Getting Started with Gollum

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

This is a pilot for an experimental way to develop short-form (less than 100 pages or so) content for O'Reilly. Here's how it works:

- We supply you with a Gollum instance, the awesome, git-powered wiki from the folks at GitHub. You write/edit your document like you would a wiki, and you can share it with friends, editors, and other people you think might help.
- You write in AsciiDoc, an open source text format developed by Stuart Rackham (more on this in a bit). Every time you save a page, it gets logged into a git repository, so it's always under version control. (We can grant you access to your repo — just ask!)
- When you're ready, we suck the AsciiDoc into our publishing toolchain, convert it to DocBook (our native format), and then push it out through various channels. You can read a bit about this at Matt Neuburg's blog, where he describes how he wrote his "Programming iOS" book. The editing environment provides some tools to help you "debug" your markup to meet DocBook's stringent requirements.

So, why bother with this, you ask? Mostly, we're trying to create a more appealing process that will get you writing as quickly as possible, not mucking around with a bunch of tools. The cool thing about AsciiDoc, Gollum, and git (and GitHub!) is that they can help ease a lot of the complexities of the publishing process, letting you (hopefully!) focus on the what you want to say.

This document covers:

- Signing in with OpenID
- Getting started with AsciiDoc
- Attaching figures and code samples
- Putting code examples on GitHub
- Building the project
- Debugging your project
- Downloading your archive
- Editing locally
- Tips and tricks

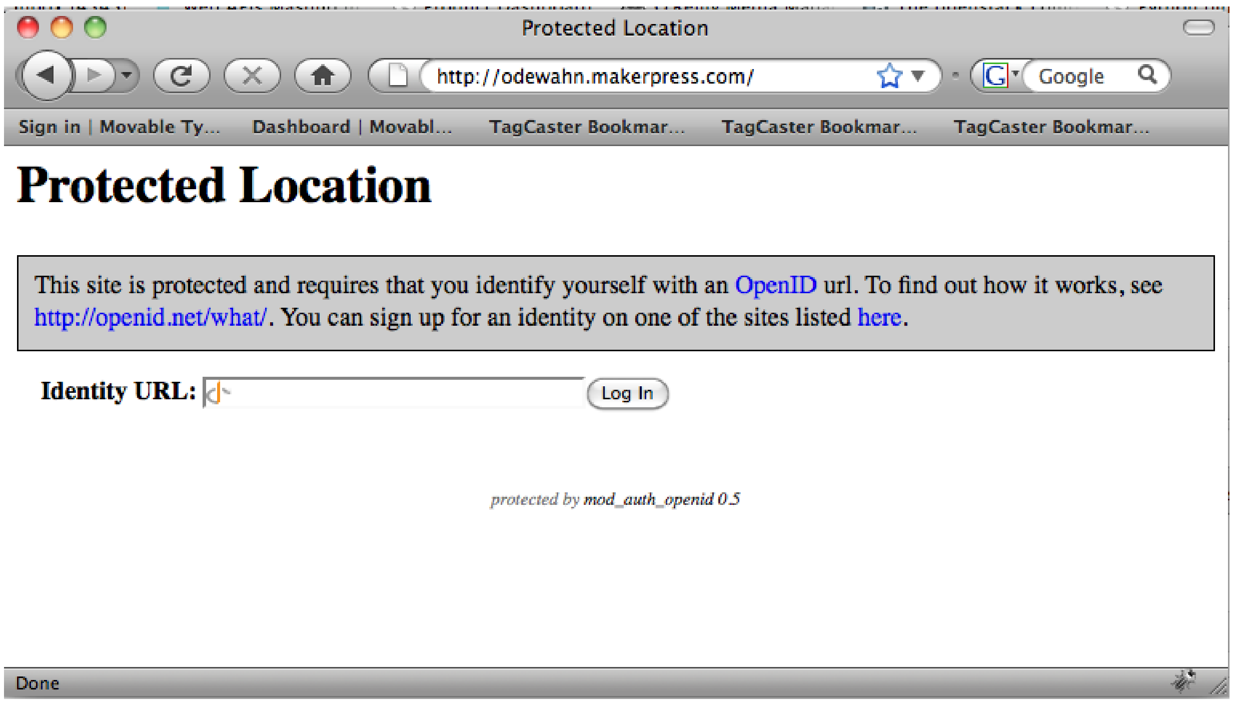
So, here come the rest of the sections.

2 Signing In With OpenID

The editing environment requires authentication using OpenID. There are many, many providers, ranging from Goolge, Flickr, myOpenID, to Verisign. Each provides a distinct identifier that you use to identify yourself. This section describes use the Google+ OpenID service.

Note
Unlike other providers, Google+ does not require you to provide a distinct identifier. Rather, you use a single, generic identifier — **https://www.google.com/accounts/o8/id** — which Google+ uses to retrieve your unique identity URL. Please note that this identifier is not specific to your account, and that you do not need to replace "id" (or anything else) with anything specific to your account. This is all done for you automatically.

Figure 1 shows the authentication screen you'll receive each time you are required to log into the system.



Protected Location

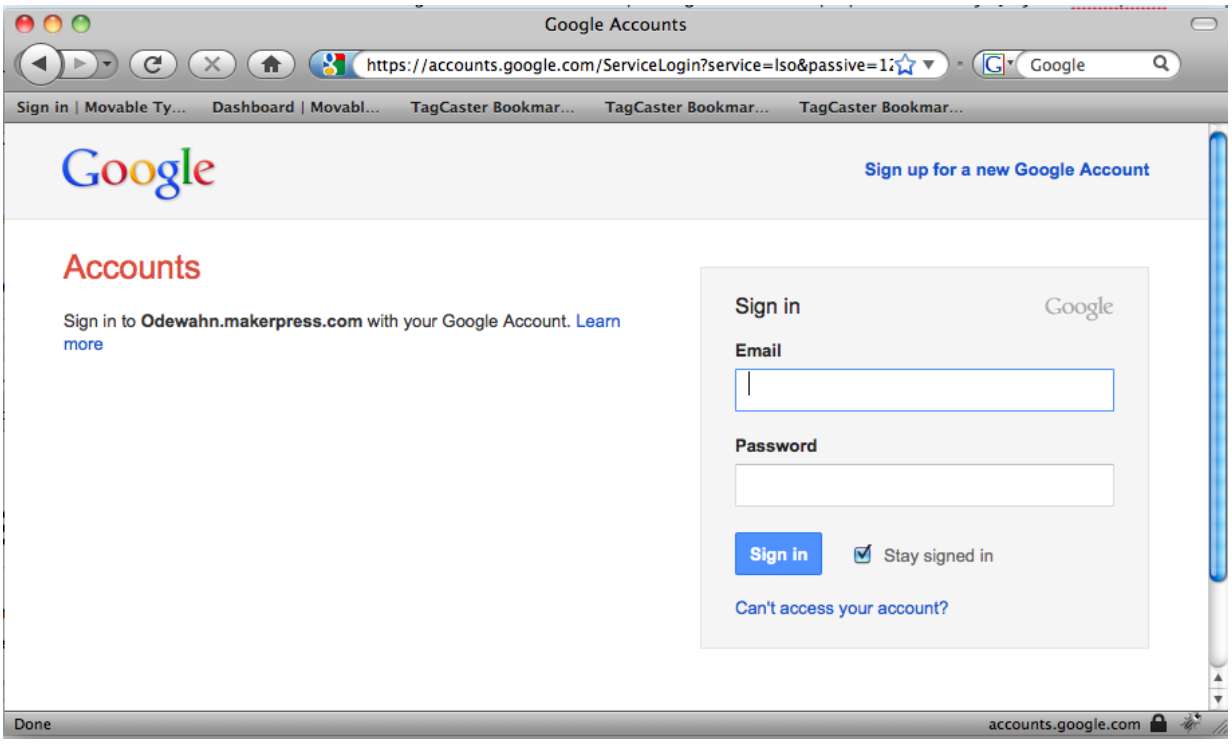
This site is protected and requires that you identify yourself with an OpenID url. To find out how it works, see http://openid.net/what/. You can sign up for an identity on one of the sites listed here.

Identity URL:

protected by mod_auth_openid 0.5

Figure 1: The OpenID signin screen.

To begin the authentication process, enter **https://www.google.com/accounts/o8/id** into the "Identity URL" box and press "Log In." You will be redirected to the sign in page for your Google Account, as shown in Figure 2.



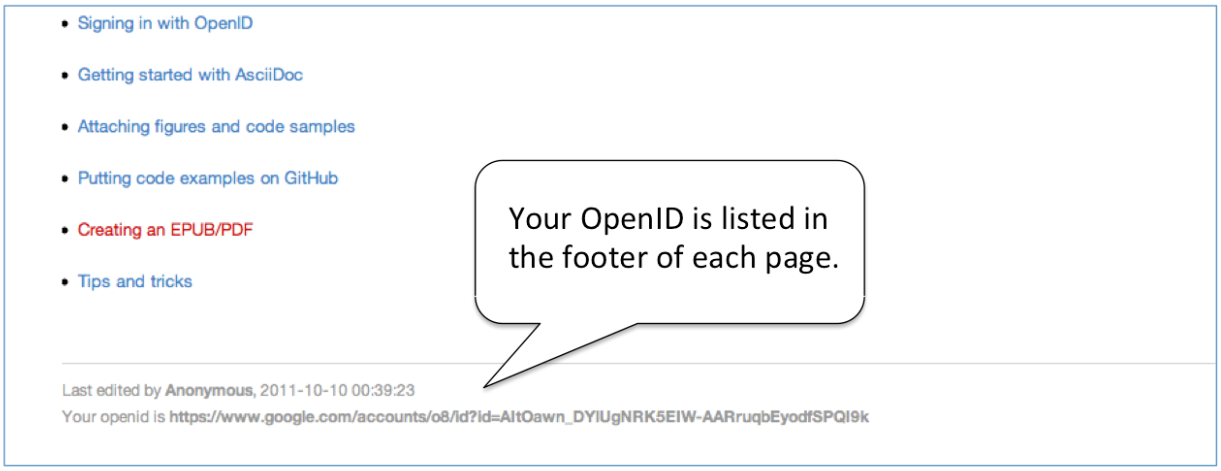
Accounts

Sign in to **Odewahn.makerpress.com** with your Google Account. Learn more

Sign in

Figure 2: The OpenID form redirects you to the login page for Google Accounts, where you can enter your Google+ login credentials.

Enter your username and password and then click "Sign In." Assuming you entered everything correctly, you'll see the "Home" page of your editing environment. You'll find your OpenID listed at the page footer, as shown in Figure 3.



- Signing in with OpenID
- Getting started with AsciiDoc
- Attaching figures and code samples
- Putting code examples on GitHub
- Creating an EPUB/PDF
- Tips and tricks

Last edited by **Anonymous**, 2011-10-10 00:39:23
Your openid is **https://www.google.com/accounts/o8/id?id=AItOawn_DYlUgNRK5EIW-AARruqbEyodfSPQI9k**

Figure 3: The OpenID appears at the bottom of every page..

Once you're authenticated, you can start editing your project.

3 AsciiDoc Quickstart

AsciiDoc is a text document format for writing (among other things) books, ebooks, and documentation. The main advantages of AsciiDoc are that it is easy to use and plays well with O'Reilly's publishing process. It's similar to wiki markup — if you can write a Wikipedia article, then you're pretty much 90% of the way there. This Asciidoc cheat sheet covers a lot of the nitty-gritty, but the following sections will give you an overview of the markup you'll use most frequently.

Note
You can get the complete nitty-gritty on how O'Reilly uses AsciiDoc from our official AsciiDoc guidelines. You'll need to enter "guest" as the username, leave the password blank, and then navigate to the "pdf" directory to download the docs. In addition, the sample chapter with markup is particularly helpful in explaining how things work.

You can create and edit AsciiDoc in any text editor, and then paste it into the wiki to push it into our system. If you're on Windows, you can use Notepad (or anything you want, really). If you're on a Mac, you can use TextEdit, TextMate, or any of a number of choices. The important thing is that you use the AsciiDoc markup.

Here are some links to AsciiDoc books that you can use for reference:

- Codebox: Adventures with Processing
- MintDuino Project Book
- Best of Radar: Data

3.1 Inline Elements

Here's some some *italic* and some `monospaced` (aka "constant-width" or "CW") text:

```
Here's some some _italic_ and some +monospaced+ (aka "constant-width" or "CW") text:
```

Backticks can also be used for literal (CW) text, for example: `ls -al`:

```
Backticks can also be used for literal (CW) text, for example: `ls -al`:
```

Warning
Using inlines in AsciiDoc can be tricky.
Delimiters may not be interpreted as intended if they don't abut whitespace on both sides; the fix for this is to double them up, as explained under "Constrained and Unconstrained Quotes" in the AsciiDoc User Guide. For example, compare how these render in the PDF: *+foo+ bar* vs. *`foo` bar*

3.2 Hyperlinks

To add a hyperlink, just type the URL followed by the description in brackets. Do not put a space between the brackets and the URL. Here's an example:

```
http://oreilly.com/[O'Reilly Media Website]
http://www.makezine.com/[Makezine]
https://github.com/odewahn/codebox2/blob/master/code/fractal_barnsley.pde[Fractal Example  ↩
    in Processing]
```

3.3 Notes, Warnings, and Sidebars

If you need to add a note or warning, here's the format:

```
[NOTE]
===============================
O'Reilly Animal books traditionally make no distinction between the
DocBook +<note>+, +<tip>+, and +<important>+ elements.
===============================

.Add a Title
[WARNING] .Add a Title
===============================
O'Reilly Animal books traditionally make no distinction between the
DocBook +<warning>+ and +<caution>+ elements.
===============================
```

Here's a Sidebar:

```
.Titled vs. Untitled Blocks
****
O'Reilly house style generally uses titles only on formal blocks
(figures, tables, examples, and sidebars in particular). Although
AsciiDoc supports optional titles on many other blocks, these
generally are not appropriate for O'Reilly books.

Conversely, _omitting_ a title from a sidebar is generally not
conformant to O'Reilly style. If you have questions about whether
content belongs in an sidebar vs. admonition vs. quote or other
element, please consult with your editor.
****
```

3.4 Code

Enclose code samples inside 4 consecutive minus signs ("-"), like this:

```
----
    if (x < X_MIN) {
       X_MIN = x;
    }
    if (x > X_MAX) {
      X_MAX = x;
    }
    if (y < Y_MIN) {
      Y_MIN = y;
    }
    if (y > Y_MAX) {
      Y_MAX = y;
    }
----
```

You can also use the AsciiDoc "include" macro to pull in code files:

```
----
include::code/example.c[]
----
```

3.5 Bullet lists

Use asterisks ("*") to create bullets. ou can indent items by using multiple asterisks:

```
* Lorem ipsum dolor sit amet, consectetur adipiscing elit.
* Nulla blandit eros eget velit bibendum placerat.
** Pellentesque id justo ultrices est pharetra suscipit.
** Cras nec magna a lectus consequat varius.
* Phasellus tempor lacinia neque, et scelerisque lectus luctus id.
```

The list will look like this:

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Nulla blandit eros eget velit bibendum placerat.
  - Pellentesque id justo ultrices est pharetra suscipit.
  - Cras nec magna a lectus consequat varius.
- Phasellus tempor lacinia neque, et scelerisque lectus luctus id.

3.6 Numbered lists

Use periods (".") to create a ordered (i.e., "1, 2, 3, . . . ") lists:

```
. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
.. Nulla blandit eros eget velit bibendum placerat.
.. Pellentesque id justo ultrices est pharetra suscipit.
. Cras nec magna a lectus consequat varius.
. Phasellus tempor lacinia neque, et scelerisque lectus luctus id.
```

Here’s how it will look:

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   a. Nulla blandit eros eget velit bibendum placerat.
   b. Pellentesque id justo ultrices est pharetra suscipit.
2. Cras nec magna a lectus consequat varius.
3. Phasellus tempor lacinia neque, et scelerisque lectus luctus id.

3.7 Simple Tables

Here’s the basic format for creating tables:

```
.An example table
[width="40%",options="header"]
|=============
|col 1| col 2| col3
|1  | 2 | 3
|4  | 5 | 6
|7  | 8  | 9
|=============
```

It will look like this:

Table 1: An example table

col 1	col 2	col3
1	2	3
4	5	6
7	8	9

4 Attaching figures and code samples

To include figures and code samples in your project, you first upload the file to the project by clicking the "Upload Attachment" button at the upper right-hand corner of the screen. This will place the file in a directory called *attachments*. Then, you create a link to the file you just uploaded, like this:

For figures. Use AsciiDoc's *image* macro to create a reference to the figure It will look something like this:

```
image::attachments/figure1.png[scaledwidth=90%]
```

Figures can "float" throughout the document and may not be on the same page you expect them to be. For this reason, saying things like "This figure. . . " or "The figure below. . . " might confuse the reader, since the figure may very well be on a totally separate page once the document is rendered. For this reason, all figures must have an accompanying cross-reference.

For Code Sample. To include code sample, use AsciiDoc's *include* macro to pull in the listing. Be sure to embed it within the **code** delimiters. Here's an example:

```
----
include ::attachments/my_program.py[]
----
```

Note that in general, it's better to link out to listing files by linking them directly to GitHub. This is explained in another section.

5 Including Code Samples on GitHub

We recommend that you place all code samples on GitHub. This makes it easy for you, and more importantly, your readers, to pull down the code and fix it. Take a look at Matthew Russell's Mining the Social Web to see how well this works when done consistently.

Matthew's included a lot of niceties on the landing page, like marketing text, praise quotes, JPEG of the front cover, a plea not to steal the book, etc. We really like the way he's set up the main page, but we're certainly not mandating authors include all this material, or that they adhere to any formal strictures. If you prefer to do something much more bare-bones, that's totally fine, too. See the Jonathan LeBlanc's Programming Social Applications for something far more simple.

Note that Jonathan's actually organized his code in directories by chapter, which we think is a good idea if there are a lot of code examples and/or chapters in the book. It makes it easier for readers to find things. Matthew Russell's solution for enhancing findability/navigability was to add an Examples Listing to his GitHub wiki.

In terms of linking to GitHub content from the book, we'd recommend you do the following:

5.1 Add preface text

Add the following text to the "Using Code Examples" section of the Preface:

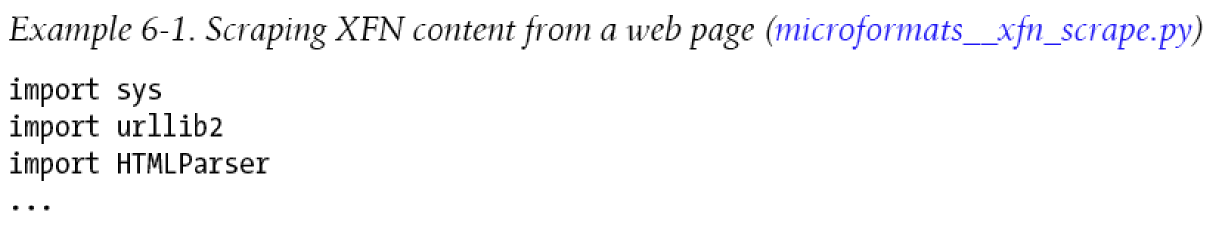
```
The code examples in the following chapters are available for download at GitHub at
https://github.com/<username>/<booktitle>/ -- the official code repository for this
book. You are encouraged to  monitor this repository for the latest bug-fixed code as
well as extended examples by the author and the rest of
the social coding community."
```

5.2 Link out to your examples in the text

For every code listing that's in both GitHub and the book, add a hyperlink in the book to the corresponding code in GitHub. If the code is in a formal example, we recommend putting the link in the Example title (otherwise, you may want to add the link to the preceding body text). Here's an example of the AsciiDoc markup you can use (note the backslash after "microformats" in the URL, which is necessary to escape the double underscore so it is properly translated to PDF):

```
[[EXTEST]]
.Scraping XFN content from a web page
 (https://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/microforma
====
----
import sys
import urllib2
import HTMLParser
...
----
====
```

Note that for the hyperlink node text, we've used just the filename of the code, so that the full URL isn't displayed in the Web PDF. The elemnts will appear as in Figure 4.



Example 6-1. Scraping XFN content from a web page (microformats__xfn_scrape.py)

```
import sys
import urllib2
import HTMLParser
...
```

Figure 4: Here's how references to your code on GitHub will appear in text

6 Building your project

To produce an EPUB or PDF, you must create a page that assembles the individual pieces of content into a final document. You can think of this page as a sort of master header file that lists all the other elements that should be included in the final document. These files should:

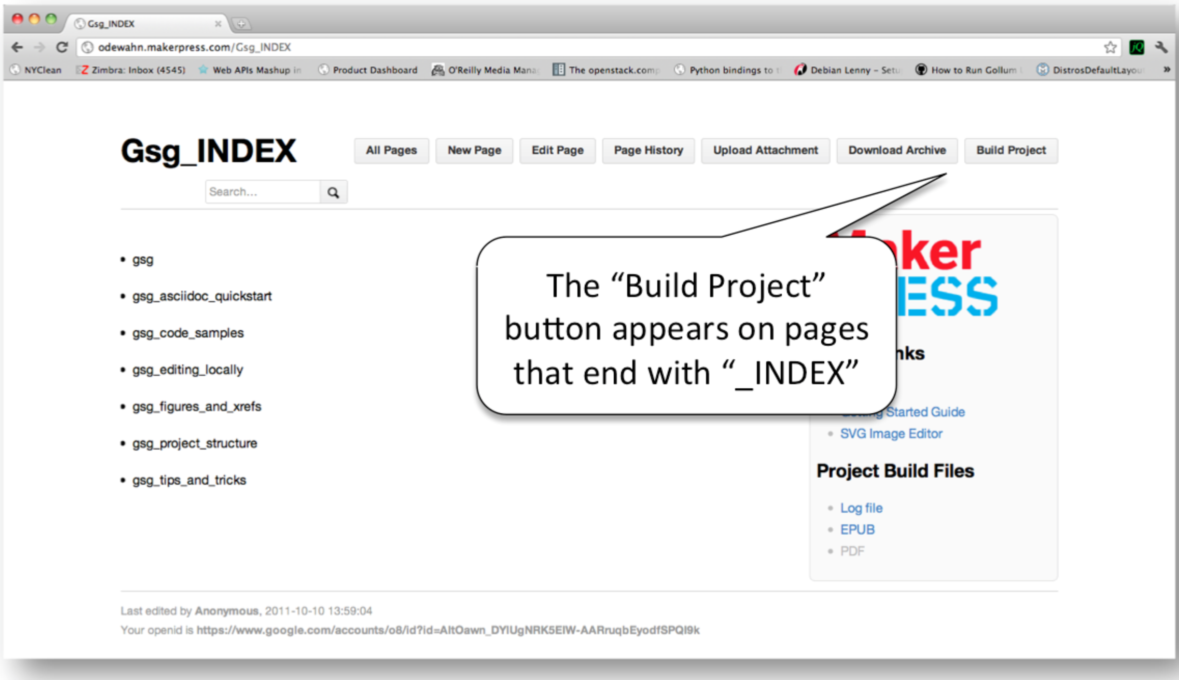
- Have a name that ends with the string "_INDEX". This tells the system that it's a special page used to build a final document.
- Contain a bulleted list of files to include in the order in which they are to appear. You reference each of these by page name, which you can find by clicking the "All Pages" button. Note that you do not have to have an "include::" at the beginning, or a ".asciidoc[]" at the end. These will be added automatically.

Note
The "All pages" button is a quick way to get list of all the pages in your project. You can use cut-and-paste to quickly create an _INDEX file, and then put them in the correct order.

Here, for example, are the contents of gsg_INDEX, which lists all the pages in this guide:

```
* gsg
* gsg_asciidoc_quickstart
* gsg_code_samples
* gsg_editing_locally
* gsg_figures_and_xrefs
* gsg_project_structure
* gsg_tips_and_tricks
```

Once you create an "_INDEX" file like this, you'll see a button called "Build Project" appear at the top right hand corner of the screen, as shown in Figure 5.



- gsg
- gsg_asciidoc_quickstart
- gsg_code_samples
- gsg_editing_locally
- gsg_figures_and_xrefs
- gsg_project_structure
- gsg_tips_and_tricks

Last edited by **Anonymous**, 2011-10-10 13:59:04
Your openid is **https://www.google.com/accounts/o8/id?id=AltOawn_DYIUgNRK5EIW-AARruqbEyodfSPQI9k**

Figure 5: The "Build Project" button appears on _INDEX pages.

When you click the button, a new window open that will show you the status of the build as it runs. Assuming all goes well, you'll receive an indicator that the build succeeded. You can then use the links in the right-sidebar to view the final output from the build.

If there is an error in your markup, you'll receive a link to a log that will help you "debug" your project. This is described in the next section.

7 Debugging your project

- Why debugging / we need valid DocBook
- Log helps you identify problems
- Common errors and their solution
- If all else fails

8 Downloading your archive

- Allows you to pull files to make a backup copy. No other backups beyond what you do, although we're working on this.
- Just click the "Download Archive" button
- No version history (Need to revisit this. Maybe it should just zip the entire git dir?)

9 Editing locally (without the wiki)

One of the cool things about the Gollum wiki is that it's basically just a plain interface on top of a git repository. This allows a lot of flexibility in using the tool. For example, suppose you have a long flight and want to edit on the plane. No problem — just pull down your changes, do your edits, and then push them back up when you land. Or, suppose you'd rather just skip the wiki interface entirely and just do everything locally. That's fine to — you can just do your edits, but you can push your project to the editing environment to share it with tech reviewers or access O'Reilly's build systems (more on that in a bit).

There are a couple of things you'll need to be able to edit locally:

- A copy of git installed on your local machine. You can get git at [link to git]
- Your public key. This should have been installed when the repo was set up, but if not, contact [???]. (Someday we hope to have a UI where you can do this yourself, but that's for the future.)

Once you've git this set up, you're ready to use git. There are a few caveats. The first is that Gollum can only work on the master branch, so any changes you want to share will need to be committed to the master and pushed up. The second is that is anyone else makes changes on the repo in the interim, you'll need to resolve any conflicts to merge their changes; git will warn you about this when you try to push up the new repo. Describing conflict resolution is beyond the scope of this document, but Scontt Chacon's Pro git book is an outstanding resource.

Note

- sudo -Hu root ssh-keygen -t rsa
- ssh-keygen -t rsa
- sudo cat /root/.ssh/id_rsa.pub

This will look something like this:

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/odewahn/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/odewahn/.ssh/id_rsa.
Your public key has been saved in /home/odewahn/.ssh/id_rsa.pub.
The key fingerprint is:
0b:d0:05:60:0a:5a:ed:95:92:07:b1:87:03:e5:2c:75 odewahn@oreilly.com
The key's randomart image is:
+--[ RSA 2048]----+
|. ooB=Eo.        |
|.o B+=+.         |
|. o.B+o          |
|   ..+           |
|      . S        |
|       . .       |
|        .        |
|                 |
|                 |
+-----------------+

$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA6HuZxHII1OFWDY5TIdGlNg0HHfaWwC/ ↩
    ClgHe2WgPezBF3UZPnXnJCUH/ecA3JRnvrcnFlJqKYlNcS/OZz8IguHh3AsNuwbBmDNXS/ ↩
    VB31LcV0gy8TovmEef20n+FO4xF6S4Zsm5Dbz8S/bOtrRxP+ ↩
    X0ujeYr3KOvs7qSq8g4ciVoXf20XJyicnJU2bFJqS5ngVecZd2h1TAvE6SgAEI2+00Bg1r/pbAaV/HNgszlvo ↩
    ++oaiGi88d1kfpdnCmemGYNLccprBgsE9etVYArOI6HiwyswbeRz/ ↩
    b5bgJR1gMwZq8hLMX1IQcMCaFfveG5EL1fYS2nmUl6/GXYTs0dkCeNw== odewahn@oreilly.com
```

9.1 Common use cases when working with git locally

The URI to access the repo on your remote editing environment is very close to the wiki URL, except that you don't include the "HTTP://". The user name is "git" and the repo name is "git_repo.git." So, if the URL for your editing environment is "http://test.makerpress.com," the URI for your repo will be "git@example.makerpress.com:git_repo.git". The remainder of this sections describes some of the common use cases for using git locally:

- If you're starting push a repo from your local machine to the remote editing environment, do this:

```
$ cd /the/local/directory
$ git add remote gollum git@example.makerpress.com
$ git push gollum master
```

This will push the repo up to your editing environment. This is the way a lot of people start the projects.

- To clone the repo from the editing environment, you can do this:

```
$ git clone git@example.makerpress.com:git_repo.git <local directory name>
```

This command will pull down the repo into a local folder.

- Add lots of images or code examples

The "Upload Attachment" feature is nice, but it can be a real drag if you want to add a bunch of images or code examples. Using git locally is one of the best ways to get a bunch of files added quickly. Al you have to do is bring down the repo and put the files into the "attachments" directory. For example:

```
$ git clone git@example.makerpress.com:git_repo.git add_images_and_code
$ cd add_images_and_code
$ cd attachments
$ cp /some/image/dir/*.jpg .
$ cp /some/code/dir/*.c .
$ cd ..
$ git add attachments/*.*
$ git commit -a -m"Added a bunch of images and code"
$ git push origin master
$ cd ..
$ rm -rf add_images_and_code
```

You can not reference all these files in your document, like this:

```
  image::attachments/new_img.jpg[]
  ...
  include::attachments/new_code.c[]
```

- Clean up file names

The native Gollum interface doesn't allow you to change file names or manipulate the directory structure for your repo. If you want to do this, you can just pull the repo down, make your changes in git, do your commit, and then push some files up. For example:

```
$ git clone git@example.makerpress.com:git_repo.git move_files
$ cd move_files
$ git mv crappy_name.asciidoc nice_name.asciidoc
$ git rm unused_file.asciidoc
$ git commit -a -m"Changed some file names"
$ git push origin master
$ cd ..
$ rm -rf move_files
```

- Manipulate your repo with scripts

Suppose you want to do something sort of complex, like performing a global search and replace or converting all the URLs in your document into bit.ly links. (Or whatever). To do this, you can simple clone the repo, execute your script (or whatever), commit your changes, and then push the repo back up. For example:

```
$ git clone git@example.makerpress.com:git_repo.git manipulate
$ cd manipulate
...
Run script to replace all URLs with equivalent bit.ly links
...
$ git commit -a -m"Changed all links to bit.ly links"
$ git push origin master
```

10 Tips and Tricks

- Write in a text editor and paste the content into the Gollum wiki
- Don’t put section headers inside your content sections — put them in the "Home" file
- Don’t use footnotes
- Don’t have an empty section
- Don’t start an xref with a number or character
- Don’t duplicate an xref name
- To generate a PDF from this repo, use this command: "a2x -fpdf --fop --no-xmllint README.asciidoc"