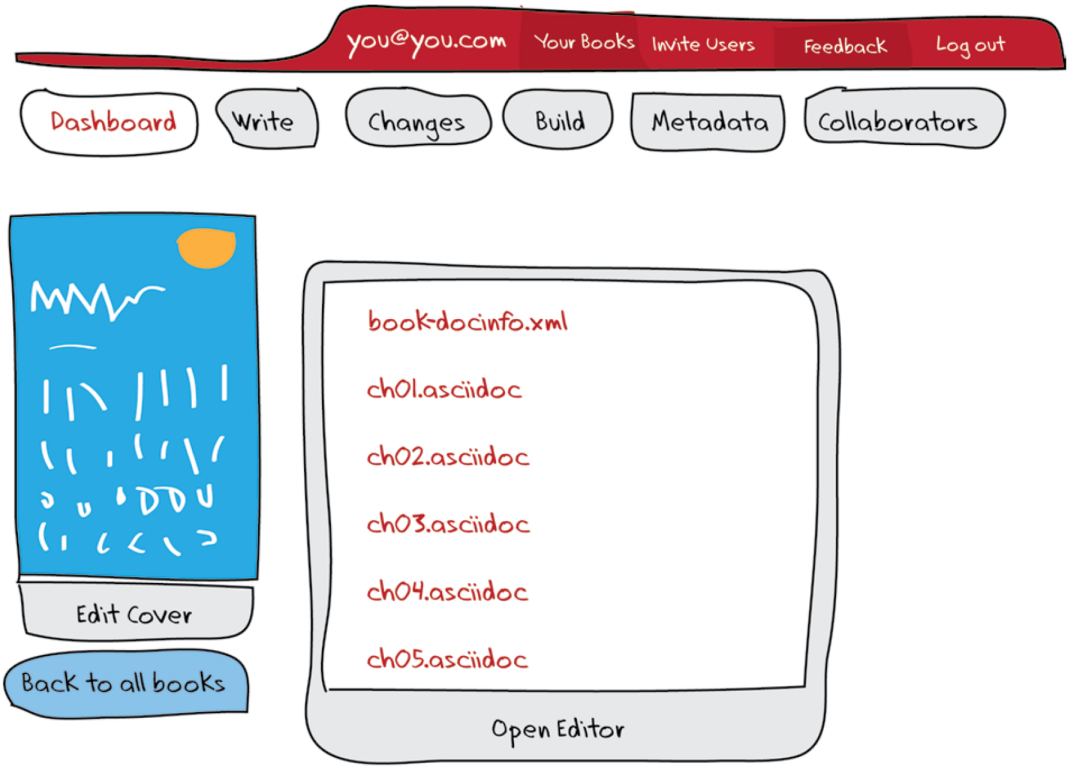
Getting Started with Atlas

O'Reilly Media, Inc.



O'REILLY®

Getting Started with Atlas

O'Reilly Media

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Getting Started with Atlas

by O'Reilly Media

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Tools Team

April 2012: First Edition

Revision History for the First Edition:

2012-05-01: First release

2012-05-22: Second release

2012-08-20: Third release

2012-10-01: Fourth release

2012-12-11: Fifth release

2013-01-16: Sixth release

2013-02-29: Seventh release

2013-04-10: Eighth release

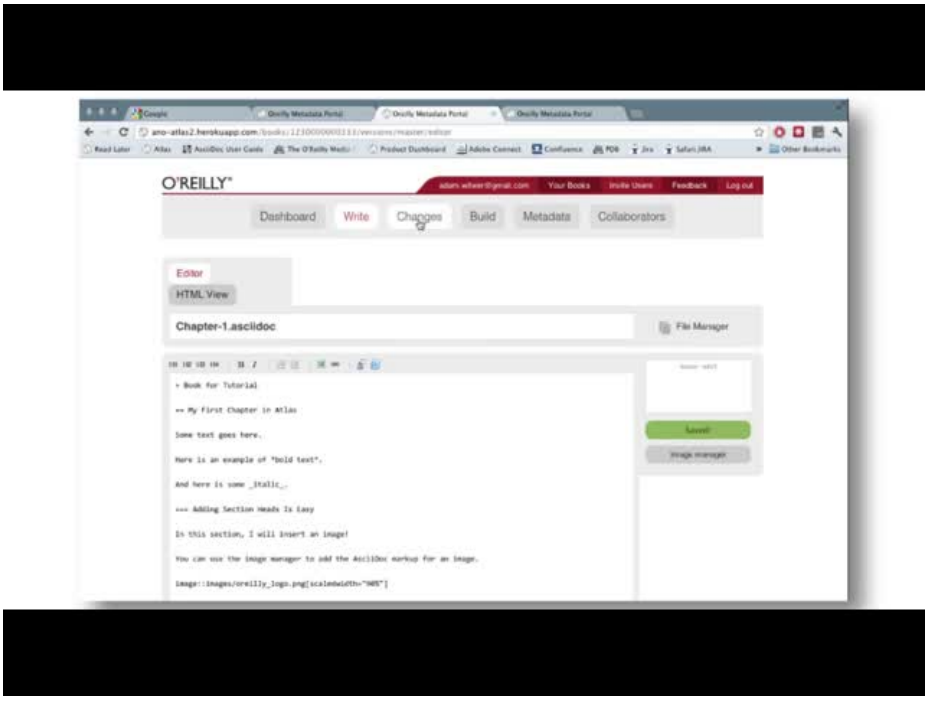
2013-05-29: Ninth release

See *http://oreilly.com/catalog/errata.csp?isbn=* for release details.

Table of Contents

CHAPTER 1
Join the Publishing Revolution

Thank you for your interest in Atlas! Atlas is a wiki-like, git-managed authoring platform for creating books. If you haven't already, you may want to check out the getting started video for a quick introduction to Atlas.



Some of the features of Atlas are as follows:

Simple markup
: Atlas supports AsciiDoc and, for simpler projects, Markdown.

Git backend
: If you have a book, you have a git repository and all of the power and convenience that comes with using git version control.

Easy book, ebook, and web builds
: Atlas lets you build your project in four formats at any time: Mobi (for the Kindle), EPUB (for most other ebook platforms), PDF (for print), and HTML (for the web). Atlas also gives you tools to debug your ebook formats so that you can identify and fix problems quickly.

An invitation system
: Add collaborators to your project at any time by simply sending an invite.

Flexible writing options
: You can write directly within Atlas, but if writing in a web browser is not for you, no problem. Just use git to clone your book to a local machine and write in the text editor of your choice.

How It Works

1. Receive an invite from an O'Reilly editor or another author and create an account.
2. Write in AsciiDoc, an open source text format developed by Stuart Rackham. Each time you save a file, it will be logged into a git repository, so it's always under version control.
3. When you're ready, build your book. Atlas interfaces with O'Reilly's publishing toolchain, generating PDF, EPUB, and Mobi files for you on the fly. Theebook files look just like those for sale on oreilly.com.

Why bother with this, you ask? We're trying to create an appealing authoring process that will get you writing as quickly as possible.

Start Writing Today

We're just getting started with Atlas, and you can play a role in determining future development by using the platform to write your book. If you're willing to use Atlas, we ask that you:

- Write in AsciiDoc, which is the preferred formats for Atlas.
- Provide feedback. Atlas is currently in beta testing, and we need to hear from you to take the platform to the next level. To submit feedback, simply click on the Feedback tab within Atlas and create an Issue. You will need a GitHub account to create

an Issue. If you're not able or willing to provide feedback, Atlas might not be right for you at this time.

If the above sounds good to you, then you're the perfect candidate to work in Atlas. We're eager to help you get started. Let's revolutionize publishing together!

CHAPTER 2

Write How You Want

When you are using Atlas to write your book, you have the option of writing in the wiki interface or locally on your own computer. This chapter covers each of those scenarios. As you write your book, you can jump back and forth between the two writing environments. As long as you are saving in the web interface or using git to `push` and `pull` from your local machine, you'll always be in sync.

Get Your Gravatar

You may notice that some users have personal icons in the Changes and Collaborators sections of Atlas. Those image icons are pulled in from Gravatar, based on the user's email address. Set up your Gravatar to get a more personalized icon.

Working in the Atlas Wiki

If you've used a wiki before, you will find the Atlas interface to be very familiar. In the following sections, you'll learn how to create and edit new files in the wiki interface.

Invite Some Collaborators

Once you are working on a project in Atlas, you can invite additional collaborators at any time. First, create your project. Then, click on the Collaborators tab. Enter the email address of anyone you'd like to collaborate with on your project. If you choose a permission type of Collaborator, the invited author will have full read and write access to your project, via the wiki interface and the git repository.

You can also remove collaborators using the same Collaborators tab.

Creating New Files

When you create your book in Atlas, it will have no files, as shown in Figure 2-1.

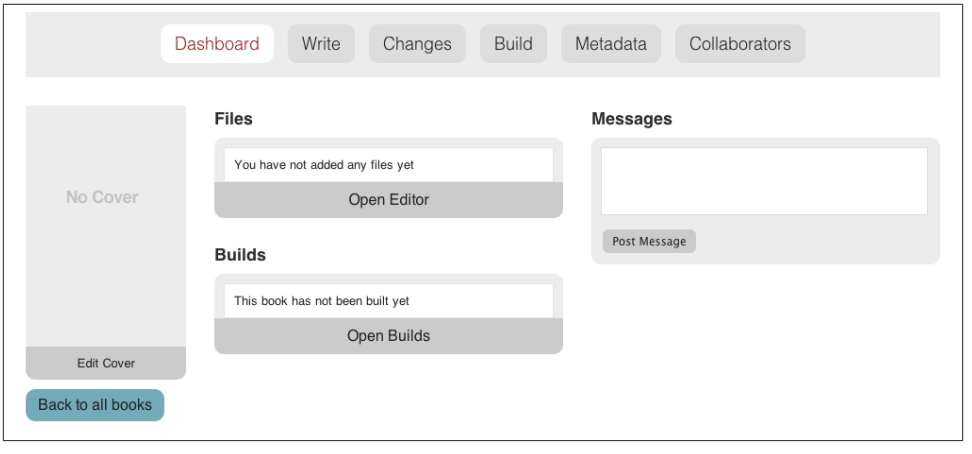


Figure 2-1. When you create a new book in Atlas, you start off with a blank slate

You can create your first file by clicking on Open Editor. Atlas will recognize that your book has no files, and you will be prompted to create one. Click "Create your first file" and name the file, as shown in Figure 2-2.

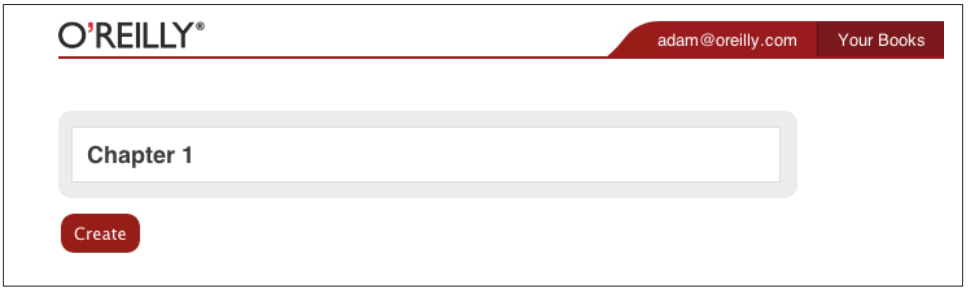


Figure 2-2. Create your first file

Once you've created the file, you're ready to start writing!

Working with AsciiDoc Text in Atlas

When you're working in the Atlas wiki interface, you will need to write in AsciiDoc.[1] See Chapter 4 for an introduction to AsciiDoc markup.

The Atlas wiki editor works just as you'd expect. Use the buttons to insert simple AsciiDoc markup. For example, Figure 2-3 shows what happens when you click the H1 button.

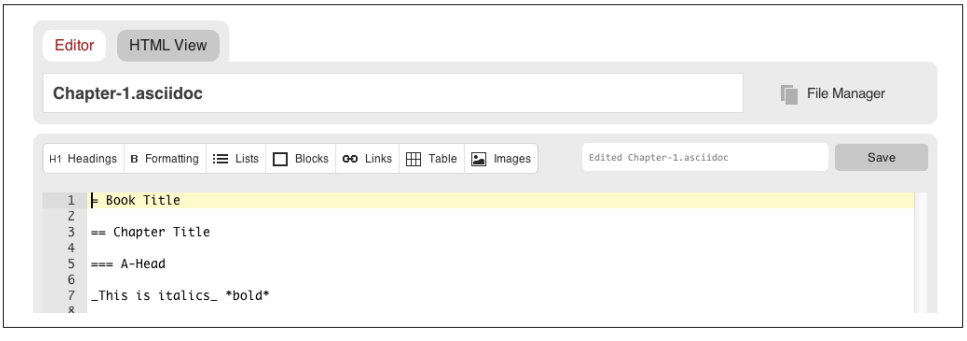


Figure 2-3. Use the Atlas wiki interface to write in your web browser

Once you've entered some text, click Save and your changes will be committed to a git repository that is created automatically each time you start a new book. Optionally, you can enter a log message before you click Save, and it will be included with the commit to the git repository, as show in Figure 2-4. Read more about making the most of your git repository in "Working Locally" on page 11 and Chapter 5.

1. Atlas supports Markdown for less technically complex text. Ask your editor if Markdown is a good fit for your project.

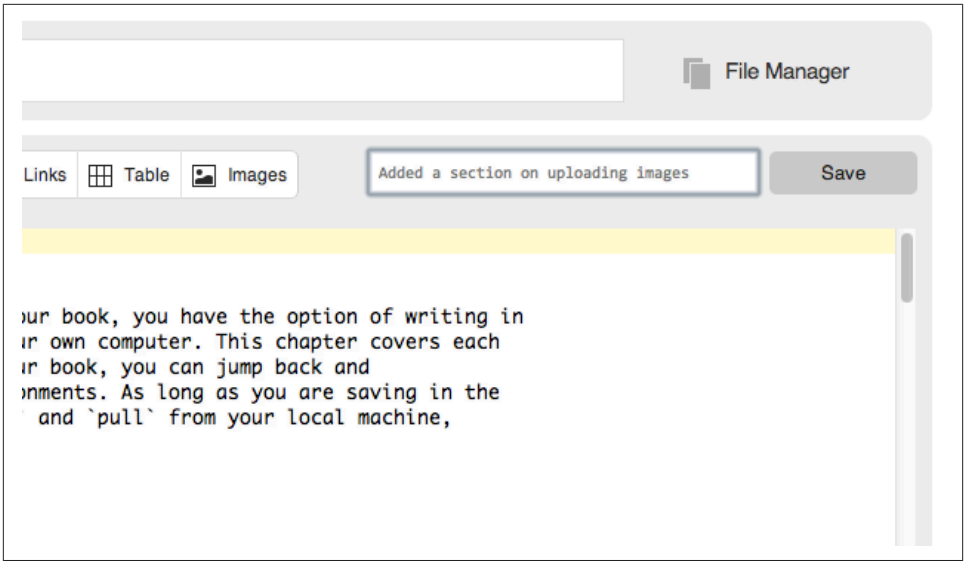


Figure 2-4. It's good practice to add a meaningful log message

Note also that you can use the File Manager drop-down to jump to other files or to create new ones.

Adding Images via the Atlas Interface

You can upload images to your book directly in the Atlas wiki interface. Click on Image manager and then select Choose File and finally Upload, as shown in Figure 2-5.

Figure 2-5. Upload your images within Atlas

Once you've uploaded your image, Atlas will generate a thumbnail preview of that image, which you can see at any time by bringing up the Image manager.

At this time, if you upload an image via git, the image will not have a thumbnail preview within the Image manager.

You can also use the Image manager to insert the AsciiDoc reference to that image:

1. Place the cursor where you'd like to insert the image within the text.
2. Click on Image manager.
3. Select the thumbnail preview of the image you've uploaded and click Insert (Figure 2-6).

Figure 2-6. You've uploaded the image; now reference it within your document

Once you click insert, Atlas will add AsciiDoc markup that looks something like the following to your document:

```
image::images/insert_image_asciidoc.png[]
```

Read more about figure and image markup in "Figures and Other Images" on page 29 in Chapter 4.

Tracking Changes in Atlas

Each time you save a change in the wiki interface or use git to push a change to your repository (as described in "Committing and Pushing" on page 12), Atlas displays that change on the Changes tab.

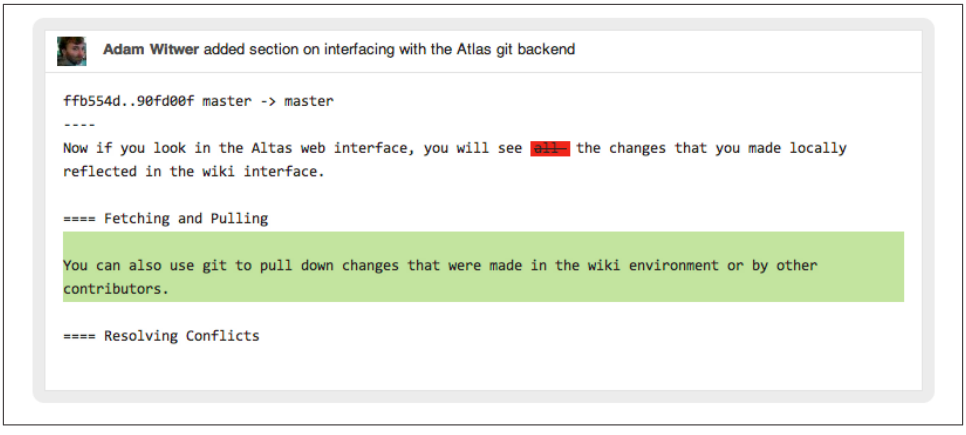


Figure 2-7. An Atlas change box shows additions and deletions made to the text

Figure 2-7 shows a change that was saved to the repository via a `git push`. In this change notification box, Atlas displays the author who made the change along with the log message used during `git commit`. If the author were to use the Save button on the wiki interface (instead of git), as explained in "Working with AsciiDoc Text in Atlas" on page 7, the change would be captured and displayed in the same way.

Working Locally

Atlas sits on top of a git repository, giving you the flexibility to write how and when you want. For example, suppose you have a long flight and want to write edit on the plane. No problem—just pull down your book files, make your changes locally, and then push them back up when you're connected to the Internet again. Or perhaps you'd rather skip the wiki interface entirely and work on your local machine exclusively. That's fine too.

Cloning Your Book

In order to work locally, you will need to have git installed on your machine. The following section describes a few basic git commands to get you up and running, but see Chapter 5 for more details around using git with Atlas.

Once you have git installed, click on the Admin tab in the Atlas interface to retrieve your git repository URL, as shown in Figure 2-8.

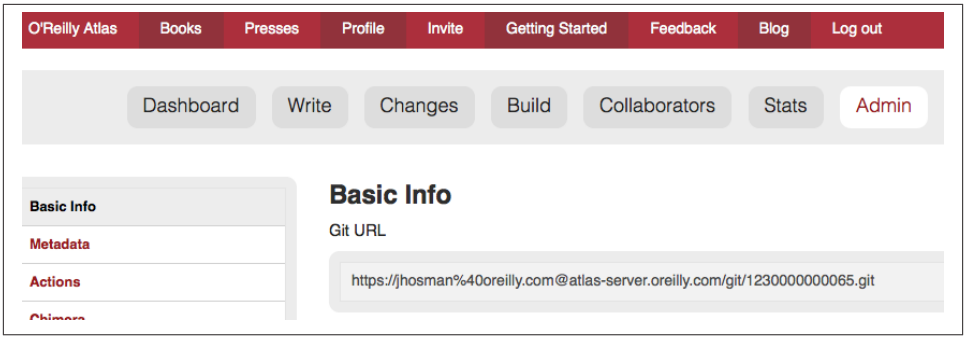


Figure 2-8. Grab your git repository URL from the Admin tab

Now that you have the URL, you can use git to clone it to your computer. Use the `git clone` command and enter your Atlas password. When you clone the repo, you can optionally add the name of the directory, like so:

```
$ git clone https://jhosman%40oreilly.com@atlas-server.oreilly.com/git/
1230000000065.git
   getting_started_with_atlas/

Cloning into getting_started_with_atlas...
remote: Counting objects: 338, done.
remote: Compressing objects: 100% (337/337), done.
remote: Total 338 (delta 136), reused 0 (delta 0)
Receiving objects: 100% (338/338), 4.10 MiB | 534 KiB/s, done.
Resolving deltas: 100% (136/136), done.
```

The `clone` command will download all of the files into a directory named *getting_started_with_atlas*, and that directory is now under version control with git.

All of the examples in this guide use the command line git client. If the command line is not for you, there are several GUI git clients available for Windows, OS X, and Linux.

Committing and Pushing

Now that you've got a local checkout of your project, you can open and edit the *.asciidoc* file. As explained in Chapter 4, AsciiDoc is a text-based markup language. You can use any text editor to edit the files. Figure 2-9 shows edits being made to this chapter in TextMate, a text editor for the Mac.

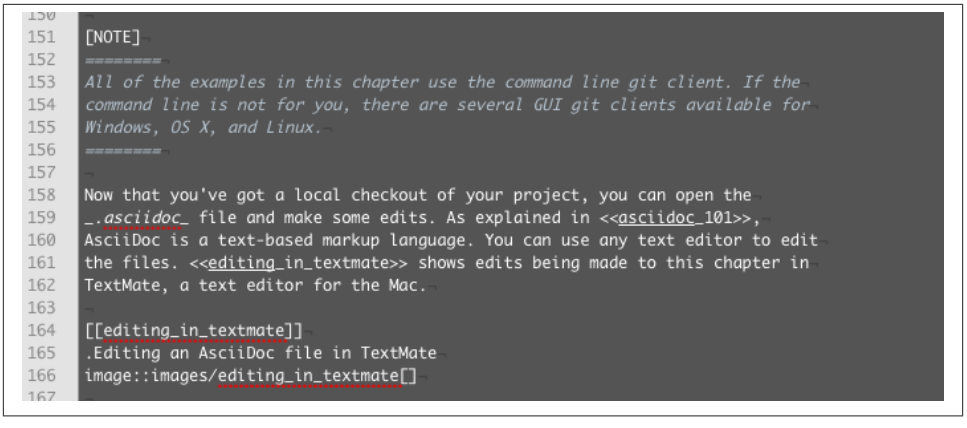


Figure 2-9. Editing an AsciiDoc file in TextMate

Now it's time to commit the changes to the git repo. You can include a log message with `-m`. The `-a` means to include all changes.

```
$ git commit -a -m'added section on interfacing with the Atlas git backend'
[master 0e487ee] added section on interfacing with the Atlas git backend
 3 files changed, 46 insertions(+), 6 deletions(-)
 create mode 100644 images/editing_in_textmate.png
```

Finally, `push` your committed changes:

```
$ git push origin
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 54.03 KiB, done.
Total 7 (delta 4), reused 0 (delta 0)
To https://adam%40oreilly.com@atlas-admin.oreilly.com/git/1230000000065.git
   ffb554d..90fd00f  master -> master
```

Now if you look in the Altas web interface, you will see the changes that you made locally reflected in the wiki interface.

Fetching and Pulling

You can also use git to pull down changes that were made in the wiki environment or by other contributors. There are two ways of downloading changes. One way is to use `fetch` followed by `merge`, as in this example:

```
$ git fetch
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
```

```
From https://atlas-admin.oreilly.com/git/1230000000065
   cd86112..cba41ff  master     -> origin/master
```

`fetch` downloads the changes. Now use `merge` to bring your local files up to date:

```
$ git merge origin
Updating cd86112..cba41ff
Fast-forward
 ch02.asciidoc |    3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)
```

Alternatively, you can use `pull`, which downloads the changes and merges them in with a single command:

```
$ git pull
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 4), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From https://atlas-admin.oreilly.com/git/1230000000065
   cba41ff..a972d49  master     -> origin/master
Updating cba41ff..a972d49
Fast-forward
 ch02.asciidoc |   25 +++++++++++++++++++++++++
 1 files changed, 25 insertions(+), 0 deletions(-)
```

Using `push` and `pull` to interface with your Atlas repo is the just the beginning of what you can do with git. Check out Git Reference andChapter 5 to learn what else is possible.

Resolving Conflicts

When you use `git merge` or `git pull`, git will attempt to combine all changes into one document. Sometimes, however, git will fail to combine the text and your AsciiDoc file will have a conflict. This situation may arise if, for example, two authors try to `push` changes to the same line of text. Conflict resolution is beyond the scope of this document, but the Git User's Manual has an excellent overview of git conflicts and how to resolve them.

CHAPTER 3

Building and Debugging

You can use Atlas to build your book in three formats—PDF, EPUB, KF8, and HTML—whenever you like.

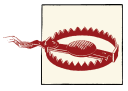
Three Major Ebook Formats

You may be wondering about the three book formats that Altas produces. An overview of each follows:

- The fixed-layout PDF format is undoubtedly familiar to you and remains the most popular format among customers who buy direct from O'Reilly. If your book ends up being printed, we will use a slightly modified version of the online-friendly PDF that Atlas outputs.
- EPUB uses a fluid layout, which means that the text reflows to fit the container in which it is presented. EPUB is sold by most major ebook retailers, with the exception of Amazon. You can read an EPUB with iBooks on an iOS device, Aldiko on an Android device, and Adobe Digital Editions on your desktop.
- The KF8 format is also a fluid format and is used by Amazon's Kindle platform. You can copy the KF8 that Atlas outputs to your Kindle, or you can preview it with the Kindle Previewer.

Building Books

To kick off a build, click on the Build tab. Then, select New Build, and you will be presented with a list of the files that you've created so far. Drag or click the files that you'd like to include in the build and select the ebook formats that you'd like generate, as shown in Figure 3-1.



A few things to keep in mind when building books in Atlas:

- Atlas expects a file extension of *.asciidoc*, so please use that instead of *.asc*.
- When you trigger a build, Atlas automatically creates a *book.asciidoc* file that references the files that you choose to include in the build, so there is no need to add one.
- Do not include *book-docinfo.xml* file in the build. This metadata-only file is described in "Adding Metadata" on page 17.

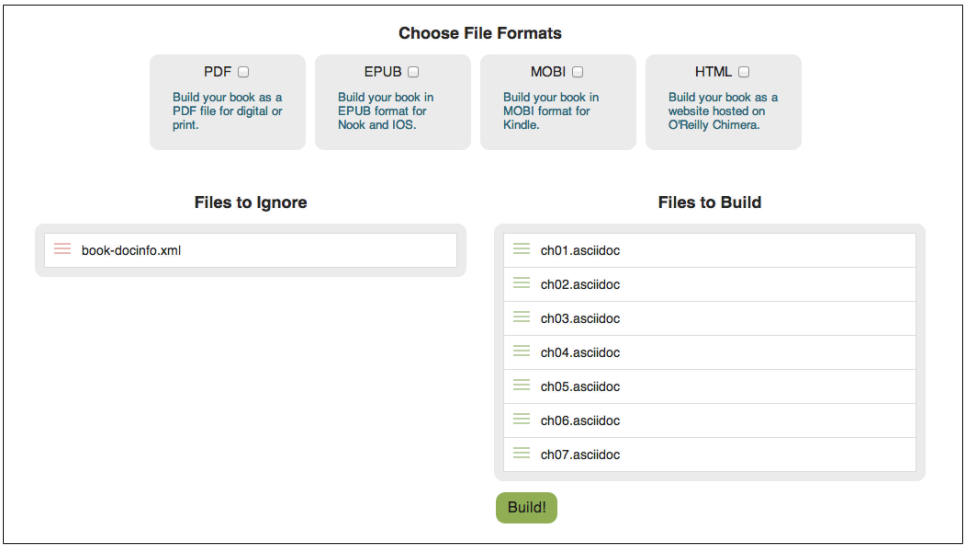


Figure 3-1. Select your ebook formats and files

The builds will change to a status of pending. Wait a few minutes, and then refresh the page. Assuming the builds were successful, you can download your files right from Atlas. If you run into an error, see "Debugging Errors" on page 18.

Build times vary based on the length of your book, but most builds finish within a few minutes. EPUB files build the fastest, PDF files the slowest, and KF8 files somewhere in between.

Adding Metadata

If the title and copyright pages in your book builds are looking bare, you can optionally fill them in by adding a file called *book-docinfo.xml* to your project via git (see Chapter 5), and then adding the appropriate metadata using XML tags. Here is an example showing the metadata that you might like to include:

```
<title>BOOK TITLE</title>

<author> <!-- feel free to add multiple authors-->
  <firstname></firstname>
  <surname></surname>
</author>

<editor>
  <firstname></firstname>
  <surname></surname>
</editor>

<copyright>
  <year>2012</year>
  <holder></holder>
</copyright>

<othercredit role="proofreader">
  <firstname></firstname>
  <surname></surname>
</othercredit>

<!-- All rights reserved. -->

<publisher>
  <publishername>O'Reilly Media, Inc.</publishername>
  <address format="linespecific">
    <street>1005 Gravenstein Highway North</street>
    <city>Sebastopol</city>
    <state>CA</state>
    <postcode>95472</postcode>
  </address>
</publisher>

<isbn></isbn>

<edition></edition>

<revhistory>
  <revision>
    <date>2012-02-10</date>
     <revremark>First release</revremark>
  </revision>
  <revision>
```

```
    <date>2012-02-27</date>
    <revremark>Second release</revremark>
  </revision>
  <revision>
    <date>2012-04-27</date>
    <revremark>Third release</revremark>
  </revision>
</revhistory>
```

When you create a new build, the metadata will be added to the title and copyright pages of your PDF. Note that you do not need to include the *book-docinfo.xml* file iself in the build. Atlas sees and picks up the metadata information automatically.

Debugging Errors

Book builds will sometimes fail. The most likely cause of a failure is that your AsciiDoc isn't structured in a syntactically valid way. When you trigger a build, Atlas transforms your AsciiDoc into DocBook XML 4.5 and, using the DocBook as a source, builds the book files. If one of your builds fails, Atlas will report the error, which should help you to fix it.

For example, here is an Atlas error log for a PDF build that failed:

```
book.xml:291: element xref: validity error : IDREF attribute
linkend references an unknown ID "cloning_to_github"
```

In this case, the problem is that there is a cross-reference referring to an ID that does not exist. You can fix it by adding an ID to the appropriate block, as explained in Chapter 4.

We realize that build failures can be tricky to troubleshoot, so we're working on improving the error messaging. Also, we've configured Atlas to be permissive enough so that it doesn't fail on minor errors. Please let us know about your experiences with troubleshooting build failures, and how Atlas can be improved to make the experience better.

CHAPTER 4

AsciiDoc 101

AsciiDoc is a text document format for writing (among other things) books, ebooks, and documentation. It's similar to wiki markup—if you can write a Wikipedia article, then you'll have no problem with AsciiDoc. The main advantages of AsciiDoc are that it is easy to use and plays well with O'Reilly's publishing tools, including Atlas.

Use the Source, Luke

If you're viewing this document as a PDF, EPUB, or KF8, you're seeing the *output* from the AsciiDoc source, rather than the AsciiDoc markup itself. Since this chapter is intended as an introduction to the markup, it may be helpful to view the source, available at cdn.oreilly.com/atlas_docs.zip.

How Do I Get Started Writing?

There are two ways you can write with AsciiDoc in Atlas:

1. **Write directly in the Atlas web interface.** Simply click on the "Write" tab in Atlas. Select an existing chapter file to edit, or choose "New File" to create one.
2. **Write in a text editor and add it to Atlas.** You can add text to Atlas by pasting the text directly into the writing interface or by using git (see Chapter 5) to push your local `.asciidoc` files to the Atlas repository.

Text Editors

Looking for a text editor to use? If you're on Windows, you could try Notepad++, or any other text editor. If you're on a Mac, TextEdit, TextMate paired with an AsciiDoc

bundle), or TextWrangler, are great choices. Or maybe you prefer old school vi or Emacs. That's okay, too.

Notes to Production

To leave a note to the Production team, please use a passthrough block with the Docbook `<remark>` element, like this:

```
++++
<remark>Use a passthrough block like this for notes to production staff</remark>
++++
```

AsciiDoc does have specific markup for comments (two slash characters, like this: //, followed by the comment text), but they do not get passed to the DocBook output by default. In order to make sure the Production team sees your comments, please use the DocBook passthrough method listed above. (See "Passthroughs" on page 42 for more info on DocBook passthroughs.)

AsciiDoc Markup Reference

This section illustrates some of the most common elements used in writing technical documentation. The AsciiDoc User Guide and the AsciiDoc cheat sheet are other helpful references, but the following sections will give you an overview of the markup you'll use most frequently, plus some customized markup that is specific to Atlas.

If you can't find what you're looking for here or if you just need a quick answer, please don't hesitate to email our support team at *toolsreq@oreilly.com*.

Text

Regular paragraph text does not need any special markup in AsciiDoc. Just add a blank line both above and below each paragraph, and the first word in the paragraph should not have a space before it. Here are some example paragraphs in AsciiDoc:

```
This is an example paragraph written in AsciiDoc. See, it's just plain text; no
special markup necessary! Do make sure there aren't spaces or manual indenta-
tions at the beginning of your paragraph text.

This is a second example paragraph in AsciiDoc. Note that there's a line break
and a blank line between paragraphs.
```

To learn how to add inline markup such as italics to your text see the "Inline Markup" on page 25 section below.

What's Going On, Anyway?

A general understanding of what is going on under the hood of Atlas will help you make the most of the system. When you build your book (as described in Chapter 3), Atlas first converts the AsciiDoc to DocBook XML and then generates the book formats from that DocBook. For each AsciiDoc markup, there is an equivalent DocBook element. As long as the AsciiDoc markup uses the expected syntax, the conversion to DocBook (and then to the book formats) goes smoothly.

If you get a build error, the most likely cause is an AsciiDoc markup error. Atlas provides error message logs to help you troubleshoot and fix syntax errors. Read more about building and debugging in Chapter 3.

Chapters

The top of each chapter file should begin with a chapter title preceded by two equals signs. It's good practice to always include a unique ID string above the chapter title, surrounded in double brackets, for example:

```
[[unique_chapter_id]]
== Chapter Title

Chapter text begins here.
```

Headings

There are multiple ways to mark up heading levels in AsciiDoc. We've chosen our favorite "delimited" style to show you below, but feel free to use the two-line underlined heading style if you prefer.

Note that the levels described in the AsciiDoc User Guide can be confusing: in AsciiDoc, the document (book) is considered level 0; generally a chapter will be at AsciiDoc level 1 (unless you're dividing the book into parts), and sections within chapters start at AsciiDoc level 3 (which is equivalent to DocBook `<sect1>`).

Top-level heading

Within a chapter, the first and highest heading level uses three equals signs:

```
=== Top-Level Heading
```

Second-level heading

This heading level should only follow a top-level heading:

```
==== Second-Level Heading
```

Third-level heading

This heading level should follow a second-level heading only:

```
===== Third-level heading
```

Parts

If you want to group your chapters into parts, navigate to the file that should be the first chapter in that part, and add the part markup above the chapter title. It's good practice to always include a unique ID string above the part title, like so:

```
[[unique_part_id]]
= Part Title

[[unique_chapter_id]]
== Chapter Title

Chapter text begins here.
```

To add introductory text to a part, add this `partintro` markup after the part title, but before the chapter title:

```
[[unique_part_id]]
= Part Title

[partintro]
--
Insert introductory text here.
--

[[unique_chapter_id]]
== Chapter Title

Chapter text begins here.
```

Prefaces

A preface file should begin with the word *preface* in single brackets, followed on the next line by two equals signs and the preface title:

```
[preface]
== Preface Title

Preface text begins here.
```

Forewords

Forewords are treated as prefaces, except they have a title of *Foreword*. For example:

```
[preface]
== Foreword

Foreword text begins here.
```

Afterwords

The markup for an Afterword is similar to the preface markup, but it has an additional role attribute with a value of "afterword". Here's the markup:

```
[preface]
[role="afterword"]
== Afterword

Afterword text begins here.
```

Dedications

A dedication file should begin with the word *dedication* in single brackets, followed by a blank line and then the dedication title and text. The title must be present, but it will not render in the output:

```
[dedication]
== Dedication

This book is dedicated to my cat, Garfield.
```

Dedication pages render on their own page at the beginning of the book, before the table of contents.

Praise Pages

Praise pages are marked up similarly to "Dedications" on page 23, but you will add a special `role="praise"` attribute. For each item of praise, use the quote markup (discussed in "Quotes" on page 29):

```
["dedication", role="praise"]
== Praise for Getting Started with Atlas

[quote, Jane Doe]
____
"This book is awesome."
____
```

Praise pages render at the beginning of the book, before the title page, and are not included in the regular page numbering of the book.

Glossaries

A glossary file should begin with the word *glossary* in double brackets, followed by the glossary title and a blank line. Following the blank line should be another instance of the word *glossary*, this time in single brackets.

Each glossary entry should consist of one glossary term, followed by two colons and a space, then the glossary definition. If you'd like to add an additional paragraph to a glossary definition, add a plus sign ("+") on the following line by itself, and begin the additional paragraph on the line after it.

Here's an example of the markup:

```
[[glossary]]
== Glossary

[glossary]
Crawler:: A program used to index documents.
+
See Also Spider
```

Appendixes

To designate a file as an appendix, simply add the word *appendix* in single brackets at the top of the file. Immediately below it should be the title of the Appendix. For example:

```
[appendix]
== Resources

The following list of resources ...
```

Chapter Contributors

For books with multiple contributors, you may want an author name to appear with each chapter. Simply add the following markup above each chapter title:

```
[au="Author Name"]
== Chapter Title
```

For multiple authors of one chapter, the markup looks like this:

```
[au="Author Name", au2="Another Author", au3="Last Author"]
== Chapter Title
```

You can also use this same markup for forewords, prefaces, and appendixes. Please note that contributor names in a foreword will render at the end of the foreword, right-aligned, and preceded by an em dash. Foreword author attributions can also include an affiliation. The markup looks like this:

```
[au="Author Name", auaffil="Editor, Such and Such Journal"]
```

Inline Markup

Here are some standard O'Reilly typographical conventions with explanations of their use:

*_Italic_*
: Italic text indicates new terms, URLs, email addresses, filenames, and file extensions. The AsciiDoc markup is one underscore character on either side of the text to be italic.

  To mark a a few letters of a word in italics, or a word that abuts a non-whitespace character, double up the underscore characters on either side of the text, like this: `__Part__ial word i__tal__ic`

Bold
: Bolded text is used to emphasize a word or phrase. Note that O'Reilly house style prefers italics for emphasis. The AsciiDoc markup is one asterisk on either side of the text to be bolded.

  To mark a partial word in bold, or a word that abuts a non-whitespace character, double up the asterisk characters on either side of the text, like this: `**Part**ial word b**ol**d`

`+Constant Width+`
: Constant width, or monospaced, text is used for code, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords. The AsciiDoc markup is one plus sign on either side of the text to monospaced.

  To mark a partial word in constant width, or a word that abuts a non-whitespace character, double up the plus signs on either side of the text, like this: `++Part++ial word con++st++ant wid++th++`

`*+Constant Width Bold+*`
: Monospaced and bolded text is used to show commands or other text that should be typed literally by the user. The AsciiDoc markup is one asterisk and one plus sign on either side of the text. Note that the asterisk (*) pair should be on the outside and the plus-symbol (+) pair on the inside.

  To mark a partial word in constant width bold, or a word that abuts a non-whitespace character, double up the markup on either side of the text, like this: `** ++Part++**ial word constant w**++id++**th bold`

`_++Constant Width Italic++_`
: Monospaced and italicized text indicates where text should be replaced with user-supplied values or by values determined by context. The AsciiDoc markup is one

underscore and two plus signs on either side of the text. Note that the underscore (_) must be on the outside and the plus-symbol (+) pair on the inside.

To mark a partial word in constant width italic, or a word that abuts a non-whitespace character, double up the underscore markup on either side of the text, like this: `__++Part++__ial word con__++st++__ant width ital__++ic++__`

~subscript~
: For $_{\text{subscript}}$ text, use a tilde character (~) on either side of the subscript text.

^superscript^
: For $^{\text{superscript}}$ text, use a caret (^) character on either side of the superscript text.

Hyperlinks
: For hyperlinks to external sources, just add the full URL string followed by brackets containing the text you'd like to appear with the URL. The bracketed text will become a clickable link in web versions. In print versions, it will appear in the text, followed by the actual URL in parenthesis.

  The markup looks like this:

```
Visit the http://oreilly.com[O'Reilly website]
```

  Note that these will become hyperlinks in online versions, so for fake or example URLs, use `+constant width+` or _*italic*_ markup instead.

Troubleshooting Inline Markup

Using inlines in AsciiDoc can sometimes be tricky. If you're having problems with inline markup not rendering correctly, or if it's producing validation errors, check if it's an issue with nested inlines. Attempting to nest inlines within each other may sometimes result in problems if the markup is done in the wrong order. For example, constant width bold text uses an asterisk and a plus sign on either side of the text, but the asterisk pair must be on the outside of the plus sign pair, otherwise you'll get an error.

Another common issue is when inline markup occurs within a single word, like foobar, or if the inline markup abuts some other non-whitespace character on one or more sides (like an em dash), it may render incorrectly like this: foo\+bar. The fix for this is to double up the markup. For example: `+foo+`bar. See the previous section "Inline Markup" on page 25.

When all else fails, you can always use passthroughs to embed DocBook markup as is (see "Passthroughs" on page 42). For example, these example passthroughs are rendered as constant-width italic, constant-width bold, and constant-width bold italic, respectively.:

```
pass:[<replaceable>foo</replaceable>]
pass:[<userinput>foo</userinput>]
pass:[<userinput><replaceable>foo</replaceable></userinput>]
```

As an enhancement for finer-tuned semantic distinctions downstream, the Atlas configuration file supports the following variants, via `@role` attributes:

- To indicate a filename or path, use: `[role="filename"]_/path/to/file.ext_`
- For a book title, use: `[role="citetitle"]_This Book Needs No Title_`

Curly brackets, { and }, may cause rendering problems because they indicate macros in AsciiDoc. If you are having trouble getting curly brackets to render, try escaping them with backslashes like this: \{ and \}

Sidebars

Sidebar markup looks like this:

```
.Sidebar Title
****
Sidebar text is surrounded by four asterisk characters above and below.
****
```

Sidebars render like this:

Sidebar Title

Sidebar text is surrounded by four asterisk characters above and below.

Notes, Tips, Warnings, and Cautions

You have the option of using note, tip, warning, and caution elements for supplemental information. Please note that the O'Reilly stylesheets make no visual distinction between the way warning and caution elements render, as well as the way note and tip elements render.

You may also notice that some of the admonitions below have a title. We do support optional titles in admonitions (in most series). If you don't want a title for your note, tip, warning, or caution, just leave out the entire *.Tip Title* line in the markup below.

The markup for a note looks like this:

```
[NOTE]
====
Here's some text inside a note.
====
```

And here's how it renders:

Here's some text inside a note.

You'd create a tip with a title like so:

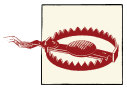
```
.Tip Title
[TIP]
====
Here's some text inside a tip.
====
```

…which will yield this result:

Titled Tip
Here's some text inside a tip.

Or you could use a warning:

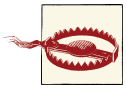
```
.Warning Title
[TIP]
====
Here's some text inside a warning.
====
```



Titled Warning
This is how warnings render. Warnings can be used for including supplemental information in your text. Including a warning title is optional.

Finally, here's a caution:

```
[CAUTION]
====
Here's some text inside a caution.
====
```



This is how cautions render. Cautions can be used for including supplemental information in your text.

Footnotes

To create a footnote, place the footnote macro directly after the text where the note number should appear, with no space between.[1] The content of the footnote should appear within the brackets. Footnotes at the end of a sentence belong after the period:

```
footnote:[This is a standard footnote.]
```

Here's an example of the markup placement in the text:

```
There should be no spacefootnote:[This is a standard footnote.] between the
text and the note number.
```

For an example of the footnote rendering, see the first paragraph in this "Footnotes" on page 29 section. You should see the content of the note rendering at the bottom of the page.

Quotes

To add a block quote to your text, use the word `quote` inside single brackets, followed by a comma and the full name of the quote's author. The text of the quote itself should appear immediately below, with four underscore characters above and below it.

Here's some example markup, a `<quote>` attributed to Benjamin Disraeli (by Wilfred Meynell, according to Frank Muir):

```
[quote, Wilfred Meynell]

____
Many thanks; I shall lose no time in reading it.
____
```

And here's how it renders:

> Many thanks; I shall lose no time in reading it.
>
> — Wilfred Meynell

Figures and Other Images

Full figure markup should include a unique ID attribute, so the figure can be cross-referenced (see "Cross-References" on page 41 for more info on using cross-references). Please note the figure IDs should be unique throughout the book. Non-captioned figures don't have ID attributes or figure numbers and are not cross-referenced.

Create a figure with a caption like so:

1. Otherwise you'll introduce extra space in the text

```
[[unique_id]]
.A Figure
image::images/tiger.png["An image of a cartoonish tiger head"]
```

And here's how it renders:

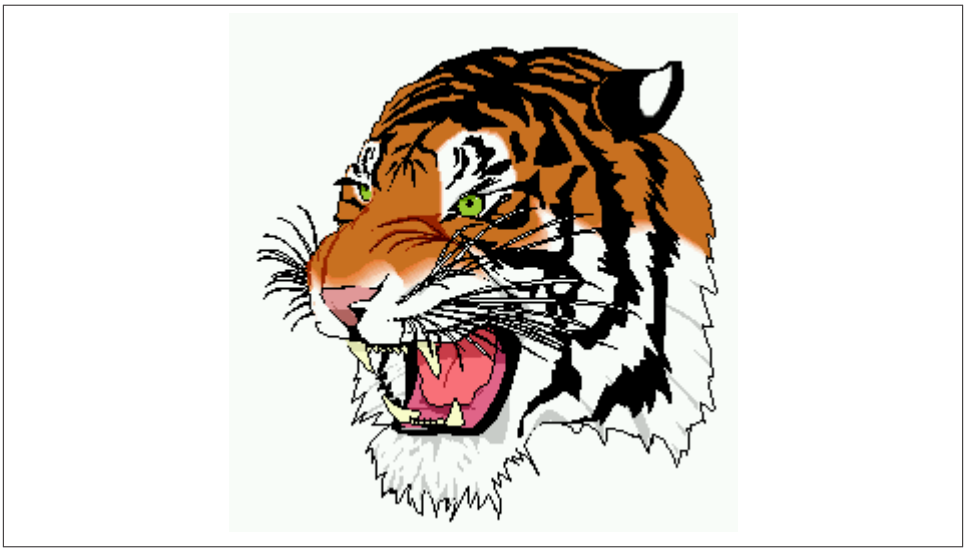


Figure 4-1. An example figure

If you'd like your figure to appear without a caption, just omit the `.title` line. Please keep in mind that in addition to having no title, these figures will also have no figure number and cannot be cross-referenced in the body of the text.

Here's the markup:

```
image::images/screenshot.png["Screenshot of the AsciiDoc website."]
```

And here's how it renders:

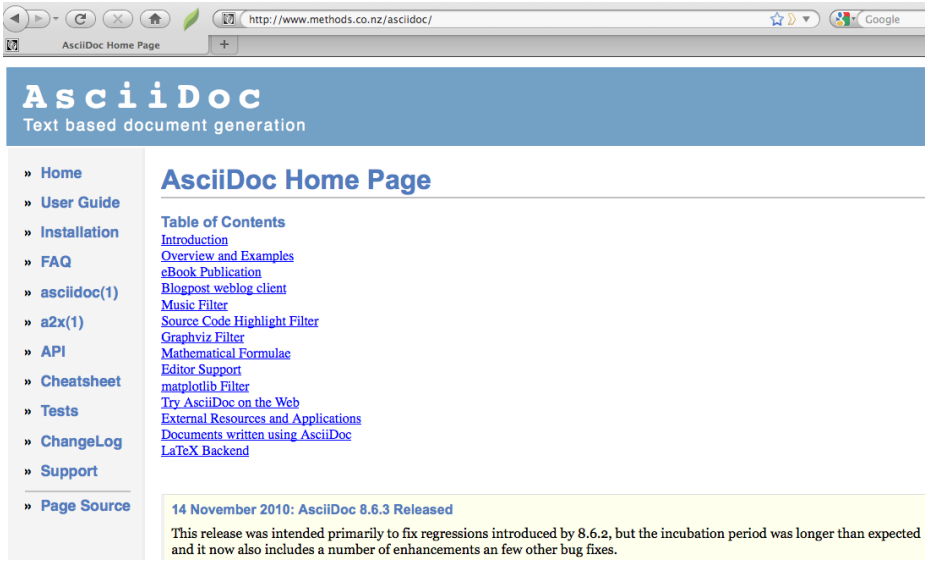


Figure floating

Figures appear exactly where you place them in the text, which can sometimes create PDF pages with a lot of white space. While it is not generally necessary, you can add an attribute of `float="true"` so that the text flows around the image:

```
[[unique_id]]
.A Figure
[float="true"]
image::images/tiger.png[]
```

To combine a float attribute with alt text, see "Adding alt text an images" below.

Adding alt text an images

To improve accessibility in your ebook files, please consider adding alt-text to the images. To do this, just add the descriptive text in quotes inside the brackets, which are at the end of the image markup:

```
[[unique_id]]
.A Figure
image::images/tiger.png["An image of a cartoonish tiger head"]
```

Please note, to combine the alt-text attribute with a `float="true"` attribute, use the following markup:

```
[[unique_id]]
.A Figure
[float="true"]
image::images/tiger.png["An image of a cartoonish tiger head"]
```

Adjust image size

While it should not be necessary in most circumstances, you can control the size of an image in the PDF output by adding an absolute value of width or height, like so:

```
image::images/filename.png[width="2in"]

image::images/filename.png[height="2in"]
```

Or you can use scale as a percentage to limit the width:

```
image::images/filename.png[scale="75"]
```

Note that you should not include a percentage sign.

Lists

There are three common types of lists. The O'Reilly Stylesheet and Word List has more details about when to use them, but here's the markup and an example of each.

Bulleted (aka itemized) lists

Here's the markup for a bulleted list:

```
* lions
* tigers
** sabre-toothed
** teapotted
* lions, tigers, and bears
+
Use a plus sign (on its own line) with the text below to add multiple para-
graphs to a list item.
```

The list renders as you'd expect:

- lions
- tigers

  — sabre-toothed

  — teapotted
- lions, tigers, and bears

  Use a plus sign (on its own line) with the text below to add multiple paragraphs to a list item.

Note that the final item in the list has multiple paragraphs. As you can see in the markup example, a plus (+) character can be used to add additional paragraphs to a list item.

Ordered (aka numbered) lists

Here's the markup for a numbered list:

```
. Preparation
. Assembly
.. Measure
.. Combine
.. Bake
. Applause
+
Use a plus sign (on its own line) with the text below to add multiple para-
graphs to a list item.
```

Using the above markup will generate a numbered list:

1. Preparation
2. Assembly
   a. Measure
   b. Combine
   c. Bake
3. Applause

   Use a plus sign (on its own line) with the text below to add multiple paragraphs to a list item.

Note that the final item in the list has multiple paragraphs. As you can see in the markup example, a plus (+) character can be used to add additional paragraphs to a list item.

Labeled (aka variable or term-definition) lists

Here's the markup for a labeled list:

```
Term 1::
  Definition/description
Term 2::
  Something else
```

Labeled lists look like this:

Term 1
: Definition/description

Term 2
: Something else

Tables

Atlas table styles vary slightly between series. If your material warrants something other than the default style as shown in Table 4-1, please consult with your editor.

Standard AsciiDoc markup for a simple table follows:

```
.A Table
[options="header"]
|=======
|P|Q|P^Q
|T|T|T
|T|F|F
|F|T|F
|F|F|F
|=======
```

Note that a single vertical bar (|)+ character is used to separate each box in the table. If you're having a problem with your table columns rendering oddly, make you have that you have the same number of vertical bars for each row.

The table will render with a title and a header row:

Table 4-1. An example table

P	Q	P^Q
T	T	T
T	F	F
F	T	F
F	F	F

Code

Code blocks (or as the AsciiDoc documentation refers to them, "listing" blocks), are defined using four hyphens above and below the code block content:

```
----
Hello world!

0        10        20        30        40
12345678901234567890123456789012345678901
----
```

Which will render like this:

```
Hello world!

0        10        20        30        40
12345678901234567890123456789012345678901
```

Formal code blocks (titled and cross-referenced)

If you'd like the code block to have a title and be able to cross-reference it, you should use the formal code block markup. As you can see below, you'll need to additionally surround the code block with four equals signs, a title, and an ID in double brackets:

```
[[EX1]]
.A code block with a title
====
----
Hello world!

0        10        20        30        40
12345678901234567890123456789012345678901234567890
----
====
```

The results:

Example 4-1. A code block with a title

```
Hello world!

0        10        20        30        40
12345678901234567890123456789012345678901234567890
```

Inline formatting within code

In AsciiDoc, there is no built-in mechanism for inline formatting within code.[2] If you want to use inline formatting—in particular, for standard O'Reilly typographical conventions such as **`<userinput> (CW+bold)`** and *`<replaceable> (CW+italic)`* or if you want to include line annotations—you can do so by using a passthrough block (see "Passthroughs" on page 42 for an explanation of passthroughs). Here's the markup to use a passthrough with the DocBook element `<screen>`:

```
++++
<screen>
hostname $ <userinput>date</userinput>
Sun Apr  1 12:34:56 GMT 1984
</screen>
++++
```

which renders like this:

```
hostname $ date
Sun Apr  1 12:34:56 GMT 1984
```

2. We do support `[subs="quotes"]` markup in passthrough blocks, but note that this markup can become complicated. Contact *toolsreq@oreilly.com* with any questions.

And here's the markup to use a passthrough with the `<programlisting>` and `<linean notation>` DocBook elements:

```
++++
<programlisting>from __future__ import with_statement # This isn't required in
Python 2.6
                     <lineannotation>Above is a comment in the code, while this
is an "annotation"</lineannotation>

with open("<replaceable>hello.txt</replaceable>") as f:
     for line in f:   <lineannotation>(note regular italic here vs. constant-
width in "hello.txt" on line above)</lineannotation>
        print line</programlisting>
++++
```

which renders like this:

```
from __future__ import with_statement # This isn't required in Python 2.6
                     Above is a comment in the code, while this is an "annotation"

with open("hello.txt") as f:
    for line in f:  (note regular italic here vs. constant-width in "hello.txt" on line above)
        print line
```

Syntax highlighting

The Atlas book-building toolchain supports syntax highlighting via Pygments. You need only add `[source]` above each code block that should include syntax highlighting, and specify the language of the code. For example:

```
[source,java]
----
int radius = 40;
float x = 110;
float speed = 0.5;
int direction = 1;
----
```

This will render in the EPUB, PDF, and KF8 (Kindle Fire only) as follows:[3]

```
int radius = 40;
float x = 110;
float speed = 0.5;
int direction = 1;
```

Here is the markup if you want to add syntax highlighting to a formal code block—note the placement of `[source]` is the same:

```
[[id_here]]
.Syntax highlighting sample
```

3. If you are viewing the HTML version of these guidelines on Chimera, syntax highlighting will not be shown.

```
====
[source,java]
----
int radius = 40;
float x = 110;
float speed = 0.5;
int direction = 1;
----
====
```

Pygments supports a wide variety of languages that can be used in `[source]`; see the full list available on the the Pygments website. Ebook readers that do not have color screens will still display the highlighting, but in more subtle shades of gray.

Please note the following caveats:

- The color scheme is consistent across books and cannot be changed at this time.
- Unless your book is printing in color, the PDF used for print will contain no highlighting.

External code files

There are two ways to include external code files. You can include an external file that is text-only (no markup like line annotations or inline formatting), or you can include an external file marked up with valid Docbook, which can contain inline markup. Callouts will work with either method.

To include an external code file that is text-only (no inline markup besides callouts), use the `include::` macro inside of a delimited code block, as shown here:

```
[source,java]
----
include::code/HelloWorld.java[]
----
```

For info about using callouts with your external Asciidoc code file, see "Code callouts" on page 40.

To include an external file that contains inline markup (e.g., line annotations or inline font formatting), use passthrough markup around around the `include::` macro, instead of code block delimiters:

```
++++
include::code/inline_markup.txt[]
++++
```

In the above case, the contents of *inline_markup.txt* would contain only the DocBook markup. In our example, which might look something like this:

```
<programlisting>Roses are <replaceable>red</replaceable>,
   Violets are <replaceable>blue</replaceable>. <lineannotation>This is a line
annotation</lineannotation>
Some poems rhyme;
   This one <userinput>doesn't</userinput>.</programlisting>
```

The results will look like this:

```
Roses are red,
   Violets are blue. This is a line annotation
Some poems rhyme;
   This one doesn't.
```

Code snippets

Atlas now supports including "snippets" in your book content so that you can excerpt just a portion of a code listing from an external code file.

The code snippet functionality is currently in beta. Feedback is welcome.

Each snippet that you'd like to pull into the book content must be delimited with `BEGIN` *`identifier`* and `END` *`identifier`* comment lines, where *`identifier`* is a unique identifier for the given snippet. Here is a sample JavaScript file with one snippet:

Example 4-2. factorial.js

```
alert("This code has a snippet below!");
factorial(3);

// BEGIN FACTORIAL_FUNC
function factorial(number) {
    if (number == 0) {
       return 1
       } else {
        return factorial(number - 1) * number
  }
}
// END FACTORIAL_FUNC
```

To include this snippet in an AsciiDoc file, add the following block:

```
[filename="factorial.js", language="js", identifer="FACTORIAL_FUNC"]
snippet~~~~
Put any descriptive text you want here. It will be replaced with the
specified code snippet when you build ebook outputs
snippet~~~~
```

When the ebook is generated, the following output will be present in place of the `snippet` block:

```
function factorial(number) {
    if (number == 0) {
       return 1
       } else {
    return factorial(number - 1) * number
  }
}
```

The `filename` attribute references the external file containing the snippet. Paths to filenames should be *relative* to the AsciiDoc file to which you're adding the snippet. So if you've got all your book files in the repository root directory, and all your code files in a *code* directory in the root directory, your filename attribute will look like `filename="code/filename.js"`.

The `language` attribute is the programming language of the code snippet you're including. Current supported language values are:

- `clj` (clojure)
- `groovy`
- `java`
- `js` (javascript)
- `rb` (ruby)
- `scala`

If you'd like support added for another programming language, please give us feedback.

The `identifier` attribute references the unique identifier used in your snippet delimiters.

Some Notes on Snippet Handling

You can nest code snippets within each other like so:

```
// BEGIN FACTORIAL_FUNC
function factorial(number) {
// BEGIN IF_STATEMENT
    if (number == 0) {
       return 1
       } else {
    return factorial(number - 1) * number
  }
// END IF_STATEMENT
```

```
}
// END FACTORIAL_FUNC
```

When snippets that contain nested snippets are referenced for inclusion into book content, the delimiters for nested snippets are removed. For example, if the above `FACTORIAL_FUNC` snippet is included into the book, the delimiters for the `IF_STATEMENT` will be stripped out from the code listing.

If you'd like to use AsciiDoc callouts in your external code files, you can comment them out (to ensure the code is runnable), and the comment delimiters will be removed when the snippet is incorporated into the book. For example, the following snippet:

```
// BEGIN FACTORIAL_FUNC
function factorial(number) {
    if (number == 0) { // ❶
       return 1
       } else { // ❷
    return factorial(number - 1) * number
  }
}
// END FACTORIAL_FUNC
```

Will be converted to this in the ebook output:

```
function factorial(number) {
    if (number == 0) {  ❶
       return 1
       } else {  ❷
    return factorial(number - 1) * number
  }
}
```

If snippet incorporation fails (e.g., due to a bad `filename` reference), any text included between the `snippet~~~~` delimiters will be used as fallback text.

Code callouts

Code callouts are used to mark specific lines of code with icons keyed to explanatory text outside the code block. These icon pairs function as bidirectional links in electronic PDF and downstream formats (i.e., you can click on the icon in the code to jump to the explanation, and vice versa).[4]

Here's the AsciiDoc markup for code callouts:

```
----
Roses are red,
```

4. The built-in AsciiDoc mechanism is somewhat more limited in that icons are hyperlinked from text to code, but not vice versa. If two-way linking is important to you, consider using a passthrough block (see "Passthroughs" on page 42).

```
   Violets are blue. <1>
Some poems rhyme;
   This one doesn't. <2>
----
<1> Violets actually have a color value of +#9933cc+.
<2> This poem uses the literary device known as a "surprise ending."
```

The output will include callouts numbers in the code, followed by a numbered list, as shown here:

```
Roses are red,
   Violets are blue. (1)
Some poems rhyme;
   This one doesn't. (2)
```

(1) Violets actually have a color value of `#9933cc`.

(2) This poem uses the literary device known as a "surprise ending."

To use code callouts with an external AsciiDoc code file, add the callout text items below the code block, like so:

```
[source,java]
----
include::code/HelloWorld.java[]
----
<1> This is number one.
<2> This is number two.
```

Cross-References

All references to titled block elements and book components—figures, tables, examples, sections, chapters, parts, and so on—should be marked up as cross-references, not entered as plain text. Cross-reference markup will become a live hyperlink to the target in digital versions and will resolve automatically if you move numbered elements (figures, chapters, etc.) around while editing.

To insert a cross-reference, follow these steps:

1. Note the id of the element you are referencing. If the element does not have an id, you will need to add one. For the book to be valid, ids must be unique across the entire book, have no spaces, and not start with a number. For example, a figure id looks like this:

   ```
   [[unique_id]]
       .Figure title
       image::images/figure.png[]
   ```

2. Once you have the id, you can insert a cross-reference to that element elsewhere in the text by enclosing the id in double angle brackets, like so:

```
<<unique_id>>
```

The Atlas build system will transform this ID into a standard cross-reference (or `<xref>`) for you: the rendered text will adjust automatically if you later move the target or reword its title, and it will work as a hyperlink in digital versions.

Table 4-2 shows the standard text generated from the cross-references in PDF builds.

Table 4-2. Standard cross-reference formats

Target	Generated Cross-Reference Text
chapter	Chapter 17
table	Table 4-1
figure	Figure 2-3
example	Example 3-5
sidebar	"Fooing the Bar" on page 23
section	"Inline Macros" on page 14

Here are some live examples (hover over the text in the PDF to locate the hyperlink):

- See "Indexing" on page 44 for details.
- The results is shown in Figure 4-1.
- Flip ahead to Chapter 6 for a preview.

which were generated from this source:

```
* See <<indexing>> for details.
* The results is shown in <<unique_id>>.
* Flip ahead to <<working_with_github>> for a preview.
```

Passthroughs

Chapter 4 makes several references to "passthroughs." You can use a DocBook passthrough when you've got an especially complex piece of markup or you're trying to do something that AsciiDoc doesn't support. AsciiDoc supports two types of passthroughs:

Inline passthroughs
: For inlines, you can indicate a passthrough at any time by like this: `passthrough goes here`. See "Troubleshooting Inline Markup" on page 26 for examples of inline passthroughs.

Block passthroughs
: You can also pass though a block of DocBook XML. Here's the image markup that we used in Chapter 4, passed through as DocBook:

```
----
<figure id="unique_id" float="none">
<title>A Figure</title>
<mediaobject>
  <imageobject>
  <imagedata fileref="images/tiger.png"/>
  </imageobject>
</mediaobject>
</figure>
----
```

Math Support via LaTeX

Atlas supports math equations set as LaTeX. For inline LaTeX equations, use the `latex math:` macro as follows:

```
The Pythogorean Theorem is latexmath:[\(a^2 + b^2 = c^2\)].
```

yielding this result:

The Pythogorean Theorem is $a^2 + b^2 = c^2$.

For block TeX equations, use a `latexmath` passthrough as follows:

```
[latexmath]
++++
\begin{equation}
{x = \frac{{ - b \pm \sqrt {b^2 - 4ac} }}{{2a}}}
\end{equation}
++++
```

And the results:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

You could also add a title. Use this:

```
[latexmath]
.The quadratic formula
++++
\begin{equation}
{x = \frac{{ - b \pm \sqrt {b^2 - 4ac} }}{{2a}}}
\end{equation}
++++
```

And get:

Equation 4-1. The quadratic formula

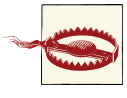
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Page Breaking

If you want to insert a hard page break into your PDF builds, you can do so with this passthrough:

```
++++
<?hard-pagebreak?>
++++
```

Please note that adding this page break processing instruction will have no effect on the EPUB and KF8 files.

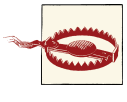


Please do not use AsciiDoc's mechanisms for forcing page breaks, as these don't mesh with Atlas's book building tools.

Controlling Line Breaks

Use an inline passthrough to prevent a line break:

```
pass:[<phrase role='keep-together'>Don'tBreakMe</phrase>]
```



Please do not use AsciiDoc's mechanisms for forcing line breaks, as these don't mesh with Atlas's book building tools.

Indexing

This section describes how to create an automatically generated index in AsciiDoc. We've extended indexing in AsciiDoc to include ranges, sees, and see alsos.

Create the index file

To include an index in your book, first create a blank *index.asciidoc* file and add this single line of text:

```
== Index
```

The index entries will be auto-generated during the build process, using the index markers you place in the text, as described in the next section. Just be sure to include the *index.asciidoc* file in the build.

See the AsciiDoc source of this documentation for an example index.

Syntax

Each index marker is generally surrounded with quotation marks. But for basic index entries without attributes (i.e., without ranges, a "see," a "see also," or a "sortas"), you do not need to enclose terms in quotation marks. For example, the following markup is fine:

```
(((XML, RDF, SPARQL)))
```

However, if you include any attributes, you must put all entries in quotes as seen below, e.g.:

```
(("XML", "RDF", "SPARQL", seealso="XQuery")))
```

For anything not mentioned below, please refer to the AsciiDoc User Guide, or contact us if you have index entries with special characters (e.g., quote marks, commas) and need guidance on how to format the markup.

Basic index entry:

```
(("primary index term")))
```

Secondary entry:

```
(("primary index term", "subentry")))
```

Tertiary entry:

```
(("primary index term", "subentry", "sub-subentry")))
```

An index entry with a range:

```
The future of ebooks is HTML5.((("HTML5", id="ix_html5", range="startofrange")))
In the following pages
...
blah blah blah canvas
blah blah blah local storage
blah blah blah geolocation
...
Learn HTML 5 today!(((range="endofrange", startref="ix_html5")))
```

An index entry with a "(see)" and no page reference:

```
Flash has been supplanted by HTML5.((("Flash", see="HTML5")))
```

An index entry with a "(see also)":

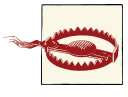
There are two markup options for "see also" entries.

- You can embed the `seealso` with the primary entry. This will render a page number next to the primary entry:

```
In addition to the Makerbot, RepRap also allows you to make 3-D stuff
((("Makerbot", seealso="RepRap")))
```

- Or if there's a subentry, you can embed the `seealso` entry after the it:

```
In addition to the Makerbot, RepRap also allows you to make 3-D stuff
((("Makerbot", "3-D", seealso="RepRap")))
```



Any "(see also)" entries that are nested under primary entries with no page numbers of their own (i.e., primary entries that have subentries) will render with extra indentation.

To alphabetize an index entry differently than the default, use "sortas" to assign a new label to be used for alphabetization. For example:

```
Makerbot lets you produce your own 3-D trinkets.((("3-D", sortas="three-d")))
```

CHAPTER 5

Using Git with Atlas

While Atlas hides a lot of the complexity of using git, there are still times when you may want to use it, such as:

- You hate editing in a web browser and would prefer a text editor, such as TextMate, Emacs, or BBEdit.
- You want to write when you're offline—during a flight, for example.
- You've got a bunch of images to upload in bulk.

If this sounds like you, then using git directly may be an attractive option. This is a quick introduction to using git with Atlas. If you want to know more about git and all the things you can do with it, check out these resources:

- Git Cheat Sheet (GitHub). A great reference to the key commands.
- Git for Designers. A nice introduction to git for non-programmers.
- Pro Git. The online version of Scott Chacon's excellent book, *Pro Git*.
- O'Reilly Webcast: Git in One Hour. A nice video from Scott Chacon that introduces all the key parts of using git.

Git Terminology and Command Reference

Here are some of the key terms you'll need to understand to use git:

Local repository
: A git repository that is on your local drive. You can make any changes you want to this copy without affecting other files.

Branch
: A named version set of files within a git repository. The default name is "master." We won't be doing much with branches, but they're a very handy tool that you can lean more in the Basic Branching and Merging chapter of the git community book.

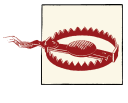
Remote repository
: A named link to a git repository on another machine. You can have as many remote repositories as you want. "Origin" is the default name, but you can choose any name you like when you add new "remotes" (links to other repositories on different servers).

Push and pull
: git lingo for sending the changes made in a local repository to a remote repository, or vice versa.

Cloning
: Downloading a remote git repository onto your own machine so that you can edit it locally.

Forking
: GitHub lingo for cloning a repo from someone else's GitHub account into your own so that you can modify and change without touching the original copy. Forking a repo is often the first step when you want to start working on a project.

Commit
: A commit saves all your changes under version control.



One major difference between editing in the Atlas wiki and editing locally is that, in the wiki, all changes are committed for you automatically. This is *not* the case when you're editing locally: you can make and save changes, but until you commit, you cannot push changes to Atlas. Why? The reason is a reflection of git's history as a software development tools.

Git was designed to allow developers to experiment with their code by changing lots of files. Assuming everything works, the programmer can then commit the changes into the master branch of the code. If the changes don't work, he or she can simply roll back the changes to the last version that worked and start over. So a commit is an affirmative statement that the changes are acceptable.

Table 5-1 summarizes some commonly used commands.

Table 5-1. Command quick reference

`git init`	Creates a new blank repo

`git commit -a -m"Your helpful commit message"`	Commits changes to the repo
`git status`	Returns the commit status of the current repo
`git add *`	Adds all files to the repo
`git remote add _remote_name_ _remote_URL_`	Adds a remote repository to which you can push or pull changes
`git remote -v`	Lists the remotes in your repo
`git push _remote_name_ _branch_name_`	Pushes changes from the specified local branch specified to a remote repo. We'll mostly use "git push origin master"

Frequently Asked Questions About Using Git with Atlas

Where Do I Get Git?

You can download git from the git-scm site. Just follow the instructions for your platform.

How Do I Clone My Project from Atlas?

To pull down your project from Atlas:

1. Go to the Admin tab and copy the Git URL (it's the first field)
2. Drop into a command line and use `git clone _<your_git_url>_` to pull down the repo. By default, using this command as is will pull the repo into a directory name based on the ISBN. You can override this by putting a new directory name at the end of the command.
3. You'll be prompted for a password—enter the password you use to log into Atlas

Here's a log:

```
$ cd ~/Desktop
$ git clone https://odewahn%40oreilly.com@atlas-admin.oreilly.com/
  git/1230000000189.git
Cloning into 1230000000189...
Password:
remote: Counting objects: 68, done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 68 (delta 16), reused 0 (delta 0)
Unpacking objects: 100% (68/68), done.
$ cd 1230000000189
```

How Do I Write Locally Using TextMate?

You can use any text editor you want to edit Atlas projects. If you're on a Mac, here's how you'd use the popular TextMate editor,

1. Install the editor. You can download it from the TextMate site.
2. Install and configure the AsciiDoc bundle.
3. Enable command-line usage.
4. Edit your files.

More details on the bundle and configuring AsciiDoc for the command line follows.

Once you've installed TextMate, go grab the AsciiDoc bundle to make it much easier to work with AsciiDoc. It will give you features like automatic previews, source highlighting, and so forth. Here's what you do for this:

```
$ mkdir -p /Library/Application\ Support/TextMate/Bundles
$ cd ~/"Library/Application Support/TextMate/Bundles/"
$ git clone git://github.com/zuckschwerdt/asciidoc.tmbundle.git "AsciiDoc.tmbun
dle"
$ osascript -e 'tell app "TextMate" to reload bundles'
```

Now the the bundle is installed, your AsciiDoc markup will have all the color-coded goodness you've come to expect in a text editors.

You have to give the files an extension of *.asciidoc* for the syntax highlighting to work.

Change into the directory where you installed the sample repository and type the following command:

```
$ mate .
```

This command will open the editor and display the *project drawer*, which is a navigation tree that you can use to move between files. Use the project drawer to open the file called *sec_environments.asc*, as shown in Figure 5-1.

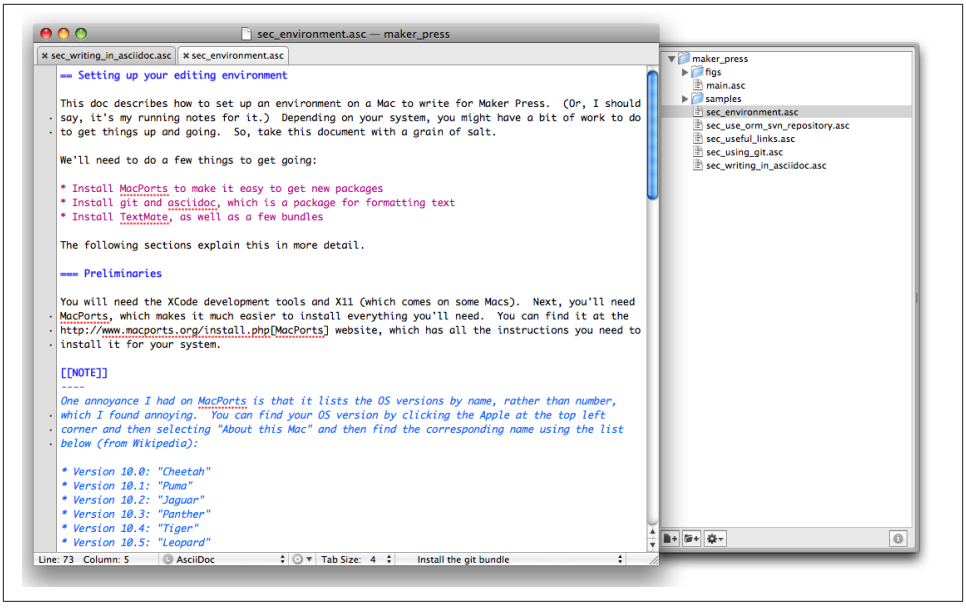


Figure 5-1. Using TextMate and the AsciiDoc Bundle

If you've worked in a wiki before, this markup should look pretty familiar. Also, note how the various AsciiDoc elements are all nicely color coded because of the AsciiDoc bundle that you installed earlier.

To run TextMate from the command line, you must configure your system so that it "knows" where TextMate is installed. The simplest way to do this is to use the "Terminal Usage" feature right in TextMate's control bar. Just click "Help → Terminal Usage…" and then click "Create Link." Figure 5-2 shows how this works.

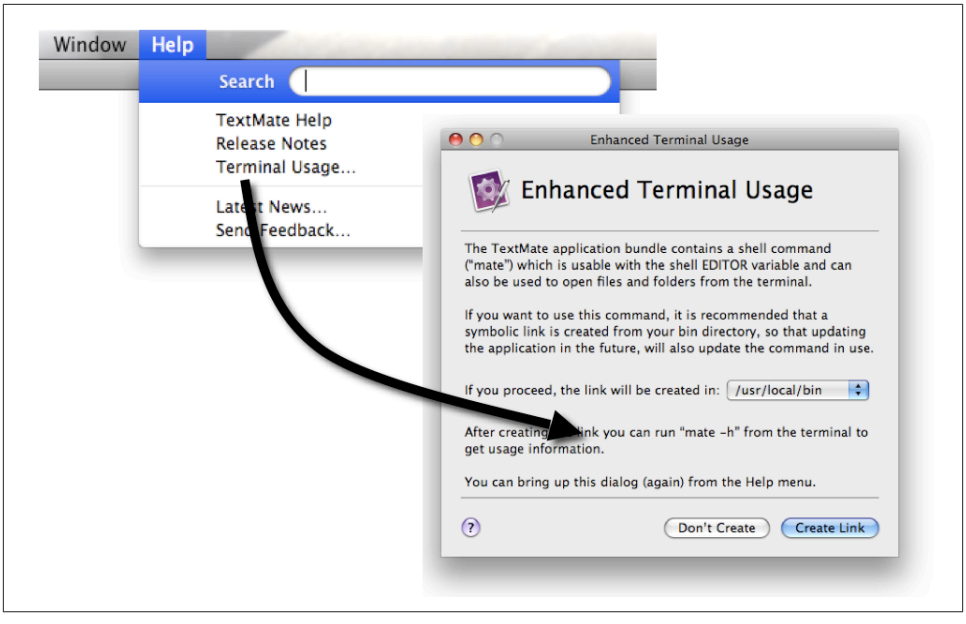


Figure 5-2. Setting Up TextMate for the Command Line

I've Edited My Files. Now What?

Once you've made your edits, you use two commands to add any new files and commit your changes:

- Add any new files so that git can start tracking them. Use `git add _<filename>_` to add an individual file. Use `git add .` to add all files in the current directory and all subdirectories.
- Commit the changes using `git commit -a -m'_commit message_'`. Try to use the commit message to leave yourself a note about what you were doing. For example, if you were just adding a big section on the *foo* method, you'd use a message like "Added section covering foo."
- Push the changes back up to Atlas using `git push origin master`.

Here's an example:

```
$ git add .
$ git commit -a -m"Made some changes while on the plane"
$ git push origin master
```

I Am Trying to Push Some Changes to Atlas, but It Keeps Reporting That Everything Is Up to Date. What's Up?

You probably forgot to either add any new files, or you forgot to commit your changes. (Or both!) You can check if you have any changes using `git status`, like this:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   git_quick_start.asciidoc
#
no changes added to commit (use "git add" and/or "git commit -a")
```

When you commit the changes, you'll get something like this:

```
$ git commit -a -m"Minor edits"
[master 955189b] Minor edits
 1 files changed, 47 insertions(+), 6 deletions(-)
new-host:1230000000197 odewahn$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
nothing to commit (working directory clean)
new-host:1230000000197 odewahn$
```

Hey, My Push to Atlas Keeps Getting Rejected. What's Up with That?

If you're getting a message that your changes are rejected, it's most likely because someone has changed the files on Atlas since you started working locally. To fix this, you'll need to commit your current changes and then use `git pull origin master` to pull in the changes from Atlas. Once you've synced the changes, you'll be able to push your work back up.

Here's the rejection notice:

```
$ git push origin master
Password:
To https://odewahn%40oreilly.com@atlas-admin.oreilly.com/git/1230000000197.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'https://odewahn%40oreilly.com@atlas-
admin.oreilly.com/git/1230000000197.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes (e.g. 'git pull') before pushing again. See the
'Note about fast-forwards' section of 'git push --help' for details.
```

To fix this, you need to pull in the new changes, like so:

```
$ git pull origin master
Password:
```

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://atlas-admin.oreilly.com/git/1230000000197
 * branch            master     -> FETCH_HEAD
Updating e26e9b6..fd7c13d
Fast-forward
 book.asciidoc |    2 --
 1 files changed, 0 insertions(+), 2 deletions(-)
```

HTTP Error When Pushing to Atlas

When pushing large batches of files to Atlas (e.g., a set of figure images), you may encounter an error like the following:

```
$ git push origin master
Password for 'atlas-admin.oreilly.com':
Counting objects: 39, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (35/35), done.
Writing objects: 100% (35/35), 83.19 MiB | 604 KiB/s, done.
Total 35 (delta 2), reused 0 (delta 0)
error: RPC failed; result=22, HTTP code = 502
fatal: The remote end hung up unexpectedly
fatal: The remote end hung up unexpectedly
```

This usually indicates that your HTTP POST buffer is too small to handle the files being posted. Try increasing the buffer size by running the following command:

```
$ git config http.postBuffer 524288000
```

And then try your `git push` again.

For more details/context on Git and the HTTP post buffer, see this Stack Overflow post.

How Do I Get a diff Between the Files I Have Locally and the Files That Are on Atlas (Regardless of the Number of Commits)?

Rather than using `git pull`, use `git fetch`, like this:

```
$ git fetch origin
Password:
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://atlas-admin.oreilly.com/git/1230000000065
   92d4f99..e9fbbe2  master     -> origin/master
```

This fetches the changes into a local branch, but does not merge them in automatically, which is what `pull` does. (In fact, as Mark Longair argues in git: fetch and merge, don't pull, this workflow is much better than just blindly pulling in remote changes without review.)

Once you've fetched, you can use this command to see the files that have changed between your local copy and the remote copy:

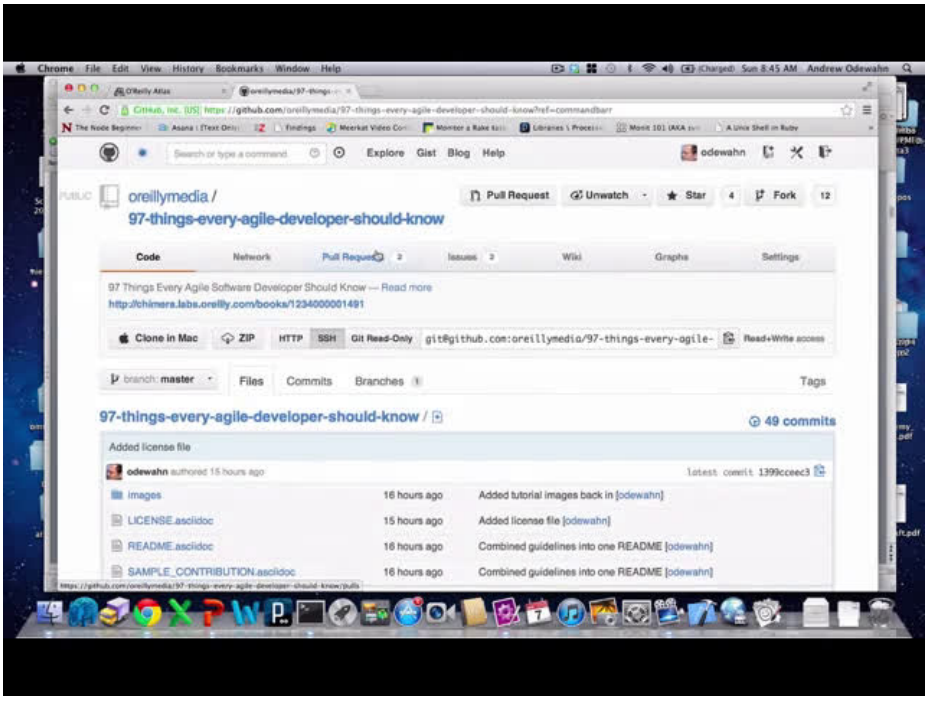
```
$ git diff --name-only master..origin/master
```

This command will just give you the list of files. (If you want to see the actual differences, just leave off the "--name-only" flag). Once you're satisfied that nothing nefarious is in there, you can then merge in the changes you just fetched using the command `git merge origin/master`, like this:

```
$ git merge origin/master
Updating 92d4f99..e9fbbe2
Fast-forward
 ch01.asciidoc |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

CHAPTER 6
Working with GitHub

Because book files in Atlas are managed with git, you could easily sync those files with a repository on GitHub. This following video describes the overall flow of working with GitHub in Atlas:



Comments about this video?

Post any questions or comments here.

The remainder of this chapter will give you step-by-step instructions:

If you're totally new to GitHub, the best place to start is the GitHub help pages. They'll walk you through what you need to know to set up an account, create a repo, set up your security credentials, and so on. Once you've got an account and have successfully completed the steps on setting up git, it's pretty simple to move stuff back and forth between Atlas and GitHub.

To put your code on GitHub, first, create a new repository. You'll be prompted to enter a name, a description, and whether you want to make the repo public or private (available only if you have a paid account). It will look very similar to Figure 6-1.

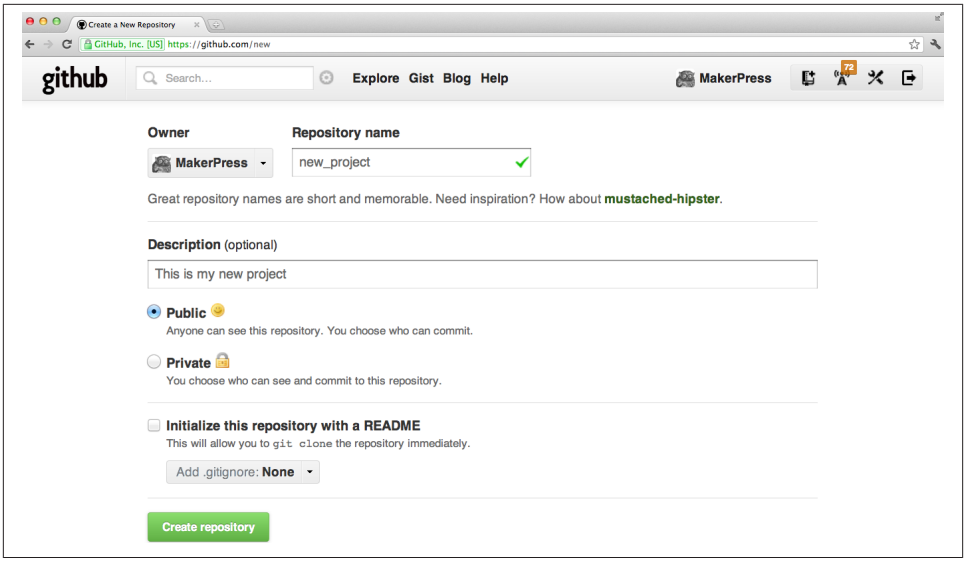


Figure 6-1. Create a new repository on GitHub

Once you create a project, you'll see a screen that lists some helpful commands for what you'll do next. Locate the "Existing Git Repo?" section and then find the line that looks like this:

```
$ git remote add origin git@github.com:MakerPress/new_project.git
```

It will look something like Figure 6-2.

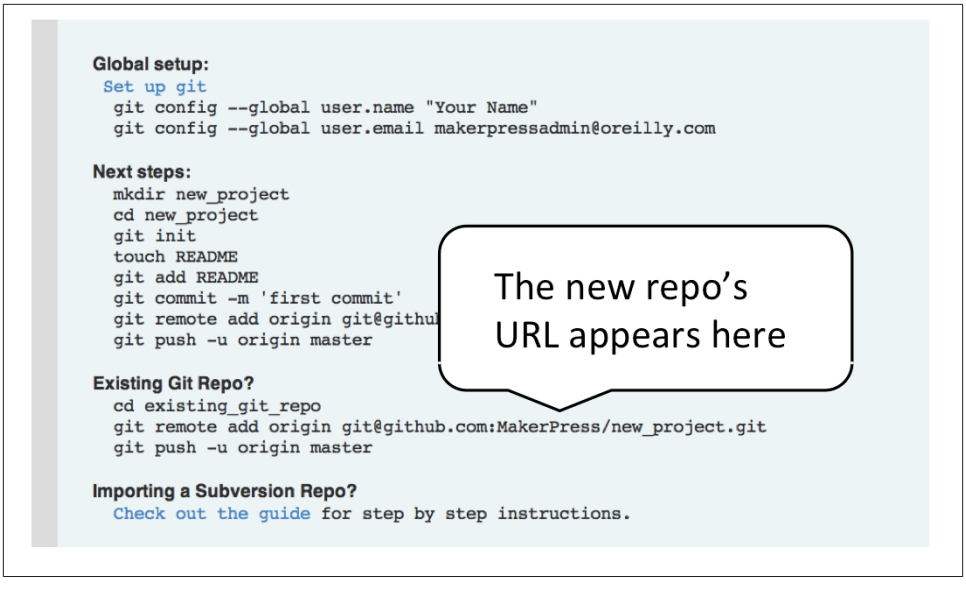


Figure 6-2. The new repo's URL appears in the "Existing Git Repo?" section

Once you've got the line, copy the repo's url (in our example, it's *git@github.com:MakerPress/new_project.git*) and enter the following command in the directory where your local Atlas repo is stored. (Note that the word "origin" is the only thing we're changing from the original command.):

```
$ git remote add github git@github.com:MakerPress/new_project.git
```

Once you've set up the new remote, you can push to it with this command:

```
$ git push github master
```

You can then take full advantage of all the amazing features and community available on GitHub.

Conversely, if you already have a repo on GitHub that you'd like to pull into Atlas, all you have to do is clone it down and add a new remote to an Atlas repo, like this:

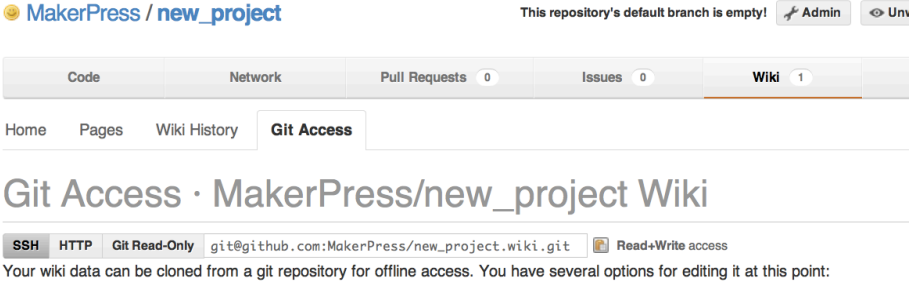
```
$ git clone git@github.com:MakerPress/new_project.git
Cloning into new_project...
warning: You appear to have cloned an empty repository.
admins-MacBook-Air-2:Desktop odewahn
$ cd new_project/
admins-MacBook-Air-2:new_project odewahn
$ git remote add atlas https://odewahn%40oreilly.com@atlas-
admin.oreilly.com/git/1230000000197.git
admins-MacBook-Air-2:new_project odewahn
$ git push atlas master
...
```

As we continue to improve Atlas, we'll add features to allow you to easily move projects back and forth from within the UI.

Using GitHub Wikis

GitHub wikis are really cool, since they store all your data as a git repo that you can clone, just like any other. If you wanted, you could write your entire book on a GitHub wiki using either AsciiDoc or Markdown (if you don't need really complex markup) and then pull it straight into Atlas to build the project.

To clone a GitHub wiki, first find the URL for the wiki's git repo, which appears on the "Git Access" tab:



Once you have the wiki's URL, you can clone it to your local system and add a remote back to Atlas so that you can move data back and forth with ease:

```
$ git clone git@github.com:MakerPress/new_project.wiki.git
Cloning into new_project.wiki...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.

$ cd new_project.wiki/

$ git remote add atlas https://odewahn%40oreilly.com@atlas-
  admin.oreilly.com/git/1230000000197.git
$ git push atlas master
```

Note that you'd need to pull in any changes from the GitHub wiki into Atlas.

CHAPTER 7

Using Atlas to Collect Early Feedback

O'Reilly is testing ways to help authors get early feedback on books being written in Atlas, O'Reilly's new authoring tool. Chimera is a site where you can host a web version of your book, with one-click publishing from Atlas (our authoring platform). Atlas and Chimera make it easy to:

- Receive comments from readers at a paragraph level, much like our earlier OFPS system.
- Publish surveys, polls, and other feedback tools. In addition to helping you answer specific questions, asking readers to complete a poll or survey is a great way to create interest around your project.
- View your Google Analytics data. In addition to direct feedback from readers, you can use Google Analytics to see how readers are acually using your content.

Our goal is to help you improve your book by getting it out as soon as you feel it's ready for people to look at it.

The following sections provide more detail for each step.

Receive Comments

When you publish your book to Chimera, readers can leave comments for you simply by clicking on a paragraph. This will open a comment box where they can leave feedback that will flow back into the authoring system, as shown in Figure 7-1.

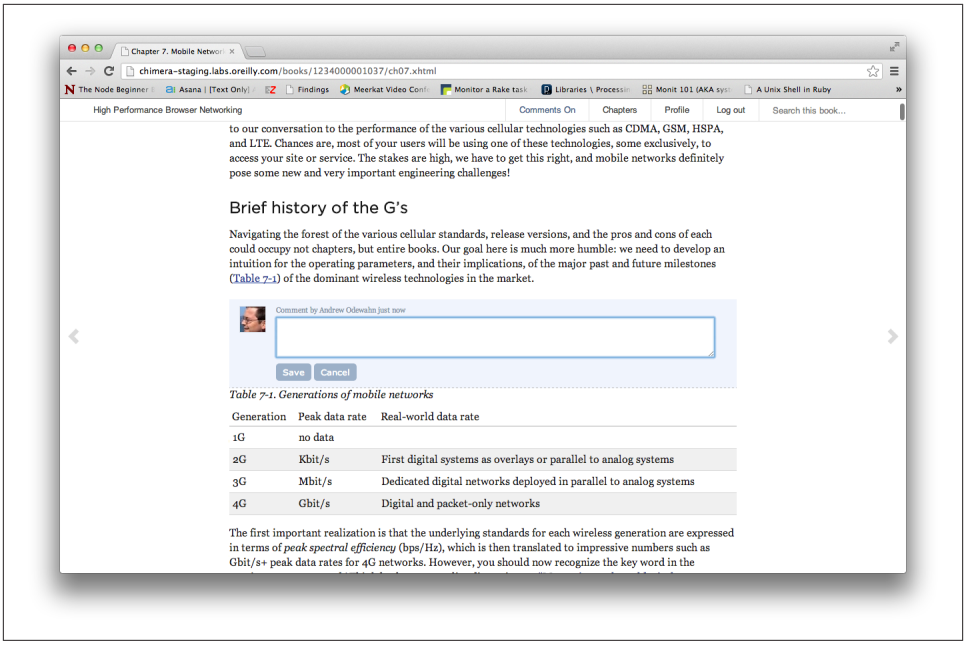


Figure 7-1. Readers can leave comments on each paragraph.

Publish Surveys and Polls

Surveys and polls are great ways to quickly collect data from your readers on targeted topics. In addition, asking for targeted feedback is a great great tool for promoting your book. Figure 7-2 shows how an embedded survey might appear.

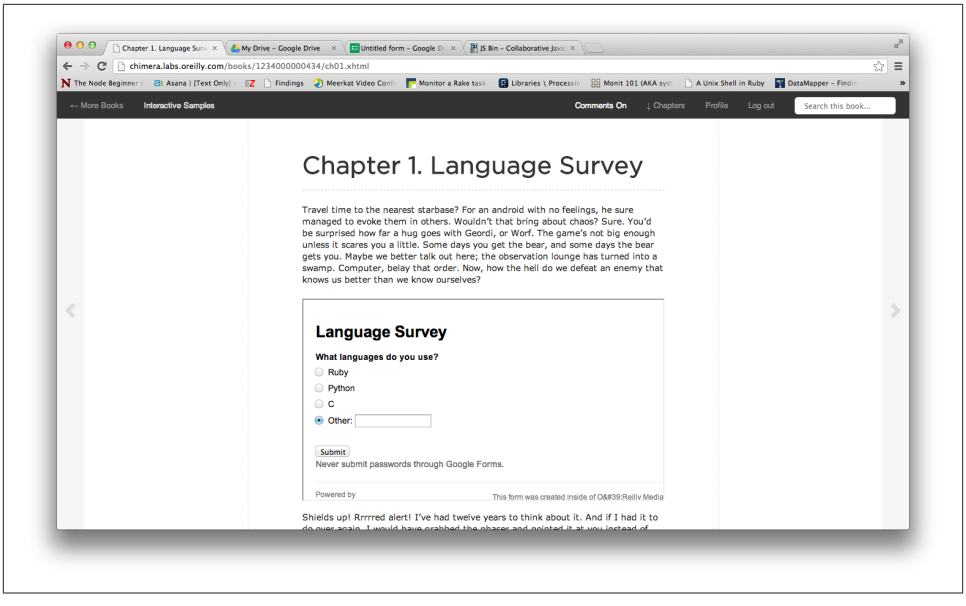


Figure 7-2. Embedding a survey inside your book on Chimera is a great way to gather early feedback.

There are three main steps for collecting this type of feedback:

1. Create a survey form in Google Docs
2. Embed the form in a JSBin or JSFiddle
3. Embed the JSBin or JSFiddle into your project

Figure 7-3 illustrates this in action.

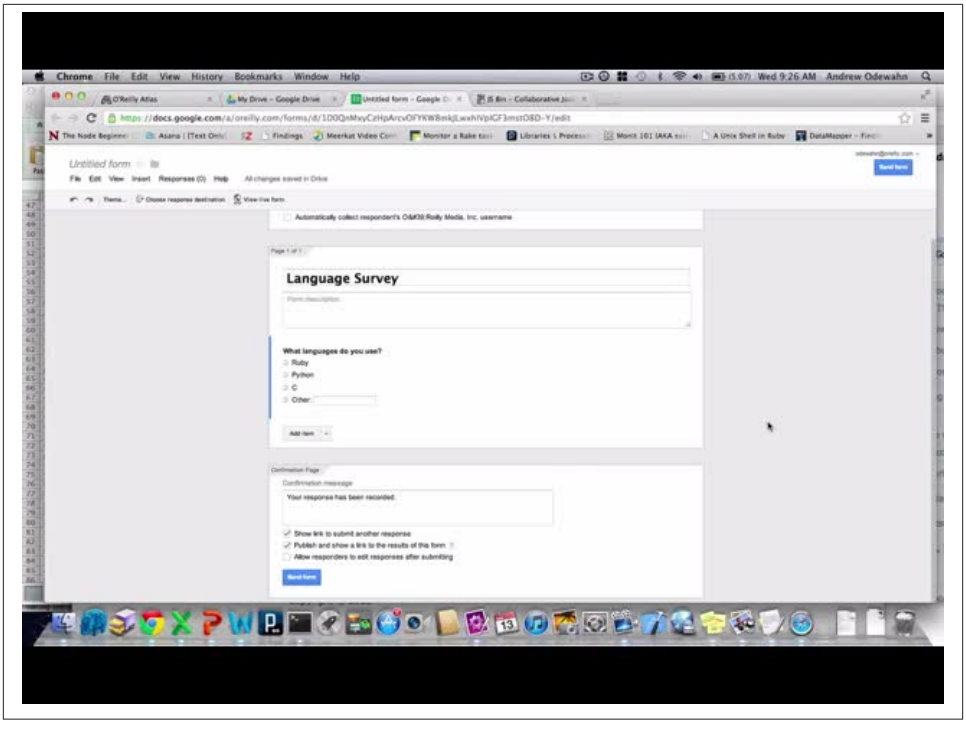
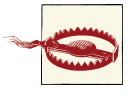


Figure 7-3. There are 3 steps to embedding a survey: create the survey in Google drive, embed it in a JSBin, and source the JSBin into Atlas.

View Google Analytics Data

Google Analytics is a powerful tool for tracking how readers actually use your material: where they come from, what they read, what search terms they use, and a host of other invaluable information.



Before you can use Google Analytics, we'll need to link our account to yours so that you can see your data. Please make a request in the Atlas feedback so that we can set up your account.

You can either log in directly to the google analytics portal or you can turn on GA's "In Page Analytics" to see the stats for your book directly in the site, as shown in Figure 7-4.

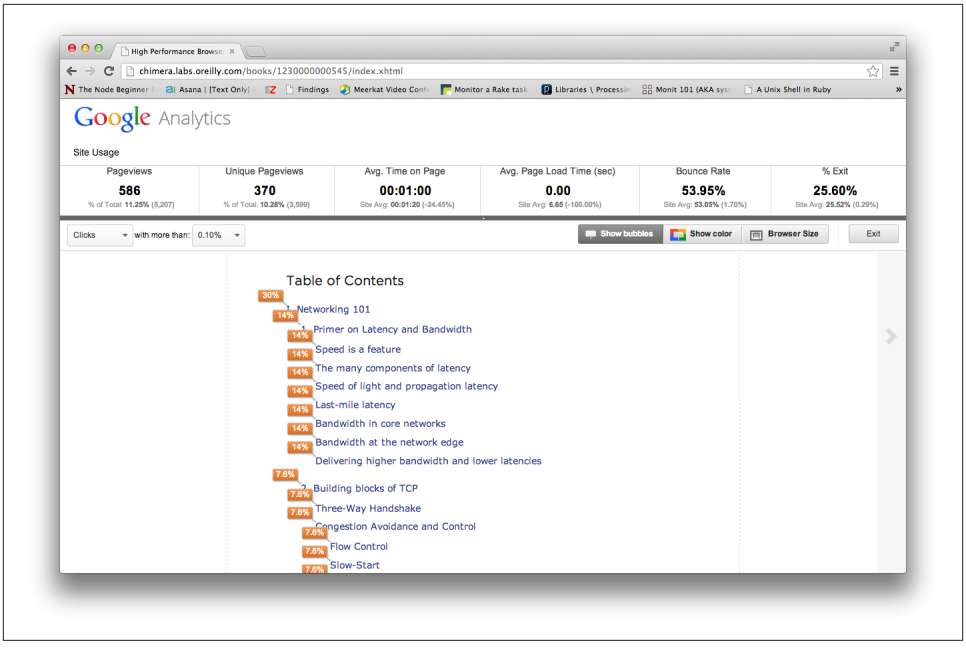


Figure 7-4. Google's "In Page Analytics" will superimpose the stats for your book directly in the browser.

We're working on more documentation for how to make the most of this powerful tool.

CHAPTER 8
Atlas Social Sidebar

The Social Sidebar is a new feature of Atlas to help you more easily engage with your readers, collaborators, and O'Reilly. The sidebar, which appears in the slide out panel on the right hand side of the writing tab, has two main functions:

- Comments from Chimera. This section pulls in comments from Chimera, the web-version of your book. Comments left in Chimera will appear as a list in the side panel so that you can address them in your book.
- Chat and Commands. This section allows you to use interactive chat, either with your collaborators or with the new Atlas chatbot.

Figure 8-1 shows the two different functions available in the sidebar.

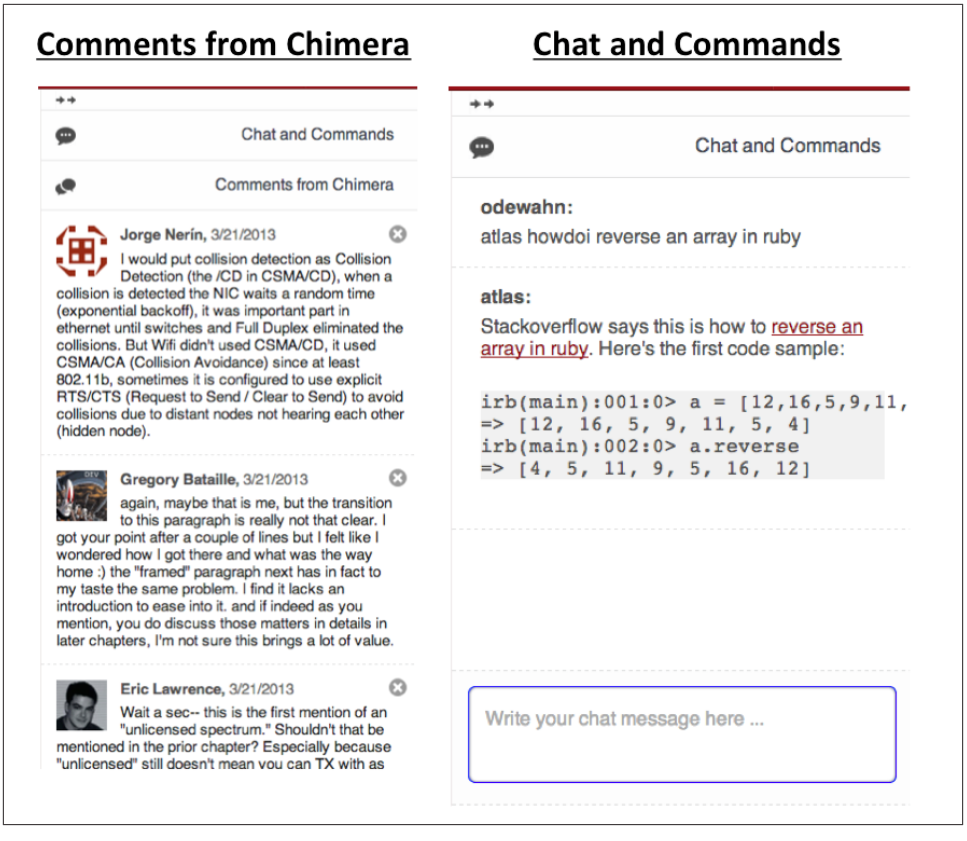


Figure 8-1. Click on the right sidepanel in Atlas' writing tab to open the social sidebar. The figure on the left shows how comments come in from Chimera, while the one on the right shows the "Chat and Command" tool.

Comments from Chimera

Chimera, the name for the web version of your book, allows readers and collaborators to leave comments at the paragraph level on your book. The goal of these comments is to help you both better engage with your audience and get feedback to improve your book. (See Chapter 7 for more information on how to use Chimera and Atlas to engage readers.)

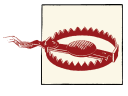
Comments left in Chimera will flow into the social sidebar so that you can more easily address whatever issue is they raise. Click on a comment to jump to the corresponding paragraph in Atlas.

Once you've addressed the comment, you have the option to delete it by clicking the small "x" to the comment's right.

Chat and Commands

In addition to managing comments from Chimera, you can also use the sidebar to chat with collaborators and issue various commands for Atlas to perform.

To chat with collaborators, simply type your message in the chat box and it will be broadcast to everyone else viewing the project. The goal of the chat is to allow you to more easily collaborate in real time with your co-authors and O'Reilly staff. For example, you might schedule a review session with your editor or tech reviewer to work on the project together using chat.

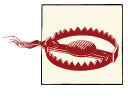


Live, multiuser editing of documents is not yet supported in the Atlas Web interface. Multiple users should avoid editing the same document at once, as they may overwrite each other's changes when they save.

In addition to chatting with collaborators, you can also chat with the Atlas chatbot to perform various tasks. The following table summarizes the things you can currently do with the bot:

Command	Description	Example
calculate	Do calculations with Google	atlas calculate square root of 400
convert	Convert units with Google	atlas convert 70 mph to kph
howdoi	Search stack overflow	atlas howdoi reverse a ruby array?
html5ref	Examples of HTML5 tags	atlas html5ref video
latexmath	Render an equation	atlas latexmath $a^2+b2=c^2$
shorten	Shorten a link using bit.ly	atlas shorten *http://www.example.com/big/long/path*

In addition, Atlas can send you notifications, as well. For example, it will send you a message when your build completes, which has been a much requested feature.



Atlas does not currently support private chat, so everything you do will be visible to all other people on the project. This includes both your requests and responses for the Atlas chatbot.

We'll be adding a lot more functionality to the atlas bot over the coming months, so stay tuned for more here. If you have suggestions for new features, post them in the "Feedback" tab.

PS — The Atlas chatbot is based on the GitHub's awesome Hubot project (*http://hubot.github.com/*). If you're familiar with that project and would like to contribute a script, let us know.

Index

We'd like to hear your suggestions for improving our indexes. Send email to index@oreilly.com.