Webengine

Matthew B. Harris

Abstract* — Webengine is an ancient piece of Kryptonian technology designed to terraform an outdated website such that its style and content are aligned with newer websites [25]. Keeping a website up-to-date, not only with content but also with styling, is what keeps users and customers coming back. They may not always be happy with these changes (look at Facebook), but the ability to quickly refresh your website is imperative to staying relevant. Two new Python scripts *webengine.py* and *webshooter.py* strives to do just that. The days of dynamically creating basic websites using php and mysql have passed. Powerful new tools such as Hyde and Jekyll are the keys to quick website creation today. With *webshooter*, a system administrator or even a project leader can create a new and fresh looking website with one of three different style options and then populate it with the data and content from the previous website by running *webengine.

***Index Terms* — Hyde, Jekyll, Python, Web 2.0**

I. Introduction

EVERY project or team needs a website. But sometimes once the website is deployed, it is forgotten and not well maintained. As the new guy on my team, I was charged with maintaining and updating our websites.

The first task was converting a Moin Moin wiki [14] to Markdown [9] to post on the project's GitHub [4] wiki. This presented a number of challenges. First I tried to use the update and export tools Moin Moin provided, but to no avail. I then tried to wget the wiki, also to no avail. The wiki presented files in such a way that wget did not guarantee the recovery of all the pages. I performed a simple Internet search for web scrapping tools, but nothing matching my use case turned up.

I therefore began writing *webengine* as a simple one-off script. After completing the Moin Moin wiki conversion, other opportunities to use *webengine* surfaced in updating our other websites, along with being the missing piece to *webshooter*. With the two scripts, you can now create a new web 2.0 website and then populate it with the data in an existing website.

I started abstracting *webengine* to different scripts and attempted to generalize the code for reuse. Every website and wiki I updated or converted offered different challenges and corner cases that required additional code or scripts. In section three, Content Management Systems, I explain more of *webengine's* history and why it came to exist.

I learned from the Moin Moin wiki conversion that the best way to recover all the pages of a website was to wget the pages listed on the website's index page. Because in most cases there are more than a hundred pages listed, a script is required to automate this process. *webengine.py* first uses wget to retrieve the wiki's index page, copies all the pages URLs (*url_gather.py*),and saves them in a file *URL.txt*. Second *webengine.py* retrieves every URL in the newly created *URL.txt* file (*file_gather.py*) using wget.

These actions were the start of building a collection of scripts for web scraping website data and content for manipulation. This Python [21] project heavily used the Beautiful Soup [2] package. The bulk of the file converting (*file_convter.py*) used Aaron Swartz's *html2text.py* script [18].

This project's origin can be traced back to a single script my intern Ben Carlsson and I wrote in the summer of 2013. *webshooter.py* [5] created a new Hyde [11] based website. *webengine.py* was the missing piece in *webshooter.py*. Once a new website is created, the next step is populating it.

II. Software used

A. Hyde

Hyde is a static website generator written in Python. While in literature Hyde took life as awesome Jekyll's evil twin, it has since been completely consumed by the dark side and has taken on an identity of its own. Hyde the website generator desires to fulfill the lofty goal of removing the pain points involved in creating and maintaining static websites. [11]

Ben Carlsson and I started with Hyde because it is written in Python and our team is a Python shop. Furthermore, Hyde uses the jinja2 [13] template engine, which gave us lots of options in page design.

B. Jekyll

Jekyll is a simple, blog-aware, static website generator. It takes a template directory containing raw text files in various formats, runs it through Markdown (or Textile) and Liquid converters, and outputs a complete, ready-to-publish static website suitable for use on a favorite web server. Jekyll also happens to be the engine behind GitHub Pages [20], which means one can use Jekyll to create a project's pages, blog, or website from GitHub's servers for free. [12]

I started using Jekyll (Ruby-based) when the Earth System Grid Federation (ESGF) [8] asked me to move their new website to GitHub Pages [20]. That required changing from Hyde to Jekyll.

C. Webshooter

Webshooter is a command-line assistant to the static website generator Hyde. It produces Hyde-ready websites in the user's choice of layout and automates page creation and website configuration through user prompts and command line arguments. [5]

III. Content Management Systems

Content management systems (CMSs) [6] are a great way to

Manuscript received June 30, 2014; revised July 28, 2014. This work was funded by the AIMS group at Lawrence Livermore National Laboratory as a toolkit to expedite the process of converting and creating new websites.

M. B. Harris is a Computer Scientist, Mathematical Programmer in the AIMS Project at Lawrence Livermore National Laboratory, Livermore, CA 94550 USA (phone: 925-423-8978 fax: 925-422-7675 email: harris112@llnl.gov).

quickly get a website styled. They are also user and administrator friendly. My team is using Drupal [7], which has a nice community around it. Plone, Zope, Moin Moin, and other CMSs also exist.

- Drupal is an open-source content management platform powering millions of websites and applications. It is built, used, and supported by an active and diverse community of people around the world.[7]
- Plone is among the top 2% of all open-source projects worldwide, with 340 core developers and more than 300 solution providers in 57 countries."[16]
- Zope is a free and open-source web application server written in the object-oriented programming language Python. [17]
- Moin Moin is an advanced, easy to use and extensible WikiEngine with a large community of users. Said in a few words, it is about collaboration on easily editable web pages. [15]

While CMSs have many advantages, there are also disadvantages. We were using a web server running RedHat4 and hosting a Plone version 1.5 website. However, Lawrence Livermore National Laboratory (LLNL) was discontinuing support for RedHat4 in favor of the latest version, RedHat6. A colleague, Dr. Jeffery Painter attempted to update the Plone website to the latest version. However, the upgrade tool failed because the system packages were too new. He tried rolling these packages back to pervious versions, but Plone and other applications crashed. He next tried the export tool, which had the same problem of packages not being the right version, which made it impossible to connect to the database.

To resolve the issue of allowing a content management system to become out of date, our solution was to stay pro-active. I wrote a cron job that performs a Drupal update on all of our websites once a week.

CMSs make the output HTML unnecessarily complex Table I shows a small excerpt of two links on a Drupal page with numerous tags for special styling and website generation. Unless one stays up to date with the CMS and keeps content new and relevant, a CMS can quickly spiral out of control.

TABLE I
OUTPUT HTML FROM A CMS

```
<div class="content">
   <ul class="menu clearfix">
      <li class="first collapsed">
         <a href="/?q=biblio" title="">Biblio</a>
         <span class="icon"></span>
      </li>
      <li class="last leaf">
         <a href="/?q=tracker">Recent content</a>
         <span class="icon"></span>
      </li>
   </ul>
</div>
```

IV. WEBENGINE.PY

Table II shows the help output of the *webengine.py* script, which explains the required inputs. *webengine.py* collects the user input, and delegate the work to the correct script.

The *webengine.py* process we developed works as follows. If the user enters a type of "wiki," two scripts will run. First *url_gatherer.py* runs, downloading the index page from the wiki or website and creating a file with all the website pages listed. Then *file_gather.py* will read in the file and wget all the pages. Alternately, *website_gather.py* will run, which does a pure wget on the URL the user passed in. Now that the website has been collected, the work of converting raw html to Markdown can begin.

TABLE II
WEBENGINE HELP OUTPUT

```
python webengine.py -h
usage: webengine.py [-h] URL href src type case

get all content (html / images) from a wiki (website) and convert to
markdown

positional arguments:
  URL         URL to website or wiki
  href        URL of the new website
  src         Path to the image directory
  type        "website" for full website or "wiki" for wiki Title Index page
  case        jekyll, hyde, none

optional arguments:
  -h, --help  show this help message and exit
```

file_extractor.py will recursively read all the php or html files and will first look for a division tag (div) with a class or identification (id) with the name of "content" or "container" inside the body tag. If neither of these tags exist, it will extract the entire html out of the body tag and save this reduced copy of the page. *file_corrector.py* will read in every file and change all the hypertext reference links (href) and image sources (src) to what the user has passed in.

html_table_2_Markdown.py then reads through all the files looking for table tags and converts them to Markdown. *file_converter.py* calls Aaron's *html2text.py,* which converts the remaining html to Markdown. *image_gatherer.py* scans all the original html and downloads any missing images, and *bold_cleanup.py* scans all the completed Markdown files and removes any stray bold or italics symbols (e.g., "**", "__").

head_adder.py adds either the Hyde or Jekyll page header to all the Markdown pages. If the choice is Hyde, it will also change all the files extensions from .md to .html—this makes parsing files in Hyde easier.

With *webengine.py* completed, the directories have been created, as shown in Table III.

TABLE III
WEBENGINE OUTPUT ON COMPLETION

```
whole_site        -> The whole site from wget
extracted_files   -> just the body of the raw html files
html_files        -> extracted file corrected with new href and src links
html_dirty_files  -> html files with converted tables html to markdown
completed_files   -> md or html files as you asked for
image_files       -> all the image files from the old site (images)

You may want to run md_files through file_trimmer.py to remove
unwanted headers and footers
If so remember to then run head_adder.py again by hand
```

V. WEBSHOOTER.PY

Running *webshooter.py* will create a new website that can be populated with files from the *webengine.py's completed_files* directory. The output from *webshooter.py* help can be seen in Table IV. *webshooter.py* currently offers the user three website styles to choose from. *webshooter.py*

is a single, self-contained script for creating a new website using Hyde.

TABLE IV
WEBSHOOTER HELP OUTPUT

```
python webshooter.py

I was written in Python 3.x, but you're running me with Python 2.x!
I was NOT tested with this version. Run anyway? [y/N]

usage: webshooter.py <command> [--help]

commands:
  new              Interactively create a new website
  gen <website_path>  Regenerate the content for the website at
  `website_path`
```

With *webshooter.py,* one can use any of the following styling frameworks. Examples of each are included.

- Bootstrap is a sleek, intuitive, and powerful front-end framework for faster and easier web development, created by Mark Otto and Jacob Thornton, and maintained by the core team with the massive support and involvement of the community [3]. This style is implanted on our project websites as shown on the Ultrascale Visualization Climate Data Analysis Tools (UV-CDAT) website [19] in Figure 1. Bootstrap is developer friendly, providing Cascading Style Sheets (CSS), Fonts, and JavaScript (JS) files. It is very customizable and is mobile ready.
- one.5lab is a clone of the one-Lab website styling implemented at LLNL [14]. This style was used for the Analytics and Informatics Management Systems group website [1], shown in Figure 2. LLNL's one-lab CMS provides a style consistent across all Lab websites. I created a fork one.5lab to implement in our Hyde websites.
- Tshirt is a web 1.0, simple page style that was the original style for our project websites, seen in Figure 3. It has not aged well but it can work for quickly displaying information. This style will be removed from our project websites.

VI. EXPERIENCE

A. Time savings

This project started as a way to save time web scraping data from our obsolete websites and wikis. I did not want to recreate every website and wiki from scratch or recreate CSS. With *webshooter.py* and *webengine.py*, transforming old content from static or dynamically created pages from databases proved to be quick and easy. In addition, using Apache or other web-hosting option is quicker and easier for serving static webpages than php and mysql.

B. Creating updated websites

Having a website and/or wiki is a quick way to share data and instructions with users. Because styling is important but not the focus of our work, plugging in new popular or home brewed styles is easy.

C. Dynamically created static webpages

Having the content in simple files (HTML, Markdown, or plain text) allows for parsing and creation of static pages quickly and efficiently. For example, if you need to update the navigation menus, you simply update the topbar.j2 file and re-generate the website again.

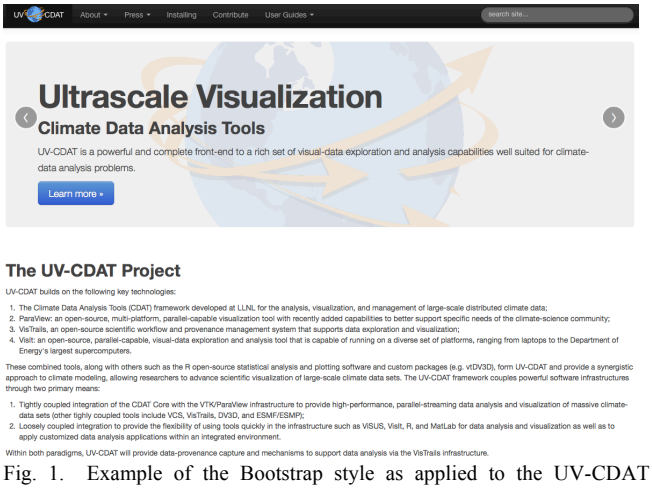

Fig. 1. Example of the Bootstrap style as applied to the UV-CDAT website.

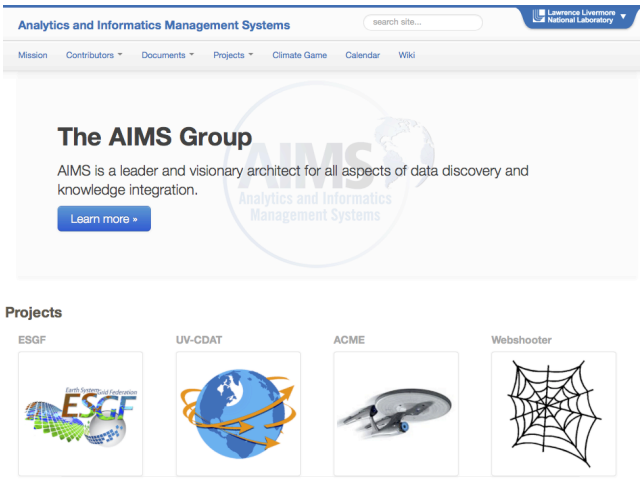

Fig. 2. Example of the one.5lab style as applied to the AIMS website.

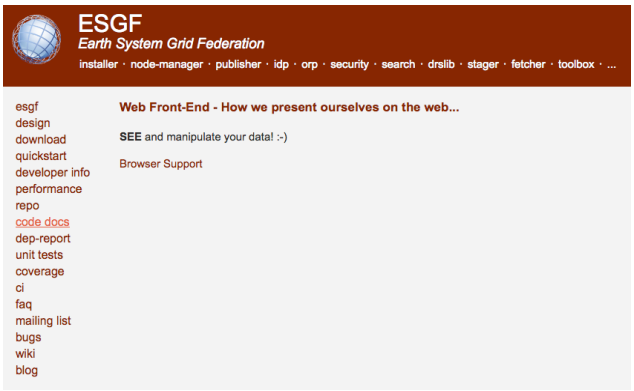

Fig. 3. Example of the tshirt style as applied to the old ESGF website.

D. Hyde vs. Jekyll

Initially, Hyde appeared to be the best option for *webshooter.py*. Hyde is a powerful tool, allowing for considerable alterations and customization of the website. An example is creating a carousel of images in a website banner, which requires defining a new variable in the *website.yaml* (Table V) file. In the same file is a menu variable, which allows the carousel to be generated using

the *macros.j2* (Table VI) file. This makes updating shared parts of the website quick.

TABLE V
EXAMPLE OF A WEBSITE.YAML FILE

```
context:
  data:
    home_url: index.html
    nav_hover: yes
    menu:
      - title: UVCDAT
        url: index.html
      - title: About
        url:
          - title: Mission
            url: mission.html
          - title: Governance
            url: governance.html
          - title: Committee
            url: committee.html
          - title: Acknowledgments
            url: acknowledgments.html
    website_title: UV-CDAT
    carousel:
      - caption: Sample output
        image: media/images/gallery/CelinesCaliforniaPlot.png
      - caption: Sample output
        image: media/images/gallery/figure3b.png

media_url: media
mode: development
plugins: [hyde.ext.plugins.meta.MetaPlugin,
          hyde.ext.plugins.auto_extend.AutoExtendPlugin,
          hyde.ext.plugins.syntext.SyntextPlugin,
          hyde.ext.plugins.textlinks.TextlinksPlugin]
```

TABLE VI
EXAMPLE OF A MACROS.J2 FILE

```
{% macro render_carousel_items(items) -%}
  {% for carousel_item in items -%}
    <div class="item peopleCarouselImg">
      <div class="flipbox-container box100">
        <div id="flipbox1" class="flipbox">
          <center>
            <img src="{{ carousel_item.image }}" class="thumbnail" />
            <div class="carousel-caption">
              <p>{{ carousel_item.caption }}</p>
            </div>
          </center>
        </div>
      </div>
    </div>
  {%- endfor %}
{%- endmacro %}
```

Table 6

However, once I started using Jekyll to publish websites to GitHub Pages [20], I decided it is a better option for quick and simple websites, such as documentation and tutorials. Jekyll is much faster from start to deploy of a basic website. The top navigation bar lives in a simple-to-read HTML file in the *_includes* directory and is included on every page.

My conclusion is that Hyde is preferable for developing complex and robust websites where only few developers will contribute changes. For smaller, simpler websites, which may have many contributors, Jekyll is the good choice.

E. Power of Markdown

Markdown comes in a few flavors one of the most used is GitHub flavored Markdown [9]. Markdown allows one to write using an easy-to-read, easy-to-write plain text format, which then converts to valid HTML. It takes up less space in the file and is easy for anyone to learn.

F. Websites built with webengine and or webshooter

1. http://aims.llnl.gov [1]
2. http://cfconventions.org
3. http://esgf.org [8]
4. http://esgf.org/wiki [24]
5. http://uv-cdat.org [19]
6. http://uv-cdat.org/wiki [19]
7. http://kitt.llnl.gov/cdat [22]
8. http://kitt.llnl.gov/cmor [23]
9. http://mattben.info

VII. FUTURE WORK

A. Further testing

Currently, I run *webengine.py* and modify its collection of scripts until I receive the desired output. This process is not agile development by any standard, and it would be better to standardize the script's naming conventions. This will help to stop confusion on further development and make the process of reading each script simpler for future developers. Unit tests ensure that changes or the addition of new scripts does not break or inject bad code. Each script in the repo is able to run independent of *webengine.py,* and each is expecting a number of command line arguments. This process needs to be simplified and made consistent across them all. Refactoring the code to make sure each script performs one and only one task by extracting any repeated task to a new script will help maintain a strong code base.

B. Solving more corner cases

Most websites we encounter are structured in files, such that all the pages about topic "X" are in the "X" directory. However, the *file_converter.py* script needs to keep track of this structure and rename the files to make sense in a flat file system. An example is the file *X/Stuff/index.php*, which would become something like *X-Stuff-Home.md*.

Currently *html_table_2_Markdown.py* will read through a file and convert a standard html table into Markdown. But it also needs to be able to handle such cases as tables inside of tables.

Links for internal or external websites are not always parsed correctly because of the many formats used to display the URL in <a> tag. For *file_converter.py* to update each link correctly every time, it needs to have more information about the link it is converting. If the link is an internal reference, *file_converter.py* will need a way to check if that page has changed its name and/or location, as mentioned above. All links referencing external websites or pages can be copied directly without modification.

C. webengine needs to be more interactive

The user needs to be able to choose which scripts to run and in what order. For example, if the entire website is complete, there is no need to fetch it again. Or if old website images are not needed, there is no need to download them.

D. Adding more options to webshooter.py

webshooter.py only supports Hyde websites. We hope to modify it to include styles for Jekyll. This year, I will update the three style repositories to include a set of files for Jekyll as well as Hyde. With these updated style repos, I will update *webshooter.py* to ask the user whether to create a Hyde or Jekyll website. Also, currently converting an existing website is a two-step process. I plan to simplify this by having *webshooter.py* ask the user once the new website is created if there is a URL s/he would like to use to populate his/her new website. If so, *webengine.py* will swing into action.

E. Automation

Once *webengine.py* and *webshooter.py* are consistently delivering the expected results, I will develop a script *automated_update.py*. This script will read in the files of popular websites provided by the system administrator. *automated_update.py* will then periodically web scrape the styles of these websites. *automated_update.py* will then call *webengine.py* and *webshooter.py* to automatically generate new updated websites, and send an email to the administrator that there are new websites to choose from.

VIII. Conclusion

The Webshooter repository [10] started with one file that my intern Ben Carlsson and I started in June 2013. I could not have imagined that a year later I would have a repository with over 20 files that automate tasks my team and I do regularly. This project has been a great learning and teaching experience for my team and most of all for me. I hope others will find it useful and even contribute to it. (To contribute to the *webshooter* [10] repository, simply make a fork and submit a pull request.)

Acknowledgments

I would like to thank Sam Fries for his time debugging, testing, and contributing to the code base. I would also like to recognize Ben Carlsson, Dean Williams for their support and contributions. Special thank you to Katie Walter, and Brian Harris for their editorial skills. LLNL-CONF-656083

References

[1] "The AIMS Group." *Home*. LLNL, n.d. Web. 20 May 2014. <http://aims.llnl.gov/>.

[2] "Beautiful Soup." *We Called Him Tortoise Because He Taught Us*. N.p., n.d. Web. 20 May 2014. <http://www.crummy.com/software/BeautifulSoup/>.

[3] "Bootstrap." *Bootstrap*. Twitter, n.d. Web. 20 May 2014. <http://getbootstrap.com/>.

[4] "Build Software Better, Together." *GitHub*. N.p., n.d. Web. 20 May 2014. <https://github.com/>.

[5] Carlsson, Ben. "Webshooter, Your Friendly Neighborhood Static Website Layout Generator." *Webshooter PDF*. LLNL, n.d. Web. 20 May 2014. <http://mattben.info/media/pdf/webshooter.pdf>.

[6] "Content Management System." *Wikipedia*. Wikimedia Foundation, 22 June 2014. Web. 24 June 2014. <http://en.wikipedia.org/wiki/Content_management_system>.

[7] "Drupal." *Drupal*. N.p., n.d. Web. 20 May 2014. <https://drupal.org/>.

[8] "Earth System Grid Federation." *ESGF Home Page*. N.p., n.d. Web. 20 May 2014. <http://esgf.org/>.

[9] "GitHub Flavored Markdown." *GitHub Help·*. N.p., n.d. Web. 20 May 2014. <https://help.github.com/articles/github-flavored-markdown>.

[10] Harris, Matthew B. "Mattben/webshooter." *GitHub*. GitHub, 16 June 2013. Web. 20 May 2014. <https://github.com/webshootertk/webshooter>.

[11] "Hyde." *Overview*. N.p., n.d. Web. 20 May 2014. <http://hyde.github.io/>.

[12] "Jekyll." *Jekyll Simple Blogaware Static Websites*. N.p., n.d. Web. 20 May 2014. <http://jekyllrb.com/>.

[13] "Jinja2." *Welcome*. N.p., n.d. Web. 20 May 2014. <http://jinja.pocoo.org/>.

[14] "Lawrence Livermore National Laboratory (LLNL)." *Lawrence Livermore National Laboratory (LLNL)*. N.p., n.d. Web. 20 May 2014. <https://www.llnl.gov/>.

[15] "MoinMoin: MoinMoinWiki." *Moin Moin Wiki*. N.p., n.d. Web. 20 May 2014. <http://moinmo.in/>.

[16] "Plone 4: Speed, Power & Beauty." *Plone CMS: Open Source Content Management*. N.p., n.d. Web. 20 May 2014. <http://plone.org/>.

[17] "Start — Zope.org." *Start — Zope.org*. N.p., n.d. Web. 20 May 2014. <http://www.zope.org/>.

[18] Swartz, Aaron. "Aaron Swartz." *Aaron Swartz*. Https://github.com/aaronsw/html2text, n.d. Web. 20 May 2014. <http://www.aaronsw.com/>.

[19] "Ultrascale Visualization." *UV-CDAT*. N.p., n.d. Web. 20 May 2014. <http://uvcdat.llnl.gov/>.

[20] "Websites for You and Your Projects." *GitHub Pages*. GitHub, n.d. Web. 20 May 2014. <https://pages.github.com/>.

[21] "Welcome to Python.org." *Python.org*. N.p., n.d. Web. 20 May 2014. <https://www.python.org/>.

[22] Williams, Dean, and Charles Doutriaux. "CDAT Home." *CDAT Home*. uvcdat.llnl.gov, n.d. Web. 20 May 2014. <http://kitt.llnl.gov/cdat/>.

[23] Williams, Dean, Charles Doutriaux, and Jerry Potter. "Climate Model Output Rewriter Overview." *Climate Model Output Rewriter*. CMOR, n.d. Web. 20 May 2014. <http://kitt.llnl.gov/cmor/>.

[24] Williams, Dean. "ESGF/esgf.github.io." *GitHub*. Esgf.org, n.d. Web. 20 May 2014. <https://github.com/ESGF/esgf.github.io/wiki>.

[25] "World Engine." *DC Cinematic Universe Wiki*. N.p., n.d. Web. 20 May 2014. <http://dccinematicuniverse.wikia.com/wiki/World_Engine>.