Static Site Generation

Lecture 17
What is Static Site Generation?

- Use a program to produce HTML pages
  - Analogous to compiling programs
  - Source code → machine code

- Development cycle:
  - Write source
  - Compile
  - Test/inspect result

- Examples
  - Jekyll (used for "GitHub Pages")
  - Middleman
  - Lots more, see: staticsitegenerators.net
Picture

source files

-generated web site

.html

.css

.md

.erb

.scss
Middleman: A Ruby Gem

- Project is a directory (eg myproj)
  
  `$ middleman init myproj`

  - Configuration files, README, Gemfile, etc

- Create source files in `myproj/source`
  
  - Subdirectories for CSS, images, etc

- Compile all the source files
  
  `$ bundle exec middleman build`

- Result is placed in `myproj/build`

- Copy site to some visible location
  
  `$ rsync -avz --del myproj/build ~/WWW`

- Or preview locally (live reload, no build)
  
  `$ bundle exec middleman server`
Why Bother?

1. Code reuse and single-point-of-control over change
2. Authoring of content in a language that is more human-friendly
3. Parameterized generation of markup and content

Let's look at each of these benefits in turn...
Motivation #1: Visual Identity

- Common headers & footers
  - Example: OSU web sites share nav bar
  - Example: course web site

- Duplication of code is EVIL
  - Corollary: cut-and-paste is EVIL
  - Destroys single-point-of-control over change

- Solution:
  - Put common HTML in one file (a "partial")
  - Every document includes that file
**ERb: Embedded Ruby**

- General templating mechanism
  - A string (usually contents of a file, "template")
  - Escaped bits of ruby
    - `<% code %>` execute ruby code ("scriplet")
    - `<%= expr %>` replace with result of expr
    - `<%# text %>` ignore (a comment)

- Example: a text file
  ```ruby
  This is some text.
  <% 5.times do %>
  Current Time is <%= Time.now %>!
  <% end %>
  ```

- Process using erb tool to generate result
  ```bash
  $ erb example.txt.erb > example.txt
  ```

- Naming convention: *filename.outputlang.erb*
  - Example `index.html.erb`

- Many alternatives, eg HAML
Generation of Site

- Source files in myproj/source
  - $ ls source
    - index.html.erb
    - syll.html.erb
    - meet.html.erb

- Compile
  - $ bundle exec middleman build

- Result after building
  - $ ls build
    - index.html
    - meet.html
    - syll.html
Partials

- A document fragment included in other documents
- Include in erb using the `partial` function
  ```erb
  <body>
    <%= partial "navigation" %>
    ...
  </body>
  ```
- Partial's filename begins with `_`
  ```erb
  <div class="navbar">
    <ul id="site-nav">
      <li> ... </li>
    </ul>
  </div>
  ```
  Note: `_` omitted in argument to function
Generation of Site with Partials

- Source files in myproj/source
  
  ```bash
  $ ls source
  _footer.erb  meet.html.erb
  _navigation.erb  syll.html.erb
  index.html.erb
  ```

- Compile
  
  ```bash
  $ bundle exec middleman build
  ```

- Result after building
  
  ```bash
  $ ls build
  index.html.meet.html.syll.html
  ```
Tricks with Partials

- Content of partial can be customized by passing arguments in call
- In calling erb: pass a hash
  ```erb
  <%= partial "banner",
    :locals => { :name => "Syllabus",
               :amount => 34 }
  %>
  ```
- In partial: access variables
  ```erb
  <h3> <%= :name %> </h3>
  <p> Costs <%= "$#{:amount}.00" %></p>
  ```
Problem

- How to guarantee every page includes partial(s)
  - Partials don't ensure one page *structure* across the site
- Every page should look like:

```html
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Meetings</title>
  <link rel="stylesheet" type="text/css" href="osu_style.css">
</head>
<body>
  <%= partial "navigation" %>
  ... <!-- different on each page -->
</body>
</html>
```
Solution: Layout

- HTML formed from: Layout + Template
- Layout is the overall structure of the HTML page
- File: layout.erb
  ```html
  <!DOCTYPE html>
  <html>
  <head>
    <meta charset="utf-8">
    <title> ... etc
  </head>
  <body>
    <%= yield =>
  </body>
  </html>
  ```
- Template replaces layout's yield
- Layout is where you put site-wide styling
  - Eg navigation bar, div's with CSS classes, footers
Page-Specific Data in Layout

- Some layout content is page-specific
  - Example: `<title>` in document's head
- Solution: Ruby variable `current_page`
  - Example: `current_page.path`
- Template contains "frontmatter" that sets the value of `current_page.data`
  - In template (contact.html.erb)
    ```ruby
    ;;;;
    "title": "Contact Information"
    ;;;;
  ```
  - In layout (layout.erb)
    ```ruby
    <title> <%= current_page.data.title %>
    </title>
    ```
Generation of Site with Layouts

- Default layout in `source/layouts/layouts.erb`

```
$ ls -F source
index.html.erb meet.html.erb
layouts/
layout.erb
$ ls source/layouts
_footer.erb _navigation.erb layout.erb
```

- Result after building

```
$ ls build
index.html meet.html syll.html
```
Motive #2: Improved Syntax

- HTML tags make content hard to read
  - `<p>`, `<h2>`, `<em>`, `<a href='...'>` etc
  - vs plain text, which is easier to read

- Common plain text conventions:
  - Blank lines between paragraphs
  - Underline titles with `-`'s or `=`'s
  - Emphasize `*`words*, `_`words_`, **`words`**
  - Links in ( )'s
  - Unordered lists with bullets using `*` or `-`
  - Numbered lists with 1., 2., 3.
Markdown

- Formalizes these ASCII conventions
  - Filename extension: `.md`
  - Adds some less familiar ones (eg ````)`

- Tools generate corresponding HTML
  - Examples: GitHub readme's, user-posted comments on web boards (StackOverflow)
  - Other target languages possible too

- See Markdown's README.md
  - Regular view
  - Raw view

- Warning: many Markdown dialects
CSS Limitations

- Literals are common
  ```
  h1 { background-color: #ff14a6; }
  h2 { color: #ff14a6; }
  ```

- Result: Lack of single-point-of-control

- Solution: SASS, allows variables
  ```
  $primary: #ff14a6;
  h1 { background-color: $primary; }
  h2 { color: $primary; }
  ```

- Preprocessor generates CSS from SASS
Motive #3: Content Generation

- Iterate over an array to create content
  - Example: rows of a table
  - See course web site
  - Partial could be used for (every) row

- Placeholder text and images
  `<%= lorem.sentence %>`
  - Many versions of this "helper" are available
    - `lorem.paragraphs 2`
    - `lorem.date`
    - `lorem.last_name`
    - `lorem.image('300x400')`  
      - `#=> http://placehold.it/300x400`
More Helper Functions

- **Links**
  ```ruby
  <%= link_to 'About', '/about.html' %>
  #=> <a href='/about/'>About</a>
  ```

- **Assets**
  ```ruby
  <%= stylesheet_link_tag 'all' %>
  <%= javascript_include_tag 'jquery' %>
  <%= favicon_tag 'images/favicon.png' %>
  <%= link_to 'Blog', '/blog', :class => 'example' %>
  <%= image_tag 'padrino.png', :width => '35', :class => 'logo' %>
  ```
Summary

- **ERb**
  - Template for generating HTML
  - Scriplets and expressions

- **Reuse of views with partials**
  - Included with partial (eg `<%= partial...`)
  - Filename is prepended with underscore
  - Parameter passing from parent template

- **Layouts and templates**

- **Markdown, SASS**

- **Content generation and helpers**