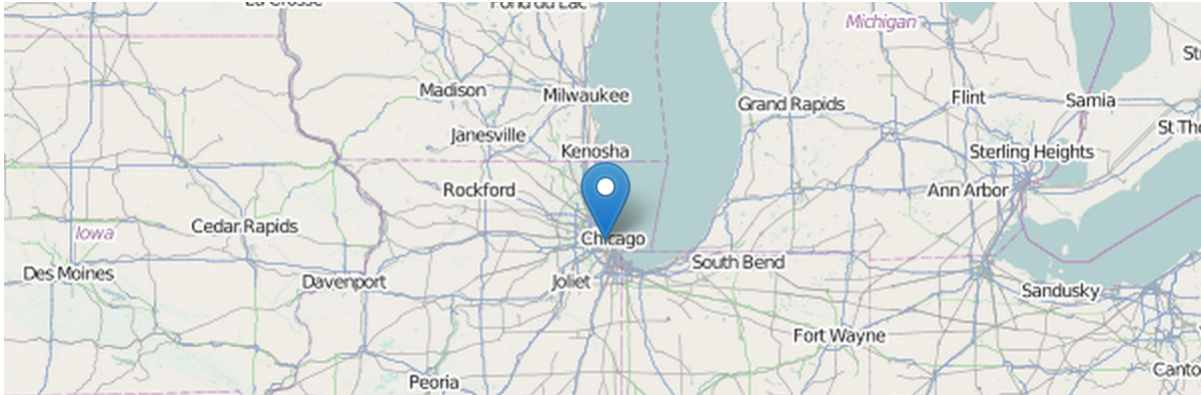
Bingo, there's Chicago! Now, suppose we want to display a little information when you click on the city:

```
chicagoMarker.bindPopup('<h3>This is the city of ' + map_data[0].city + '</h3>');
```

Now, when you click on Chicago, the popup should show the name of the city.
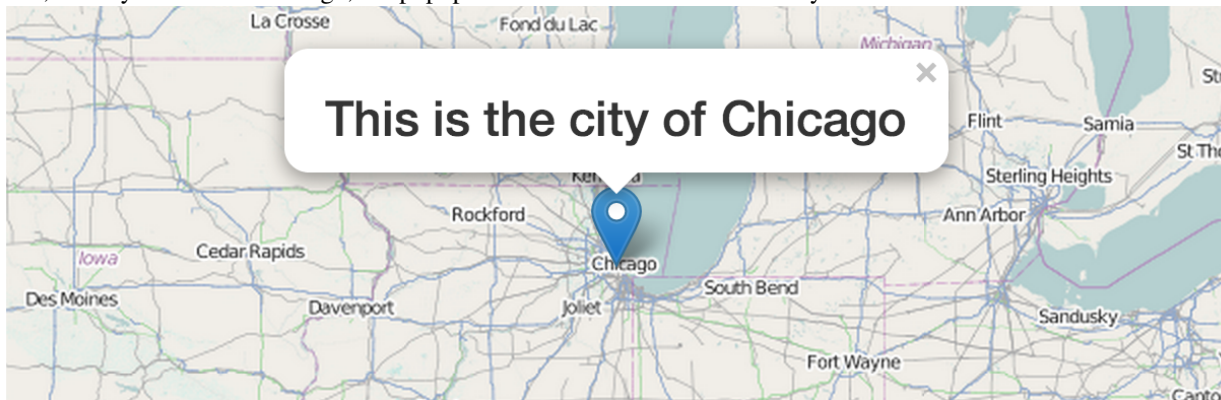
Remember that we assigned the contents of your spreadsheet to the variable `map_data`. Now we can access the first element in the JSON data by using the syntax `[0]`. To grab the second city, we'd use `map_data[1]`, because in this instance, our counting is zero-indexed. We can tell Javascript which column header we want to reference with the syntax `.columname`. So `map_data[0].latitude` translates to, "Give me the latitude column for the first city in the data."

You can see how we could easily create markers for the other two cities:

```
var detroitMarker = L.marker([map_data[0].latitude, map_data[1].longitude]);
detroitMarker.bindPopup('<h3>This is the city of ' + map_data[1].city + '</h3>');
detroitMarker.addTo(map);

var minneapolisMarker = L.marker([map_data[2].latitude, map_data[2].longitude]);
minneapolisMarker.bindPopup('<h3>This is the city of ' + map_data[2].city + '</h3>');
minneapolisMarker.addTo(map);
```
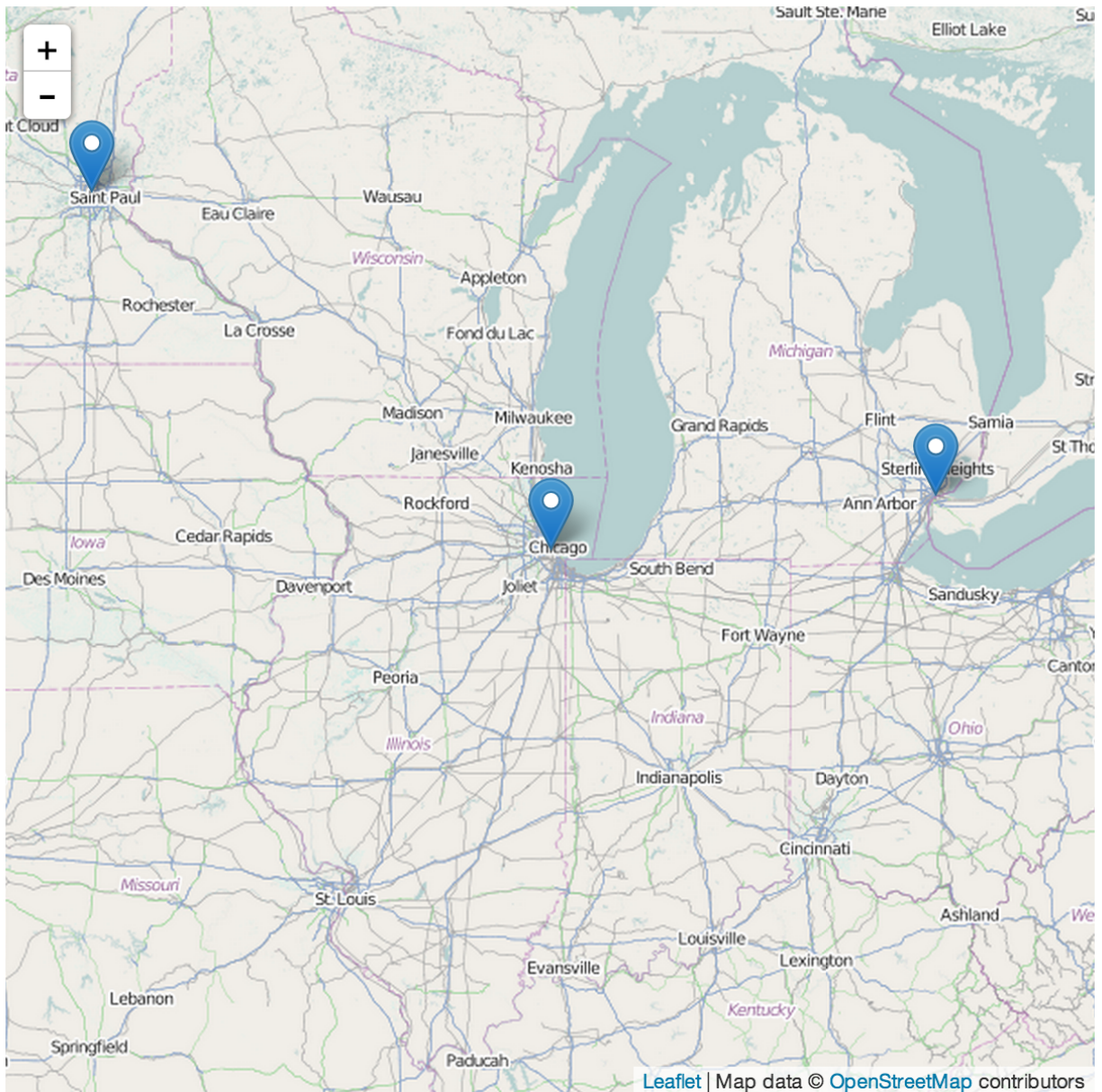
Yay! But wait...what if we have *a lot* of other cities? This is going to take forever. There is a better way! Replace all the city marker code with this:

```
for (i=0; i < map_data.length; i++){
    var marker = L.marker([map_data[i].latitude, map_data[i].longitude]);
    marker.bindPopup('<h3>This is the city of ' + map_data[i].city + '</h3>');
    marker.addTo(map);
}
```

Now you're cooking with gas! This is a standard Javascript for loop that creates a counter, `i`. Note that instead of using numbers with `map_data`, we're now using `i` instead. Each time the loop runs, as long as `i` is less than the number of items in the `map_data` array, `i` will be incremented. So on the first pass, `i` will equal 0, and will pull in the information for Chicago. On the second pass, it will equal 1 and pull in the information for Detroit. Finally, it will equal 2, and will grab Minneapolis' information before it quits. This will work for a spreadsheet of 3 cities or 300.

To delve deeper into what makes a Leaflet map tick, we recommend reading the Tribune Leaflet docs.

Happy Tarbelling!

3.3 Upgrading

If you've already installed Tarbell, you can upgrade with:

```
pip install -U tarbell
```

Note: In version 0.9-beta6, some naming conventions changed. The _base folder and base.py file are now called _blueprint and blueprint.py. Future versions in the 0.9.x and 1.0.x branches will maintain backwards compatibility with Tarbell projects created using the old naming convention.

3.4 Create and install projects

Note: Tarbell requires Git to install project and blueprints. does not support interactive Git sessions. If Git attempts to open an interactive prompt when accessing a private repository via HTTPS, Tarbell will exit with a warning.

3.4.1 Create a new project with *tarbell newproject <projectname>*

Run

```
tarbell newproject
```

You'll be asked a few questions, such as which path you'd like to create the project on, whether you want to use Google spreadsheets, and whether you want to instantiate a git repo. (See the tutorial for more details.)

When you're done, run a preview server

```
tarbell switch myprojectname
```

You can now open up */path/to/myprojectname* and start editing the "index.html" file.

3.4.2 Install an existing project with *tarbell install <repository-url>*

Note: The project must include a tarbell_config.py file and be able to be cloned with Git. If the project uses Google spreadsheets, your Google account must be able to access the spreadsheet in question.

Run

```
tarbell install https://urltorepository.com/projectname
```

e.g.

```
tarbell install https://github.com/sc3/crime-punishment
```

3.5 Building projects

3.5.1 Editing templates

Tarbell projects consist of simple HTML pages that may use Jinja2 templating features.

If you create a file in your project directory called `chapter1.html`, you'll be able to preview the file at http://localhost:5000/chapter1.html and publish to the same file. This file can be straight up HTML, or it can inherit from a Tarbell blueprint file.

Files and directories that start with an underscore (_) or a dot (.), like the `_blueprint` directory containing the Tarbell blueprint files, will not be rendered by the preview server or included in the generated static HTML.

3.5.2 Understanding Tarbell Blueprints

Blueprints are exactly what they sound like — a basic structure for building projects upon. From the Flask documentation:

> Flask uses a concept of blueprints for making application components and supporting common patterns. Blueprints can greatly simplify how large applications work, but a blueprint is not actually an application. Rather it is a blueprint of how to construct or extend an application.

Tarbell ships with a default blueprint called _blueprint. This folder contains boilerplate code like advertising, analytics, and common page elements. Tarbell projects should inherit from blueprints.

Here's a simple `_blueprint/_blueprint.html` example.

```html
<html>
  <head>
    <title>{{ title }}</title>
    {% block css %}
    <link rel="stylesheet" type="text/css" href="css/style.css" />
    {% endblock css %}
  </head>
  <body>
    {% block content %}{% endblock content %}
  </body>
</html>
```

To inherit from this template, you use the "extend" syntax in `index.html` or other project files you create. All your `index.html` needs to contain is:

```html
{% extends '_base.html' %}

{% block content %}
<h1>{{ title }} </h1>
{{ content|markdown }}
{% endblock content %}
```

You might notice we're using the `|markdown` filter. Blueprint templates also define filters, enabled by Jinja2. See building blueprint templates for more, and the Jinja2 docs for more on Jinja2.

If a blueprint defines a static file or template (e.g. `_blueprint.css`), it will be available relative to the project's base path (e.g. http://127.0.0.1:5000/style.css). If a project defines a file with the same name, the project's version will be used instead.

See the basic Tarbell template for a simple implementation of a Blueprint.

3.5.3 Template inheritance: Override files from Tarbell Blueprints by copying to your project directory

Any file in a Tarbell Blueprint can be overridden in your project files.

For example, if your blueprint includes a file `_blueprint/_nav.html`, you can create a file named `_nav.html` in your project directory and it will be published instead of the blueprint version.

This works for all files, static or templates.

3.5.4 Files prefixed with underscores (_) will not be generated or published

To suppress a file from publishing, use a filename like `_filename.txt`.

3.5.5 Configuring projects

Project configuration is kept in the *tarbell_config.py* file in your project's blueprint directory. See *Project settings (tarbell_config.py)* for configuration documentation.

3.5.6 Creating JSON

You can publish the data coming from your Google spreadsheet as JSON if so desired. To do this, set the *CREATE_JSON* flag in *tarbell_config.py* to *True*. When you visit *yoursite.com/data.json*, Tarbell will create some JSON that will look something like this:

```
{
  name: "ethelpayne",
  title: "Ethel Payne: A life in journalism",
  headline: "Ethel Payne, Chicago journalist",
  quote: "I stick to my firm, unshakeable belief that the black press is an advocacy press, and that
  data: [
    {
      name: "Ethel Payne",
      known_for: "civil rights journalism",
      born: "8/14/1911",
      died: 33386
    }
    {
      name: "Grace Hopper",
      known_for: "mathematics and computer programming",
      born: "12/9/1906",
      died: 33604
    },
  ]
}
```

The first block of keys and values comes from the *values* workbook. The *data* array represents another workbook. Any other workbooks you create within your spreadsheet will be represented as separate arrays.

Optionally, you can use the *CONTEXT_SOURCE_FILE* setting in *tarbell_config.py* to determine your data source, which can be a URL, local file, CSV or Excel file.

Note: The `data.json` file is created on the fly and will not appear in your project root. You can view and access it locally at `127.0.0.1:5000/data.json`. If JSON creation is enabled, it will override any local file named `data.json`.

3.5.7 Using context variables

Template data can come from Google spreadsheets, a local or remote CSV or Excel file, or tarbell_config.py's DEFAULT_CONTEXT. The context source is configured in tarbell_config.py (see *Project settings (tarbell_config.py)* for reference).

This simple DEFAULT_CONTEXT shows many of the key template features:

```
DEFAULT_CONTEXT = {
    'name': 'nellie-bly',
    'title': 'The Story of Nellie Bly',
    'font_size': '20px',
    # Nested dictionary
    'photos': {
        'intro': {
            'url': 'img/bly01.jpg',
            'caption': 'A caption',
        }
    },
    # Nested list
    'timeline': [
        {'year': '1902', 'description': 'Description...'},
        {'year': '1907', 'description': 'Description...'},
        {'year': '1909', 'description': 'Description...'},
    ],
  }
}
```

To print the title in your template, use *{{ title }}*:

```html
<h1>{{ title }}</h1>
```

Address a nested dictionary:

```html
<img src="{{ photos.intro.url }}" alt="{{ photos.intro.caption }}" />
<aside>{{ photos.intro.caption }}</aside>
```

Access a list of data:

```html
<ul>
  {% for year in timeline %}
  <li><strong>{{ year }}</strong>: {{ description }}</li>
  {% endfor %}
</ul>
```

3.5.8 Where can context variables be used?

Context variables can be used in HTML, CSS, and Javascript files. If the text file causes a Jinja template error (which can happen if the file has Jinja-like markers), the file will be served as static and the preview server will log an error.

This means that CSS and Javascript files may include variables. style.css might include:

```css
#content { font-size: {{ font_size }}; }
```

Similarly, a Javascript file could include:

```
var data = {{ photos|tojson|safe }}
console.log(photos.intro.url);
```

Use this feature with care! Missing variables could easily break your CSS or Javascript.

3.5.9 Anatomy of a project directory

When you run `tarbell newproject` with the default blueprint, a number of new files and folders are created, many of them with special significance. Details may vary for other blueprints, but they're likely to have many similar files and concepts.

Here's a rundown of what they all do.

Files in the root directory:

index.html The first page people will see when they visit your project. This is typically where most of the content lives, and is probably where you want to start editing.

tarbell_config.py The settings and context for this specific project, described in more detail in the *Configuring projects section above*.

Files in the _blueprint directory:

Keep in mind that you rarely want to edit the blueprint files in the `_blueprint/` directory - if you want to change something, copy the file to the root directory and make the change there. If a file of the same name exists in both the root directory and the `_blueprint/` directory, Tarbell will rely on the one in the root directory.

The only time you should edit the files in the `_blueprint/` directory is when you'd like to create or update the blueprint itself.

_base.html The base page template, containing `<head>` and `<body>` tags, and pointing to many of the Javascript and CSS files that will be loaded for each page in the project.

_footer.html The partial template containing anything you'd like to appear consistently in the footer at the bottom of each page.

_nav.html The partial template containing the nav bar that runs along the top of the page.

_spreadsheet.xlsx This is the template file that Google spreadsheets will be based upon. Unlike most other files in `_blueprint`, overriding it in your root directory won't do anything. However, if you want future projects to be created with a different spreadsheet template, you can edit this file and commit it to a repository you control; learn more about this feature in the Developing blueprints section.

base.css The base CSS file imported by the blueprint for this project. Override this file in your root directory if you'd like to make CSS changes.

blueprint.py A Python file that contains a default set of template filters for use in this project. Override this file in your root directory if you'd like to alter the behavior of these filters, or add your own. If you'd like to make your changes available to other projects, check out the Developing blueprints section.

favicon.ico Favicons are small logos for websites that typically appear next to a website's name in a browser tab. Change this file in order to change the logo associated with your site in users' browser tabs, though keep in mind that favicons have a number of rules about how they should be constructed.

favicon-preview.ico This is the icon for the in-development version of a site that appears next to the website's name in a browser tab, following the same rules as for favicon.ico above. The key difference is that this icon is meant to remind developers and testers that they're not looking at a live site.

index.html This is a fallback version of the project's front page, in case the `index.html` file in the root directory is removed or renamed. It begins life as an exact copy of the root directory's `index.html`.

3.5.10 Adding custom routes

Sometimes, you'll find that you need create pages programatically, instead of simply adding template files. Or you may need to output data in a format other than HTML (like JSON, for example).

For example, here's a hook from a project's *tarbell_config.py* that publishes special social media stub pages for each row in a worksheet. This allows individual items to be shared from a single-page listicle app:

```python
from itertools import ifilter
from flask import Blueprint, render_template
from tarbell.hooks import register_hook

# create a blueprint for this project
# tarbell will consume this when the project loads
blueprint = Blueprint('myproject', __name__)

@blueprint.route('/rows/<id>.html')
def social_stub(id):
    "Build a social stub for in-page permalinks"
    site = g.current_site

    # get our production bucket for URL building
    bucket = site.project.S3_BUCKETS.get('production', '')
    data = site.get_context()
    rows = data.get('list_items', [])

    # get the row we want, defaulting to an empty dictionary
    row = next(ifilter(lambda r: r['id'] == id, rows), {})

    # render a template, using the same template environment as everywhere else
    return render_template('_fb_template.html', bucket=bucket, **row)
```

Here's the *_fb_template* referenced above:

```html
<html>

<head>
  <script>
    document.location = "http://{{ bucket }}/#{{ row.id }}";
  </script>

  <meta property="og:url" content="http://{{ bucket }}/rows/{{ row.id }}.html" />
  <meta property="og:title" content="Great moments in history: {{ row.heading }}" />
  <meta property="og:description" content="{{ row.og }}" />
  <meta property="og:image" content="http://{{ bucket }}/img/{{ row.img }}" />
</head>

<body></body>

</html>
```

Since this is a custom route, we need to tell Tarbell to build it as an HTML file when we call *tarbell generate* or *tarbell publish*. There are two ways to do this: *url_for* tags, or URL generators.

Note: Under the hood, Tarbell uses Frozen-Flask to generate static pages, so you can consult that project's documentation for more details and further customization.

Auto-linking:

In your main *index.html* template, generate a link for each stub:

```
{% for row in list_items %}
<a href="{{ url_for('myproject.social_stub', id=id) }}">Stub</a>
{% endfor %}
```

Frozen-Flask will automatically track every call to *url_for* and build out those URLs. If that doesn't make sense for your project, you can also write a generator function, and use a Tarbell hook to register it at build-time.

```
# in tarbell_config.py

def social_stub_urls():
    "Generate a URL for every social stub"
    site = g.current_site
    data = site.get_context()
    rows = data.get('list_items', [])

    for row in rows:
        yield ('myproject.social_stub', row['id'])

@register_hook('generate')
def register_social_stubs(site, output_root, extra_context):
    "This runs before tarbell builds the static site"
    site.freezer.register_generator(social_stub_urls)
```

3.5.11 Using Flask Extensions

The Flask ecosystem includes all sorts of useful extensions for building web applications.

Every Tarbell site includes a Flask app that handles request routing and template rendering. You can hook into this underlying app to take advantage of Flask extensions to speed up your development process.

1. Define a Tarbell Blueprint in your *tarbell_config.py* file. (Your variable name must be *blueprint* for Tarbell to find it.)

2. Use the blueprint.record decorator to tell Flask to run a function when the blueprint is loaded onto an app. This will happen at the end of your Tarbell site's *__init__* method. The function will be passed a *state* object, with a reference to your Flask app at *state.app*.

3. Create an instance of the extension you're using. Inside the function you decorated with *blueprint.record*, run the extension's *init_app* method with your site's Flask app. (You can also initialize the extension with the app in one step, if you don't need a reference to the extension outside that function.)

4. Add any configuration settings the extension needs to *state.app.config*.

Here's how to use flask-thumbnails with Tarbell:

```
# tarbell_config.py

from flask import Blueprint
from flask.ext.thumbnails import Thumbnail

# initialize a blueprint and thumbnails extension
blueprint = Blueprint('project', __name__)
thumbnails = Thumbnail()

# media settings, note that these are relative paths
MEDIA_FOLDER = "img/uploads"
MEDIA_THUMBNAIL_FOLDER = "img/thumbnails"
```

```
# this function will run when Tarbell's underlying Flask app
# adds this blueprint
@blueprint.record
def app_setup(state):
    "Configure thumbnails"

    # configure thumbnails with the active app
    state.app.config['MEDIA_FOLDER'] = MEDIA_FOLDER
    state.app.config['MEDIA_THUMBNAIL_URL'] = MEDIA_FOLDER
    thumbnails.init_app(state.app)
```

Now, in your templates, you can use the *thumbnail* filter:

```
<img src="{{ 'image.jpg'|thumbnail('200x200') }}" alt="A cropped image">
<img src="{{ 'image.jpg'|thumbnail('200x200', crop='fit', quality=100) }}" alt="A cropped image">
```

3.6 Using Google spreadsheets

3.6.1 The *values* worksheet

The values worksheet must have "key" and "value" columns. These key-value pairs will be provided as global variables to templates. So if there's a row with a key column value of "foo" and a value of "bar", *{{ foo }}* in a template will print bar.

Take this sample worksheet:

key	value
title	Project title
intro	Project intro

A *values* worksheet that contains this data provides the *{{ title }}* and *{{ intro }}* variables to the template.

Use them in your templates:

```
<h2>{{ title }}</h2>
<p class="intro">{{ intro }}</p>
```

3.6.2 Named worksheets

Other worksheets can hold any kind of data supported by Google spreadsheets. These variables can be accessed by their worksheet name.

If there is no *key* column in the worksheet, the worksheet can be accessed as a list. Imagine a spreadsheet named *cars* with these values:

model	mpg
Civic	25.9
Accord	28.1
Element	24.6

You can access these variables in your spreadsheet with a loop:

```
{% for car in cars %}
  <h3>{{ car.model }}</h3>
  <p>MPG: {{ car.mpg }}</p>
{% endfor %}
```

If a column named *key* does exist, elements may be accessed by key. Imagine a spreadsheet named *companies* with these values:

key	name	country
ford	Ford	U.S.A.
honda	Honda	Japan
volvo	Volvo	Sweden

You can access these variables by their key name:

```
<p>{% manufacturers.ford.name %} is from {% manufacturers.ford.country %}</p>
```

3.6.3 Worksheet, column, and key names are slugified

Spaces and dashes are replaced with underscores (_). Non alphanumeric characters are removed. Case is preserved.

Examples of names that will be transformed:

- *My Worksheet* becomes *My_Worksheet*
- *My key\n* becomes *My_key*
- *my-Column* becomes *my_Column*

Names that will not be transformed:

- *MyColumn* remains *MyColumn*
- *mycolumn* remains *mycolumn*
- *my_column* remains *my_column*

3.6.4 Worksheets, columns, and keys names preceded by _ (underscore) are ignored

Precede any worksheet name, column name, or key with an underscore to hide it from your templates and JSON data.

3.6.5 Merged cells raise an error

Because merged cells exhibit unpredictable behavior, if merged cells are encountered while Tarbell is processing a spreadsheet, an error is raised containing the merged ranges.

3.7 Publishing

3.7.1 Generate static sites with `tarbell generate`

Generate HTML in a temporary directory:

```
tarbell generate
```

Generate HTML in a specific directory:

```
tarbell generate ~/output/myproject
```

Note: `tarbell generate` can be used to manually publish sites to hosts other than Amazon. Write a simple deployment script or use Fabric to call `tarbell generate <mydirectory>` and invoke a command to sync `<mydirectory>` with your host.

3.7.2 Publish projects with `tarbell publish <target>`

If Amazon S3 is configured, you can publish with:

```
tarbell publish <bucketname>
```

The default bucket is `staging`.

You can specify a bucket when publishing (defined in `tarbell_config.py`):

```
tarbell publish production
```

3.7.3 Configuring S3 buckets for a project

As touched on in the Configuring projects section, you can change the names and locations of your S3 buckets in the `tarbell_config.py` file for a given project. Often, projects have a `staging` version for testing, and a `production` version for the final product. However, these names are entirely arbitrary, so you can pick anything you like.

As an example, let's say you have 3 versions of your site: one for testing, one for the live site, and one that you'd like to preserve as an archive. They're all kept as subdomains of tribapps.com, and your staging site actually houses many different sites at various stages of development, so you want to publish to a specific directory.

This example assumes the same AWS ID and secret access key can be used to authenticate with each of the targets.

Create or update the `S3_BUCKETS` variable in `tarbell_config.py` as follows:

```
S3_BUCKETS = {
    "staging": "staging.tribapps.com/tarbell-staging",
    "production": "tarbell.tribapps.com",
    "archive": "archive.tribapps.com",
}
```

To push your site to http://staging.tribapps.com/tarbell-staging, run:

```
tarbell publish staging
```

To push your site to http://tarbell.tribapps.com, run:

```
tarbell publish production
```

You've probably already figured out the pattern, but to push your site to http://archive.tribapps.com, run:

```
tarbell publish archive
```

Note: The preferred format for S3 buckets is demonstrated above, without an `s3://` protocol indicator or trailing slash. However, `s3://foo.com/bar/` will work as well.

Handling buckets with differing credentials

What if `archive.tribapps.com` uses different Amazon S3 credentials?

To handle buckets with non-default credentials, run `tarbell configure` and configure a new bucket:

```
Please specify an additional bucket (e.g. additional.bucket.myorg.com/, leave blank to skip adding bu

Please specify an AWS Access Key ID for this bucket: [XXXXXXXXX] XXXXXXXXX

Please specify an AWS Secret Access Key for this bucket: [XXXXXXXXX] XXXXXXXXX
```

Or add some lines to `~/.tarbell/settings.yaml`:

```
# ...
s3_credentials:
  foo.bar.com:
    access_key_id: XXXXXXXX
    secret_access_key: XXXXXXXXXXXX
```

You can now publish to a bucket with non-default access credentials.

3.7.4 Tarbell does not delete files on S3

Because altering Amazon S3 buckets has some inherent dangers, Tarbell 1.0 does not include a delete feature. You can manually delete files on Amazon through the web interface or with a client like Cyberduck.

3.8 Developing Tarbell blueprints

Tarbell blueprints are starting points for your projects. They allow you and your collaborators to take the tedium out of creating similar projects over and over.

For example, your organization could use a Tarbell blueprint for all mapping projects that includes all the common libraries, your organization's branding, and default data to get started.

3.8.1 Basic blueprint ingredients

All Tarbell blueprints must include a file named `blueprint.py`. This file may be empty, but should include a variable called `NAME` when running `tarbell newproject`:

```
NAME = "My Tarbell blueprint"
```

Most Tarbell blueprints will want to define some standard files:

- `_base.html`: This can be named whatever you want. It is the Jinja base template the rest of your templates should extend with `{% extends '_base.html' %}`.
- `index.html`: A default page to use as a starting point for project development.
- `_spreadsheet.xlsx`: This Excel file is used to create the default Google spreadsheet your project will use.

(Optional) If your blueprint defines a `.gitignore` file, it will be copied to new projects when created.

3.8.2 Adding filters and functions

All Tarbell blueprints (and projects) can define *Flask blueprints <http://flask.pocoo.org/docs/0.10/blueprints/>* in order to add filters, context functions, and custom routes.

To enable this functionality, add to `blueprint.py`:

```python
from flask import Blueprint

NAME = "My Tarbell blueprint"
blueprint = Blueprint("base", __name__)
```

Note: The `"base"` argument above is an arbitrary, unique name for the Flask blueprint. It can be anything, but you should stick with "base" for most blueprints. Same with the `blueprint` variable name.

Now you can do anything a Flask blueprint can do, such as define a template filter. Here's a simple example:

```python
from flask import Blueprint
from jinja2 import Markup

NAME = "My Tarbell blueprint"
blueprint = Blueprint("base", __name__)

@blueprint.app_template_filter()
def embiggen(text):
    return Markup('<div class="embiggen">{0}</div>'.format(text))
```

Now you can use the `{{ myvariable|embiggen }}` in your templates to wrap the output of `myvariable` in a special `div` tag.

3.8.3 Implementing hooks

Tarbell blueprints can also implement Tarbell's hooks. For example, to print an cheery message after creating a new project:

```python
from tarbell.hooks import register_hook

NAME = "My Tarbell blueprint"

@register_hook("newproject")
def cheery_message(site, git):
    print("You created a new project! Keep saving journalism and make us proud.")
```

A common use case for such a hook in the real world is to create a new repository in your organization's version control system and set up default tickets.

3.8.4 Handling requirements

Tarbell blueprints can include a `requirements.txt` file in the same format used by `pip`. These requirements will be installed when the Tarbell blueprint or a project that uses it is installed.

Reference

4.1 Remote configuration

Note: Improving remote use is significant goal for Tarbell 1.2.

To use run Tarbell on a remote server, you'll want to generate a credentials file locally and transfer your Tarbell configuration directory to the server.

Assuming you have Tarbell properly configured, create a credentials file:

```
tarbell credentials > ~/.tarbell/credentials.json
```

Now copy the `~/.tarbell` directory to your server. You could use `scp`, `rsync`. A preferred way of doing it is to use a private git repository:

```
ssh myuser@myserver.tld
git clone git@github.com:myuser/tarbell-secrets.git .tarbell
```

Once the credentials file is in place on the server, you could create a `fabfile.py` for deployment from your server.

```python
from fabric.api import *

env.hosts = ['servername']
env.user = 'myuser'
env.directory = '/home/myuser/virtualenvs/project'


@task
def publish(target='staging'):
    with cd(env.directory):
        with prefix('workon tarbell'):
            run('tarbell publish {0}'.format(target))
```

Now you can run `fab publish:production` and deploy from your server.

Or perhaps a simple cron job, with a line like this in the crontab:

```
*/15 * * * * myuser /home/myuser/projects/myproject/cron.sh
```

And a little bash script like this.

```bash
#!/bin/bash

workon myproject
```

```
cd /home/myuser/projects/myproject
tarbell publish production
```

4.2 Managing projects

4.2.1 Show all projects with `tarbell list`

`tarbell list` shows all projects in your Tarbell projects directory.

4.2.2 Switch to a project with `tarbell switch <projectname>`

You can be in any directory, but the project must be in your default Tarbell projects directory.

4.2.3 Run preview server with `tarbell serve`

You must be in a Tarbell project directory to run Tarbell serve.

4.2.4 Update a project's blueprint with `tarbell update`

Run `tarbell update` to update a project's blueprint.

4.3 Hooks

Tarbell hooks allow project and blueprint developers to take actions during Tarbell project creation, projection installation, generation, and publishing.

To define a hook, edit `tarbell_config.py` or `blueprint.py`:

```python
from tarbell.hooks import register_hook


@register_hook("newproject")
def create_tickets(site, git):
    # ... code to create tickets on service of your choice
```

Here is a more advanced hook from the Bootstrap blueprint that prompts the user to create a new repo for their project on Github after project creation:

```python
import requests
import getpass

from clint.textui import puts, colored
from tarbell.hooks import register_hook


@register_hook('newproject')
def create_repo(site, git):
    create = raw_input("Want to create a Github repo for this project [Y/n]? ")
    if create and not create.lower() == "y":
        return puts("Not creating Github repo...")

    name = site.project.DEFAULT_CONTEXT.get("name")
```

```
    user = raw_input("What is your Github username? ")
    password = getpass.getpass("What is your Github password? ")
    headers = {'Content-type': 'application/json', 'Accept': 'application/json'}
    data = {'name': name, 'has_issues': True, 'has_wiki': True}
    resp = requests.post('https://api.github.com/user/repos', auth=(user, password), headers=headers,
    puts("Created {0}".format(colored.green("https://github.com/{0}/{1}".format(user, name))))
    clone_url = resp.json().get("clone_url")
    puts(git.remote.add("origin", "git@github.com:{0}/{1}.git".format(user,name)))
    puts(git.push("origin", "master"))

    create = raw_input("Would you like to create some default issues [Y/n]? ")
    if create and not create.lower() == "y":
        return puts("Not creating default issues")

    for title, description in ISSUES:
        puts("Creating {0}".format(colored.yellow(title)))
        data = {'title': title, 'body': description}
        resp = requests.post('https://api.github.com/repos/{0}/{1}/issues'.format(user, name), auth=
```

Use the publish hook to update the Facebook cache:

```python
import urllib

from clint.textui import colored, puts
from tarbell.hooks import register_hook

def _ping_facebook(url):
    url = "{0}?fbrefresh=CANBEANYTHING".format(url)
    fb_url = "http://developers.facebook.com/tools/debug/og/object?q={0}".format(urllib.quote_plus(ur
    urllib.urlopen(fb_url)
    puts("Pinging {0}".format(colored.yellow(fb_url)))


@register_hook('publish')
def update_facebook(site, s3):
    for name, path in s3.find_file_paths():
        if name.endswith(".html"):
            _ping_facebook("http://{0}/{1}".format(s3.bucket, name))
```

4.4 Command line reference

4.4.1 `tarbell configure`

Usage: `tarbell configure <optional:  subcommand>`

Subcommands: `drive`, `s3`, `path`, `templates`

Configures Tarbell. Specifying a subcommand will set up just that section of the configuration.

4.4.2 `tarbell newproject`

Usage: `tarbell newproject <optional:  projectname>`

Create a new Tarbell project.

4.4.3 `tarbell serve`

Requires current directory to be a Tarbell project.

Usage: `tarbell serve <optional: ip:port>`

Run a preview server. Specify an optional listening address, such as "0.0.0.0" or "192.143.23.10:5000".

4.4.4 `tarbell publish`

Requires current directory to be a Tarbell project.

Usage: `tarbell publish <optional: target, default: staging>`

Publish a Tarbell project to Amazon S3 using optional target (configured in `tarbell_config.py`).

4.4.5 `tarbell update`

Requires current directory to be a Tarbell project.

Usage: `tarbell update`

Updates blueprint with git submodule update.

4.4.6 `tarbell generate`

Requires current directory to be a Tarbell project.

Usage: `tarbell generate <optional: output directory>`

Make HTML on the file system. If output directory is not specified, a temporary directory will be used.

4.4.7 `tarbell switch`

Usage: `tarbell switch <project name>`

Serve the project specified by project name if it exists in your default Tarbell project directory.

4.4.8 `tarbell list`

Usage: `tarbell list`

List projects from your default Tarbell project directory.

4.4.9 `tarbell install`

Usage: `tarbell install <git repository url>`

Install a Tarbell project from the Git repository url specified.

4.4.10 `tarbell install-blueprint`

Usage: `tarbell install-blueprint <git repository url>`

Install a Tarbell blueprint from the given repository url.

4.4.11 `tarbell credentials`

Usage: `tarbell credentials`

Print JSON with the current OAuth credentials.

4.5 Configuration reference

4.5.1 Project settings (`tarbell_config.py`)

Each project has a `tarbell_config.py` file that controls settings for the project. These are the possible configuration variables:

NAME Short name of project, such as 'myproject' (required)

TITLE Descriptive title of project, such as 'My award winning project' (required)

EXCLUDES A list of files to exclude from publication such as `["*.txt", "img/mockup.psd"]` (optional)

CREATE_JSON Boolean. If true, spreadsheet will be previewed and published as `data.json` (default: False)

SPREADSHEET_CACHE_TTL How long to cache spreadsheet values, in seconds (default: 4)

SPREADSHEET_KEY If provided, Tarbell will use a Google spreadsheet with this key for template context (optional)

CONTEXT_SOURCE_FILE If provided, Tarbell will use this data file for the template context. CSV, XLS, and XSLX files are supported. The value may use a relative path, an absolute path, or a remote (http) URL. (optional)

S3_BUCKETS A dict of target->url pairs such as `{ 'production': 'apps.myorg.com' }` (required for publishing to S3)

DEFAULT_CONTEXT A dict of fallback values for the project context. Use this if you don't want or need a Google spreadsheet or external file.

4.5.2 Tarbell settings (`~/.tarbell/settings.yaml`)

The settings file uses a simple YAML-based format:

```
default_s3_access_key_id: <DEFAULT KEY ID>
default_s3_secret_access_key: <DEFAULT SECRET KEY>
default_s3_buckets:
  production: apps.chicagotribune.com
  staging: apps.beta.tribapps.com
google_account: davideads@gmail.com
project_templates:
- name: Basic Bootstrap 3 template
  url: https://github.com/newsapps/tarbell-template
- name: Searchable map template
  url: https://github.com/eads/tarbell-map-template
projects_path: /Users/davideads/tarbell
default_server_ip: 127.0.0.1
```

```
default_server_port: 5000
s3_credentials:
  26thandcalifornia.recoveredfactory.net:
    access_key_id: <KEY ID>
    secret_access_key: <SECRET KEY>
```

This example shows every possible setting.

google_account Default Google account to use when creating new projects

project_templates A list of `{name:  ..., url:  ...}` objects with project templates.

projects_path Path to the user's Tarbell projects

default_s3_access_key_id Default key ID to use when publishing

default_s3_secret_access_key Default key to use when publishing

default_s3_buckets alias->s3 url pairs to be used during project creation for setting up default bucket aliases. These are only used during project creation and can be overridden on a per-project basis.

s3_credentials Define S3 credentials using a `bucket-uri->{ access_key_id:  ...,` `secret_access_key:  ...}` data strucutre.

default_server_ip Default preview server IP address

default_server_port Default preview server port

4.5.3 Google SDK client secrets (`~/.tarbell/client_secrets.json`)

Place a `client_secrets.json` file in `~/.tarbell` or use `tarbell configure drive`.

4.6 Contributing

- Github repository: https://github.com/newsapps/flask-tarbell