PUBLISHING MADE EASY

Getting Started with Atlas

O'Reilly Media, Inc.

O'REILLY®

Getting Started with Atlas

O'Reilly Media

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Getting Started with Atlas

by O'Reilly Media

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: Adam Witwer

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

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

Contents

QuickStart Guide

Why is Atlas great? How can I get up and running in 5 minutes or less? Read on!

Why Is Atlas Great?

- Text to come!

Signing Up and Logging In

- Text to come!

Creating a New project

- Go to the Your Books tab and click "New Book". Enter the Title, click the button, and your book is ready to go.
- Add files by "Open Editor" on the Dashboard, or click the Write tab. If you haven't created any files yet, you'll be prompted to do so before you start writing.
- To create new files, click the Switch File button at the top right of the editing pane, and choose "new File" from the bottom.
- Use the formatting buttons along the top to add asciidoc markup to your text, as well as insert images, videos, 3D models, and more.
- Invite Collabortaors by clicking the Collaborators button in the toolbar, and entering the email address of the person you'd like to add.

Building a Project

- Text to come!

Publishing to Chimera

- Text to come!

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.

You can find more details and instructions in the written documentation.

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 and ebook builds
: Atlas lets you build your project in three formats at any time: Mobi (for the Kindle), EPUB (for most other ebook platforms), and PDF. 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 alpha 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!

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 mail 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.

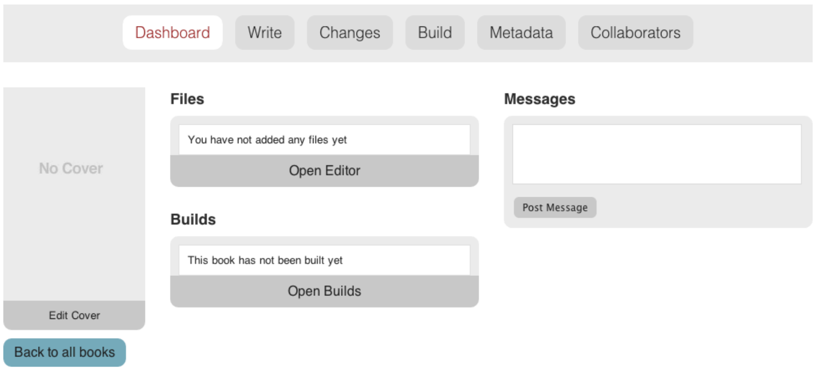


Figure 2-1. *When you create a new book in Atlas, you start off with a blank slate*

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.

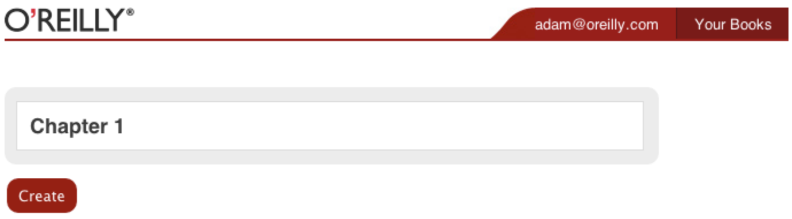


Figure 2-2. *Create your first file*

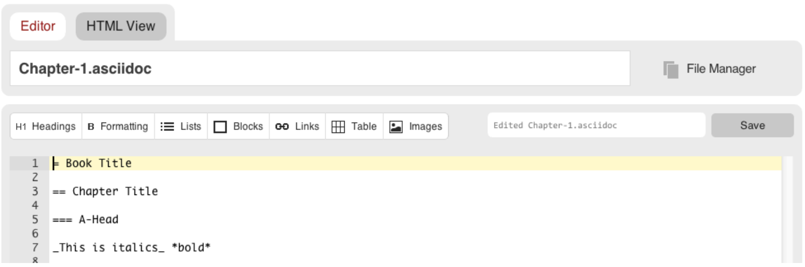
Once you've created the file, you're ready to start writing!

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.

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



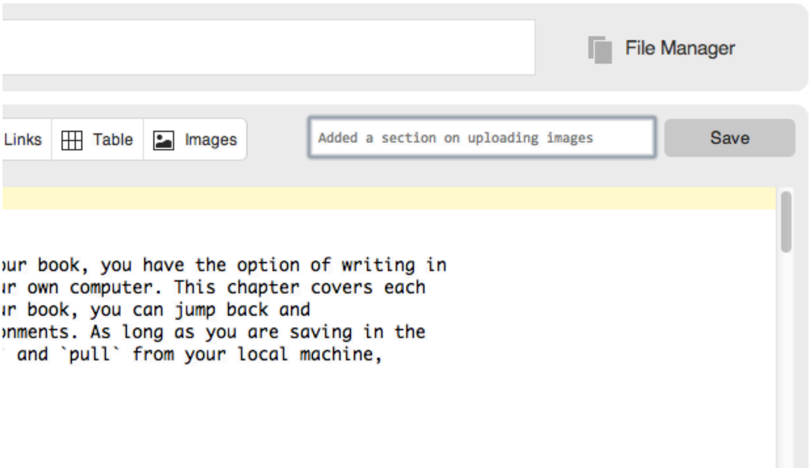
Chapter-1.asciidoc

Edited Chapter-1.asciidoc

```
= Book Title

== Chapter Title

=== A-Head

_This is italics_ *bold*
```

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" (page 10) and Chapter 6.



Added a section on uploading images

```
ur book, you have the option of writing in
ır own computer. This chapter covers each
ır book, you can jump back and
onments. As long as you are saving in the
 and `pull` from your local machine,
```

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.

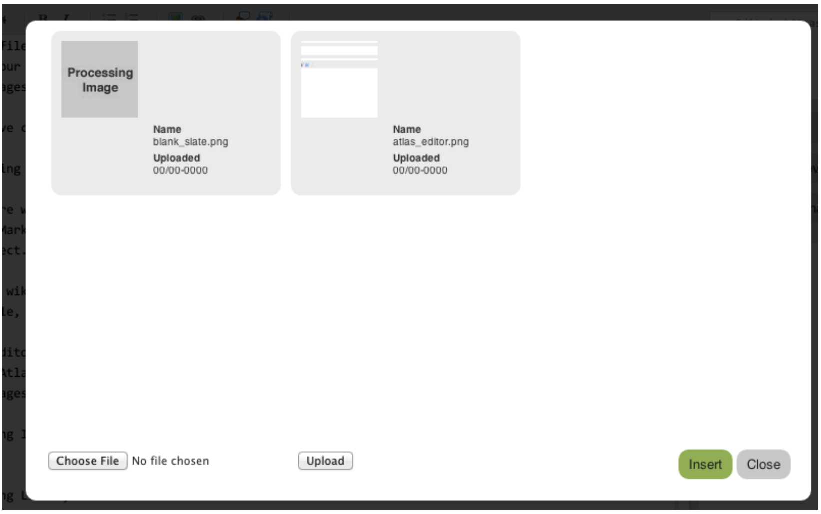


Figure 2-5. *Upload your images within Atlas*

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).

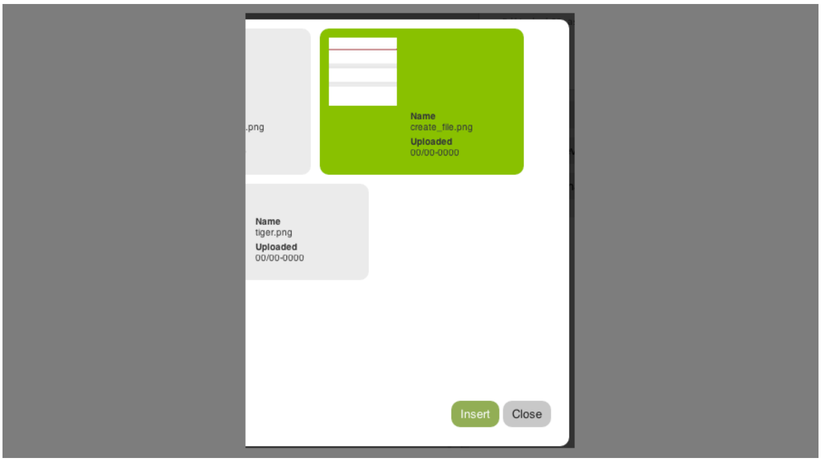


Figure 2-6. *You've uploaded the image; now reference it within your document*

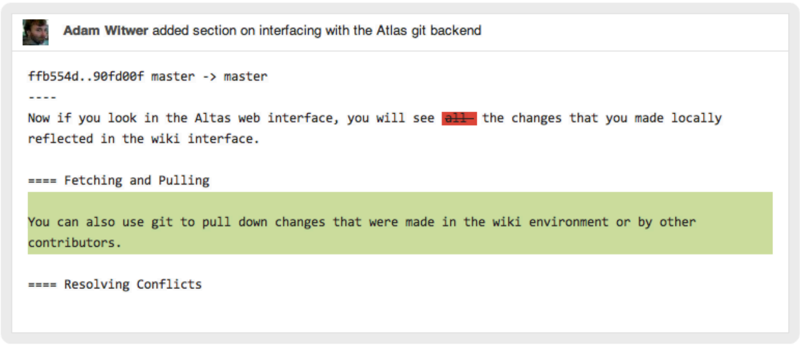
Once you click insert, Atlas will add AsciiDoc markup that looks something like the following to your document:

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

Read more about figure and image markup in "Figures and Other Images" (page 26) 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" (page 11)), Atlas displays that change on the Changes tab.


Adam Witwer added section on interfacing with the Atlas git backend

```
ffb554d..90fd00f master -> master
----
Now if you look in the Altas web interface, you will see ~~all~~ 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.

==== Resolving Conflicts
```

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" (page 6), the change would be captured and displayed in the same way.

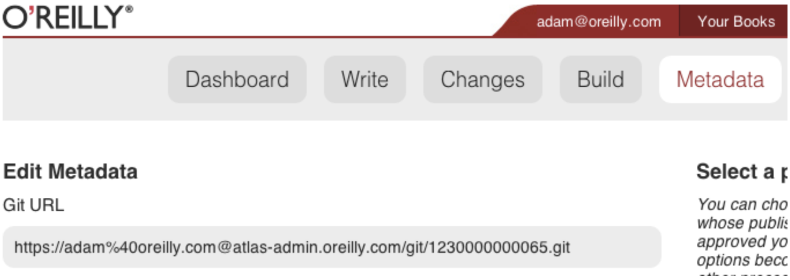
Working Locally

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 6 for more details around using git with Atlas.

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



https://adam%40oreilly.com@atlas-admin.oreilly.com/git/1230000000065.git

You can cho
whose publis
approved yo
options becc

Figure 2-8. *Grab your git repository URL from the Metadata 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:

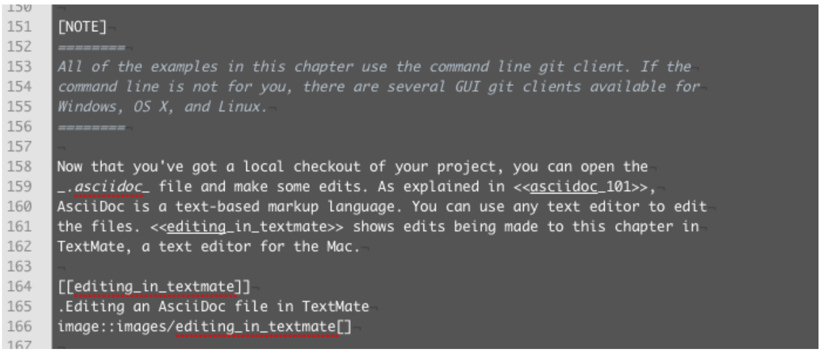
```
$    git    clone    https://adam%40oreilly.com@atlas-admin.oreilly.com/git/
1230000000065.git
   getting_started_with_atlas/

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.



```
[NOTE]
========
All of the examples in this chapter 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.
========

Now that you've got a local checkout of your project, you can open the
_.asciidoc_ file and make some edits. As explained in <<asciidoc_101>>,
AsciiDoc is a text-based markup language. You can use any text editor to edit
the files. <<editing_in_textmate>> shows edits being made to this chapter in
TextMate, a text editor for the Mac.

[[editing_in_textmate]]
.Editing an AsciiDoc file in TextMate
image::images/editing_in_textmate[]
```

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 6 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.

3/Building and Debugging

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

Three Major Ebook Formats

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

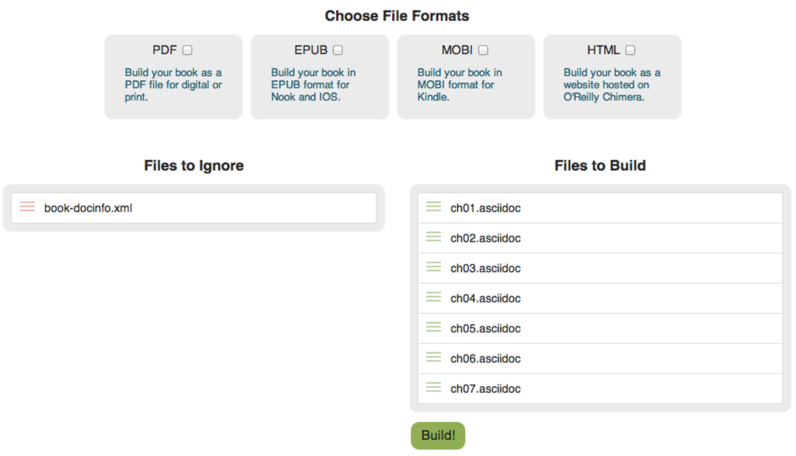
- The fixed-layout PDF format is undoubtedly familiar to you and remains the most popular format among customers who buy direct from O'Reilly. If your book ends up being printed, we will use a slightly modified version of the online-friendly PDF that Atlas outputs.
- EPUB uses a fluid layout, which means 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" (page 17).



Choose File Formats

PDF ☐ Build your book as a PDF file for digital or print.

EPUB ☐ Build your book in EPUB format for Nook and IOS.

MOBI ☐ Build your book in MOBI format for Kindle.

HTML ☐ Build your book as a website hosted on O'Reilly Chimera.

Files to Ignore

- book-docinfo.xml

Files to Build

- ch01.asciidoc
- ch02.asciidoc
- ch03.asciidoc
- ch04.asciidoc
- ch05.asciidoc
- ch06.asciidoc
- ch07.asciidoc

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" (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 6), and then adding the appropriate metadata using XML tags. Here is an example showing the metadata that you might like to include:

```xml
<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.

4/AsciiDoc 101

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 tools, including Atlas. It's similar to wiki markup—if you can write a Wikipedia article, then you'll have no problem with AsciiDoc. 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.

You can create and edit AsciiDoc in any text editor, and then add it to Atlas by either pasting it into the wiki interface or using git. If you're on Windows, you can use Notepad++ or any other text editor. If you're on a Mac, you can use TextEdit, TextMate (which you can pair with an AsciiDoc bundle), TextWrangler, or any of a number of choices. Or maybe you prefer old school vi or Emacs. That's okay, too. The important thing is that you use the AsciiDoc markup.

This chapter illustrates some of the most common elements used in writing technical documentation.

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, you should view the source, available at cdn.oreilly.com/atlas_docs.zip.

Chapters

Each chapter 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. Here's the markup:

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

Top-Level Section

Within the chapter, this is the first and highest heading level. This markup designates AsciiDoc level 2 (equivalent to DocBook `<sect1>`). The markup looks like this:

```
=== Top-Level Section
```

Second-Level Section

This heading level should only follow a top-level heading. It designates AsciiDoc level 3 (equivalent to DocBook `<sect2>`). The markup looks like this:

```
==== Second-Level Section
```

Third-Level Section

This heading level should only follow a second-level heading. It designates AsciiDoc level 4 (equivalent to DocBook `<sect3>`). The markup looks like this:

```
===== Third-Level Section
```

Divide the Book into 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.

Here's the markup:

```
[[unique_part_id]]
= Part Title

[[unique_chapter_id]]
== Chapter Title

Chapter text begins here.
```

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

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

Add a Preface

To add a preface, use this markup at the top of your preface file:

```
[preface]
== Preface Title

Preface text begins here.
```

Add a Dedication

To add a dedication, use this markup at the top of your dedication file:

```
[dedication]
= Dedication

Dedication text begins here.
```

Inline Elements

Here's some *italic*, `monospaced` (aka "constant-width" or "CW"), **bold**, **`CW bold`**, $^{\text{super}}$, and $_{\text{sub}}$ text.

Standard O'Reilly font conventions are as follows. The correct asciidoc markup for each is shown below.

*_Italic_*
: 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.

Bold*
: A general-purpose tag provided for where you would use bold type to emphasize a word or phrase. (Note that O'Reilly house style prefers italics for emphasis.)

`+Constant width+`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`*+Constant width bold+*`
: Used to show commands or other text that should be typed literally by the user. Note that the asterisk (*) pair must be outside and the plus-symbol (+) pair inside.

`pass:[<replaceable>Constant width italic</replaceable>]`
: Indicates text that should be replaced with user-supplied values or by values determined by context. This inline element markup uses a docbook passthrough.

*^Superscript^*
: Superscript text.

~Subscript~
: Subscript text.

Hyperlinks
: For hyperlinks to external sources, use the following markup. The text inside the brackets will appear as a clickable link in online versions.

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

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

Cross-References

To generate a cross-reference, use this syntax:

```
<<ID>>
```

where `ID` is the anchor or BlockID of the target, which you place in double square-brackets above that block.

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 online versions. Any time you refer to another component of your book, please be sure to use xref markup, not hardcoded text.

Table 4-1 shows the standard text generated from xrefs in PDF builds.

Table 4-1. *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 "Block Elements" (page 24) for details.
- The results is shown in Figure 4-1.
- Flip ahead to Chapter 5 for a preview.

generated from this source:

```
* See <<BLOCKS>> for details.
* The results is shown in <<FIG1>>.
* Flip ahead to <<advanced_asciidoc>> for a preview.
```

Please do not use AsciiDoc's optional `xreflabel` and `caption` features on anchors and xrefs, as these interfere with standard generated xref formats.

Indexing

We've extended indexing in AsciiDoc to include ranges, sees, and see alsos.

Using the syntax below, you can insert index markers anywhere in your text. To include an index in your book, you must also include an *index.asciidoc* file with a single header:

```
= Index
```

See the AsciiDoc source of this documentation for an example.

The complete list of indexing syntax follows.

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")))
```

A "(see also)" entry:

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

Changing how an entry is alphabetized:

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

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, e.g.:

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

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.

Block Elements

Sidebars

Sidebar markup looks like this:

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

Sidebars render like this:

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. One of the primary appeals of AsciiDoc is that it was created to export to DocBook XML. In other words, for each AsciiDoc markup syntax, there is an equivalent DocBook element. When you build your book (as described in Chapter 3), Atlas converts the AsciiDoc to DocBook and then generates the book formats from that DocBook. This series of magical transformations is part of what makes writing in Atlas fun, but it also requires you to use the correct AsciiDoc markup for things to work right.

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.

Admonitions (Notes and Warnings)

Here are some admonitions:

O'Reilly books traditionally make no visual distinction between the DocBook `<note>`, `<tip>`, and `<important>` elements.

Titled Tip
We do support optional titles in admonitions (in most series).

Titled Warning
O'Reilly Animal books traditionally make no visual distinction between the DocBook `<warning>` and `<caution>` elements.

This one is a `<caution>`.

Figures and Other Images

Below this paragraph is Figure 4-1 (a figure, titled and cross-referenced). 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:

```
[[FIG1]]
.A Figure
image::images/tiger.png[float="true"]
```

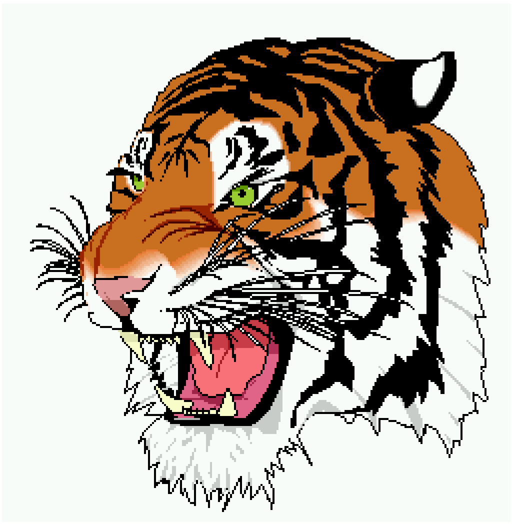
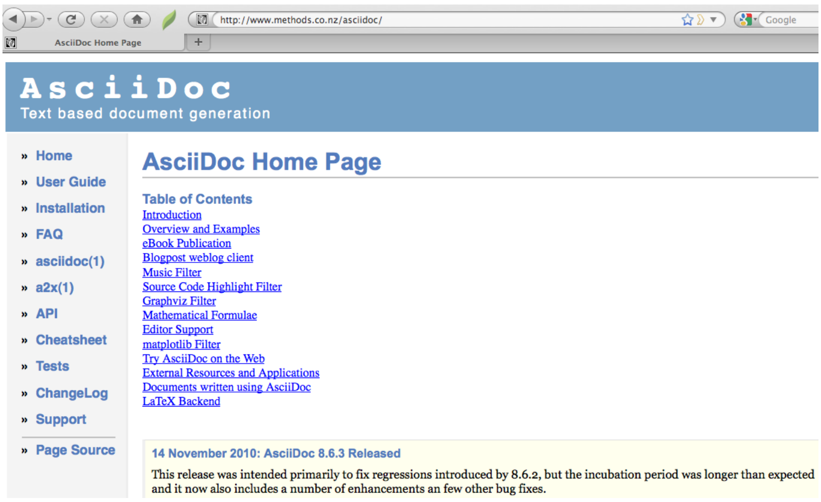


Figure 4-1. *A Figure*

Or you may prefer an image with no caption, like so:


http://www.methods.co.nz/asciidoc/

AsciiDoc Home Page

AsciiDoc
Text based document generation

» Home
» User Guide
» Installation
» FAQ
» asciidoc(1)
» a2x(1)
» API
» Cheatsheet
» Tests
» ChangeLog
» Support
» Page Source

AsciiDoc Home Page

Table of Contents
Introduction
Overview and Examples
eBook Publication
Blogpost weblog client
Music Filter
Source Code Highlight Filter
Graphviz Filter
Mathematical Formulae
Editor Support
matplotlib Filter
Try AsciiDoc on the Web
External Resources and Applications
Documents written using AsciiDoc
LaTeX Backend

14 November 2010: AsciiDoc 8.6.3 Released
This release was intended primarily to fix regressions introduced by 8.6.2, but the incubation period was longer than expected and it now also includes a number of enhancements an few other bug fixes.

Adding Alt Text to Images

To improve accessibility in your ebook files, please consider adding alt text to the images, like so:

```
[[FIG1]]
.A Figure
image::images/tiger.png["An image of a cartoonish tiger head"]
```

Or combine it with a `float="true"`:

```
[[FIG1]]
.A Figure
[float="true"]
image::images/tiger.png["An image of a cartoonish tiger head"]
```

Using AsciiDoc to Size Images

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

Labeled (aka Variable or Term-Definition) Lists

Term 1
: Definition/description

Term 2
: Something else

Bulleted (aka Itemized) Lists

- lions
- tigers
  - sabre-toothed
  - teapotted
- Lions, tigers, and bears.

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

Ordered (aka Numbered) Lists

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 a paragraph to a list item.

Tables

Atlas table styles vary slightly between series. If your material warrants something other than the default style as shown in Table 4-2, please consult with your editor.

The Asciidoc markup for tables looks like this:

```
.A Table
[width="50%",options="header"]
|=======
|P|Q|P^Q
|T|T|T
|T|F|F
|F|T|F
|F|F|F
|=======
```

Depending on the particular series your book is in, it will render similar to this:

Table 4-2. *A 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. Here's an example of the markup:

```
----
Hello world!

0         10        20        30        40
1234567890123456789012345678901234567890
----
```

Which will render like this:

```
Hello world!

0         10        20        30        40
1234567890123456789012345678901234567890
```

Formal code blocks (titled and cross-referenced) use the following markup:

```
[[EX1]]
.An Example
====
----
Hello world!

0         10        20        30        40
1234567890123456789012345678901234567890
----
====
```

And here's how it renders:

Example 4-1. An Example

```
Hello world!

0         10        20        30        40
1234567890123456789012345678901234567890
```

Inline Formatting Within Code

In AsciiDoc, there is no built-in mechanism for inline formatting within code. If you want to use inline formatting—in particular, for standard O'Reilly font 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" (page 37) 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
```

And here's the markup to use a passthrough with the `<programlisting>` and `<lineannotation>` 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 you want to be syntax-highlighted, and specify the language of the code. For example, the following code:

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

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

```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:

- Highlighting will not be applied to any code that has inline markup (as described in "Inline Formatting Within Code" (page 30)), even if `[source]` is added above the code block.
- The color scheme cannot be changed at this time.
- 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.

Text-only External Code File (Asciidoc markup)

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" (page 32).

External Code File with Inline Markup (Docbook markup)

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[]
++++
```

And then the contents of inline_markup.txt would contain only the Docbook markup. In our example, it 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>
```

and it will render like this:

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

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).

The built-in AsciiDoc mechanism (shown below) is somewhat more limited; for one thing, icons are hyperlinked from text to code, but not vice versa. However, you can always use a passthrough block (see "Passthroughs" (page 37)) for full functionality.

If you have a need to refer to the same bit of explanatory text from more than one line of code, see "Yet More on Code Callouts" (page 39) (in Chapter 5).

AsciiDoc Code Callouts

Here's the Asciidoc markup for code callouts:

```
----
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."
```

which renders like this:

```
Roses are red,
   Violets are blue. ❶
Some poems rhyme;
   This one doesn't. ❷
```

❶ Violets actually have a color value of `#9933cc`.

❷ This poem uses the literary device known as a "surprise ending."

Asciidoc External Code File with Callouts

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

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

The corresponding callout markers should be present in the external code file. In our example, the external HelloWorld.java file contents would look like this:

```
package com.marakana;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorld extends Activity {  <1>
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);  <2>
    setContentView(R.layout.main);
  }
}
```

And it would render like this:

```
package com.marakana;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorld extends Activity {  ❶
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);  ❷
                setContentView(R.layout.main);
        }
}
```

❶ This is number one.

❷ This is number two.

Docbook Passthroughs with Callouts

You can use Docbook passthroughs for your code if you'd like to have bidirectional callout links, or if you have inline formatting in your code. The markup for Docbook passthrough callouts is mostly the same whether you're writing the markup directly in the Asciidoc chapter file or writing the markup in an external file.

If you're adding the passthrough directly in the Asciidoc chapter file, the markup looks like this:

```
++++
<programlisting>Roses are <replaceable>red</replaceable>,
      Violets  are  <replaceable>blue</replaceable>.  <co  id="violets-co"
linkends="violets"/>
Some poems rhyme;
        This    one   <userinput>doesn't</userinput>.   <co   id="poem-co"
linkends="poem"/></programlisting>

<calloutlist>
  <callout id="violets" arearefs="violets-co">
    <para>Violets actually have a color value of +#9933cc+.</para>
  </callout>
  <callout id="poem" arearefs="poem-co">
     <para>This poem uses the literary device known as a &ldquo;surprise
```

```
ending.&rdquo;</para>
  </callout>
</calloutlist>
++++
```

If using passthroughs for callouts, please use the paired markup as shown above (not `<areaspec>`).

If you have your code in an external file, your markup will look like this:

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

And the contents of your external code file (inline_markup_with_callouts.txt, in our example here) will look something like this:

```
<programlisting>Roses are <replaceable>red</replaceable>,
      Violets  are  <replaceable>blue</replaceable>.  <co  id="violets-co"
linkends="violets"/>
Some poems rhyme;
   This one doesn't. <co id="poem-co" linkends="poem"/></programlisting>

<calloutlist>
  <callout id="violets" arearefs="violets-co">
    <para>Violets actually have a color value of +#9933cc+.</para>
  </callout>
  <callout id="poem" arearefs="poem-co">
     <para>This poem uses the literary device known as a &ldquo;surprise
ending.&rdquo;</para>
  </callout>
</calloutlist>
```

which will render like this:

```
Roses are red,
   Violets are blue. ❶
Some poems rhyme;
   This one doesn't. ❷
```

❶ Violets actually have a color value of +#9933cc+.

❷ This poem uses the literary device known as a “surprise ending.”

Other Block Elements

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>
++++
```

Quotes

To add a quote block to your text, use the following markup. Here's 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.
____
```

which renders like this:

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

5/Advanced AsciiDoc

This is another chapter, with some more esoteric/experimental things…

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: `pass through goes here`. See “Challenging Inlines” (page 38) 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="FIG1" float="none">
<title>A Figure</title>
<mediaobject>
  <imageobject>
  <imagedata fileref="images/tiger.png"/>
  </imageobject>
</mediaobject>
</figure>
----
```

Using the block passthrough method described above, you could potentially pass through an entire chapter of DocBook.

Underlined vs. Delimited

Section titles can be in either of two formats: “underlined” or delimited.

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 Section Title (Underlined Style)

Narrative text here.

Second-Level Section Title (Underlined Style)

More text here.

Third-Level Section Title (Underlined Style)

And some more here.

Challenging Inlines

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*
- Attempting to nest inlines within each other may result in invalid DocBook (because the content model of one may not include the other). For example, **`foo`** requires the asterisk (*) pair outside and the plus-symbol (+) pair inside. If you try the other way around, the build will likely fail with this error message:

  ```
  Element emphasis is not declared in literal list of possible children
  ```

- For more precise semantics, you can always use "passthroughs" to embed DocBook markup as is, e.g. *`foo`*, **`foo`**, and ***`foo`*** are fairly common in O'Reilly books, rendered as CWI, CWB, and CWBI, respectively (constant-width with italic, bold, and bold+italic).

For generic emphasis, you can also surround a string with single, straight quotation marks (apostrophes): i.e., *this* is equivalent to *that*. 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 */path/to/file.ext* or */path/to/file.ext*
- for a book title, use *This Book Needs No Title* or *This Book Needs No Title*

To get curly quotes you can use AsciiDoc's mechanism for 'singles' and "doubles" (or you can always use Unicode like so: "quote me").

Subscripts and superscripts work like so: H_2O and $2^5 = 32$ (but if you're doing math, you'd probably want to italicize the variables, like so: $x^2 + y^2 = z^2$).

If using footnotes, please place the macro [1] directly after the text, with no space between (otherwise you'll introduce extra space [2] before the in-text mark); footnotes at the end of a sentence belong after the period.[3]

Yet More on Code Callouts

If you want to refer to the same explanation from more than one line of code, please *do not* use the built-in mechanism (Example 5-1), which does not conform to house style. Instead, use either Example 5-2 or Example 5-3.

Example 5-1. Built-in Mechanism (Vertical Layout Bad)

```
10/17/97   9:04         <DIR>    bin
10/16/97  14:11         <DIR>    DOS            ❶
10/16/97  14:40         <DIR>    Program Files
10/16/97  14:46         <DIR>    TEMP
10/17/97   9:04         <DIR>    tmp
10/16/97  14:37         <DIR>    WINNT
10/16/97  14:25           119    AUTOEXEC.BAT   ❷
 2/13/94   6:21        54,619    COMMAND.COM    ❸
10/16/97  14:25           115    CONFIG.SYS     ❹
11/16/97  17:17    61,865,984    pagefile.sys
 2/13/94   6:21         9,349    WINA20.386     ❺
```

❶ This directory holds MS-DOS.

❷ ❸ ❹ System startup code for DOS. **[Too much whitespace; confusing to readers]**

❺ Some sort of Windows 3.1 hack.

Example 5-2 uses the `<coref>` element to refer to the same `<callout>` from multiple code lines.

1. This is a standard footnote.
2. This one has extraneous space before the in-text mark (although the footnote itself is fine).
3. If your footnote text includes [square brackets], you can escape them with a pass-through macro.

Example 5-2. Alternate Approach 1: Repeat Icon in Code

```
10/17/97   9:04         <DIR>    bin
10/16/97  14:11         <DIR>    DOS      ❶
10/16/97  14:40         <DIR>    Program Files
10/16/97  14:46         <DIR>    TEMP
10/17/97   9:04         <DIR>    tmp
10/16/97  14:37         <DIR>    WINNT
10/16/97  14:25             119  AUTOEXEC.BAT   ❷
 2/13/94   6:21          54,619  COMMAND.COM    ❷
10/16/97  14:25             115  CONFIG.SYS     ❷
11/16/97  17:17      61,865,984  pagefile.sys
 2/13/94   6:21           9,349  WINA20.386     ❸
```

❶ This directory holds MS-DOS.

❷ System startup code for DOS.

❸ Some sort of Windows 3.1 hack.

Example 5-3 uses a different mechanism for several code lines to point to the same `<callout>`. In this case, each one gets a uniquely numbered icon. This is done by placing multiple values in a single `<callout arearefs=…>` while duplicating the `@linkends` value in the corresponding `<co>` elements in code. Note also the use of `<?dbfo…>` markup below the `<calloutlist>` opener; this adjusts the alignment for side-by-side icons.

Example 5-3. Alternate Approach 2: Unique Icons; Align Side-by-Side in Explanation

```
10/17/97   9:04         <DIR>    bin
10/16/97  14:11         <DIR>    DOS       ❶
10/16/97  14:40         <DIR>    Program Files
10/16/97  14:46         <DIR>    TEMP
10/17/97   9:04         <DIR>    tmp
10/16/97  14:37         <DIR>    WINNT
10/16/97  14:25             119  AUTOEXEC.BAT   ❷
 2/13/94   6:21          54,619  COMMAND.COM    ❸
10/16/97  14:25             115  CONFIG.SYS
11/16/97  17:17      61,865,984  pagefile.sys
 2/13/94   6:21           9,349  WINA20.386     ❹
```

❶ This directory holds MS-DOS.

❷❸ System startup code for DOS.

❹ Some sort of Windows 3.1 hack.

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.

Controlling Line Breaks

Use an inline passthrough to prevent a line break:

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

Everything Else

Finally, keep in mind that there may be situations where it's hard to get AsciiDoc to format something the way you want. Sometimes there's a trick to get around it; sometimes it's better to use a passthrough block to embed a bit of DocBook; and sometimes there may be a different formatting approach that will mesh better with our production systems. If you need to do something not illustrated in this guide, please check with your editor.

6/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 6-1 summarizes some commonly used commands.

Table 6-1. *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 Metadata 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  "Ascii
Doc.tmbundle"
$ osascript -e 'tell app "TextMate" to reload bundles'
```

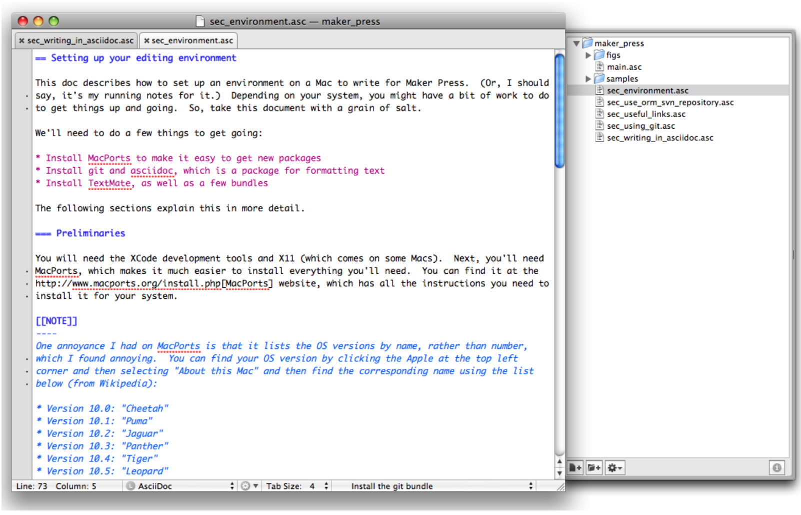
Now the the bundle is installed, your AsciiDoc markup will have all the color-coded goodness you've come to expect in a text editors.

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 6-1.

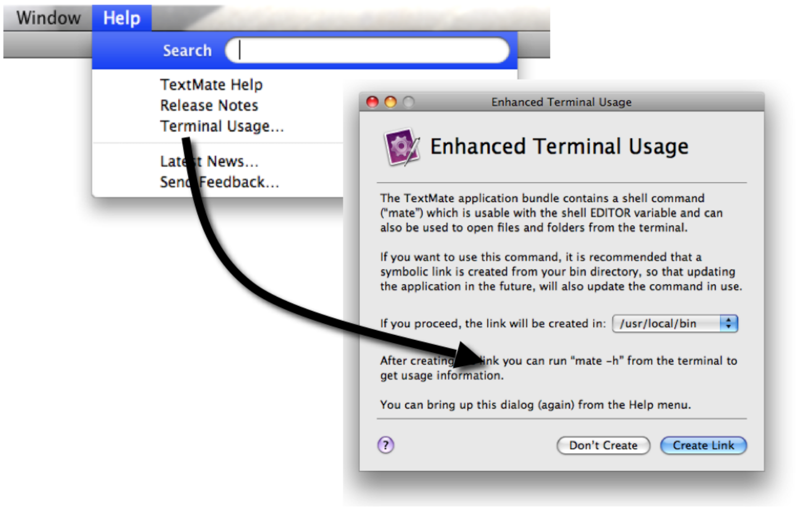


```
== Setting up your editing environment

This doc describes how to set up an environment on a Mac to write for Maker Press.  (Or, I should
say, it's my running notes for it.)  Depending on your system, you might have a bit of work to do
to get things up and going.  So, take this document with a grain of salt.

We'll need to do a few things to get going:

* Install MacPorts to make it easy to get new packages
* Install git and asciidoc, which is a package for formatting text
* Install TextMate, as well as a few bundles

The following sections explain this in more detail.

=== Preliminaries

You will need the XCode development tools and X11 (which comes on some Macs).  Next, you'll need
MacPorts, which makes it much easier to install everything you'll need.  You can find it at the
http://www.macports.org/install.php[MacPorts] website, which has all the instructions you need to
install it for your system.

[[NOTE]]
----
One annoyance I had on MacPorts is that it lists the OS versions by name, rather than number,
which I found annoying.  You can find your OS version by clicking the Apple at the top left
corner and then selecting "About this Mac" and then find the corresponding name using the list
below (from Wikipedia):

* Version 10.0: "Cheetah"
* Version 10.1: "Puma"
* Version 10.2: "Jaguar"
* Version 10.3: "Panther"
* Version 10.4: "Tiger"
* Version 10.5: "Leopard"
```

Figure 6-1. *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 6-2 shows how this works.



Enhanced Terminal Usage

The TextMate application bundle contains a shell command ("mate") which is usable with the shell EDITOR variable and can also be used to open files and folders from the terminal.

If you want to use this command, it is recommended that a symbolic link is created from your bin directory, so that updating the application in the future, will also update the command in use.

If you proceed, the link will be created in: /usr/local/bin

After creating the link you can run "mate -h" from the terminal to get usage information.

You can bring up this dialog (again) from the Help menu.

Figure 6-2. *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 _<file name>_` 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:

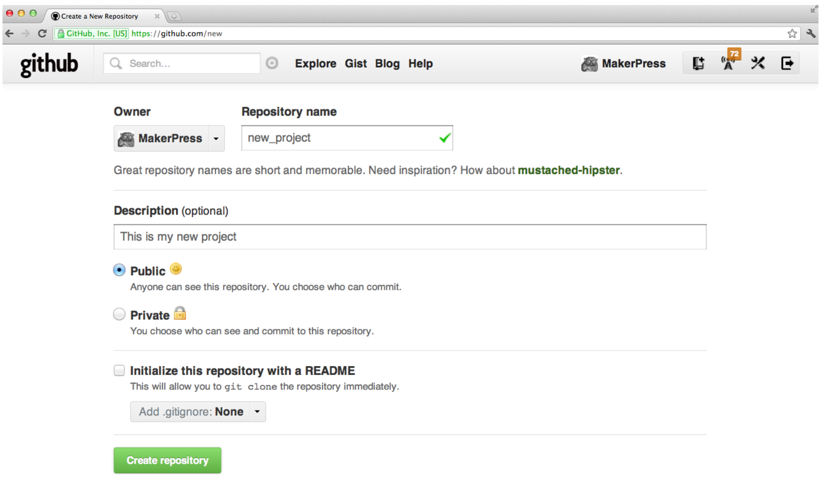
```
$ git merge origin/master
Updating 92d4f99..e9fbbe2
Fast-forward
 ch01.asciidoc |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

7/Working with GitHub

Because book files in Atlas are managed with git, you could easily sync those files with a repository on GitHub. This chapter describes that process.

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 7-1.



Owner: MakerPress
Repository name: new_project

Great repository names are short and memorable. Need inspiration? How about **mustached-hipster**.

Description (optional): This is my new project

● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately.

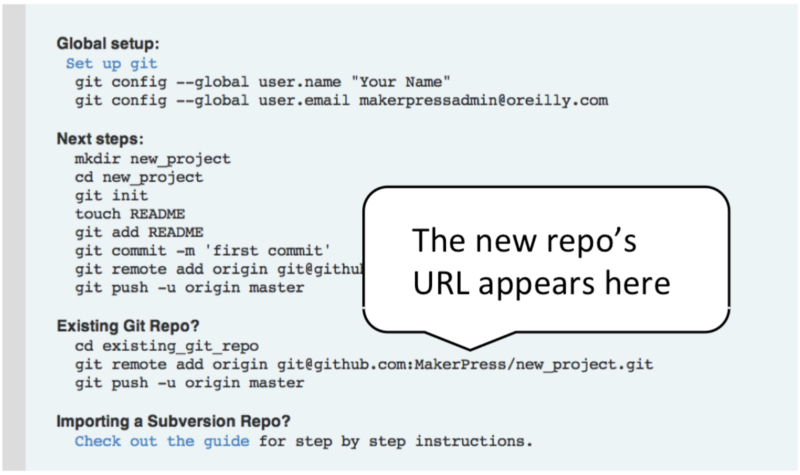
Add .gitignore: **None**

Figure 7-1. *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 7-2.



```
Global setup:
 Set up git
  git config --global user.name "Your Name"
  git config --global user.email makerpressadmin@oreilly.com

Next steps:
  mkdir new_project
  cd new_project
  git init
  touch README
  git add README
  git commit -m 'first commit'
  git remote add origin git@githu
  git push -u origin master

Existing Git Repo?
  cd existing_git_repo
  git remote add origin git@github.com:MakerPress/new_project.git
  git push -u origin master

Importing a Subversion Repo?
  Check out the guide for step by step instructions.
```

Figure 7-2. *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-
```

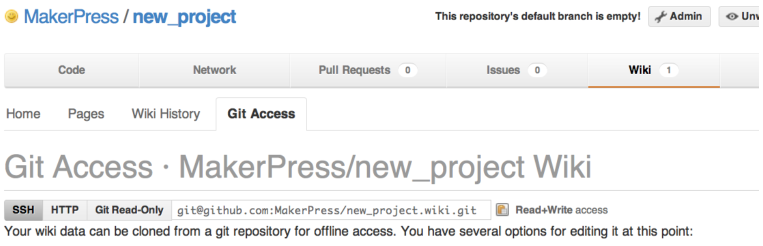
```
admin.oreilly.com/git/1230000000197.git
admins-MacBook-Air-2:new_project odewahn
$ git push atlas master
...
```

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:



Git Access · MakerPress/new_project Wiki

SSH HTTP Git Read-Only git@github.com:MakerPress/new_project.wiki.git Read+Write access

Your wiki data can be cloned from a git repository for offline access. You have several options for editing it at this point:

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.

Index

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