.htaccess
made easy

Learn how to redirect stuff.

Fine-tune site configuration.

Improve site performance.

Strengthen site-security.

Enhance usability.

.htaccess
made easy

A PRACTICAL GUIDE
FOR ADMINISTRATORS,
DESIGNERS & DEVELOPERS

by JEFF STARR

.htaccess made easy

perishablepress.com | monzilla.biz

For updates, purchase, or more info, visit:
htaccessbook.com

PUBLISHER
Perishable Press

EDITION
First Edition, Version 1.3, November 2012

LAYOUT & DESIGN
Jeff Starr

TYPEFACES
Calluna, designed by Jos Buivenga
Gill Sans, designed by Eric Gill
Monaco, designed by Susan Kare and Kris Holmes

THANKS & INSPIRATION
Thanks to God, my family, friends, teachers, peers, and everyone who helps along the way. This book is dedicated to my loving wife, Jennifer.

PDF FORMAT
ISBN-10: 0983517827
ISBN-13: 978-0-9835178-2-5

PRINT FORMAT
ISBN-10: 0983517835
ISBN-13: 978-0-9835178-3-2

contents

3.0 essential techniques

4.0 optimizing performance

5.0 improving SEO

6.0 .redirecting stuff

7.0 tighten security

8.0 enhance usability

9.0 .htaccess tricks for WordPress

10.0 even more techniques

Epilogue

Sections 2.7 and 10.3 are highlighted for quick reference.

httpd.conf sidebar menu

chapter 1

welcome

For websites hosted on Apache-powered servers, .htaccess is the perfect tool for a wide range of tasks. From protecting and redirecting pages to compressing and delivering content to specific browsers, .htaccess is both powerful and practical, enabling you to streamline, optimize and secure your website with ease.

.htaccess is often referred to as "voodoo" because it's not as well known as say, CSS, JavaScript, or even PHP. Written with a deceptively simple syntax, .htaccess enables admins and designers to customize core functionality involving how content is delivered and how traffic flows throughout your site. Just as CSS is meant for styling pages, .htaccess is meant for configuring and fine-tuning the server at the directory-level, giving you much control over the functionality of your site. Voodoo it's not, but rather .htaccess is easy enough that virtually anyone can benefit from its many practical uses.

Welcome to the footer! Watch this area throughout the book for notes, links & more. See section 2.3 for icon definitions and other conventions used in the book.

Just getting into web-design? Smashing Magazine is a must-read: http://smashingmagazine.com/

Another excellent resource for CSS and web-design is Chris Coyier's CSS-Tricks: http://css-tricks.com/

1.1 Is this book for you?

.htaccess made easy is for admins, designers, and developers out there in the trenches, busy making awesome sites every day. It's your one-stop, go-to guide for all things .htaccess. With over 100 handpicked recipes ranging from the practical to the extraordinary, this book brings it all together and delivers the best techniques with more signal and less noise.

.htaccess made easy is written for people who work with Apache-powered websites and want to make best use of .htaccess for stuff like redirecting traffic, optimizing for search engines, improving usability, and securing their sites against malicious scripts. Whether you're just getting started with web-design or have tons of coding experience, *.htaccess made easy* equips you with awesome techniques explained in clear, concise language.

1.2 Why htaccess?

Technically, .htaccess is part of something much larger, the Apache server language, which enables server-configuration at the directory-level via ".htaccess" files. Apache is free, open-source software that many web-hosts run on their Linux/Unix-based servers. It's server software, and the same directives that are used to configure and instruct the server are also available for use on a "per-directory" basis using Hypertext Access files, also known as "HTAccess" — or ".htaccess" — files. The ability to include .htaccess files in specific directories gives you more control of your site's configuration, optimization, and security.

There is a wealth of great .htaccess information on the Web, including some great sites to visit when working with specific techniques and implementations.

The official Apache Documentation is available at: http://httpd.apache.org/docs/

The .htaccess archive at Perishable Press contains many in-depth articles: http://perishablepress.com/

Another good site for all things Apache, including some great .htaccess techniques: http://www.askapache.com/

1.3 Goals of the book

Some folks may *cringe* at the thought of messing with .htaccess, and will search for another option — anything — to avoid meddling with any .htaccess voodoo. One of the main goals of this book is to help designers and developers understand what .htaccess is and when to use it (and when *not* to use it). Another important goal is to provide all of the code, techniques, and information required to get the job done. Need to redirect an entire website without losing any SEO rank? Done. Want to password-protect specific directories for specific users? Done. Need to prevent other sites from stealing your files and bandwidth? Done and done. And it goes far beyond the basics into some really advanced techniques like conditional file compression, query-string firewalls, blocking proxy visits, and much more. After years of working with .htaccess, I wrote this to be THE book that designers, developers, and admins reach for when working with .htaccess.

1.4 Now you're an .htaccess ninja

Well, not yet... but this book is your ninja-toolbelt equipped with .htaccess awesomeness. Just reach in, grab what you need, and you're on your way. It's a guidebook designed to make understanding and using .htaccess as simple as possible. The book begins with the basics, and then walks through (most) every technique. Along the way, each technique is explained in a simple, concise manner, and includes tips and links in the footer-area to provide additional materials, notes, and resources.

General rule of thumb: use .htaccess only when it's not possible to use Apache's main configuration file, `httpd.conf`. If in doubt, ask your web-host or read the Apache Docs about when (not) to use .htaccess: http://htaccessbook.com/79

For help with .htaccess, visit the book's exclusive Help Forum: http://htaccessbook.com/forum/

For help with Apache, .htaccess, and much more, check out: http://www.webmasterworld.com/apache/

1.5 Bonus material

This book includes the following awesome stuff:

- The book (in either print or PDF format)
- Access to the Members Area & Help Forum
- Free updates for the life of the book
- Modular site-maintenance pack
- .htaccess templates ▸

.htaccess templates!

- Universal starter template
- WordPress .htaccess template
- .htaccess/.htpasswd combo
- httpd.conf file (Apache 2.4.3)
- Blank .htaccess file (zipped)

Downloads available in the Members Area @ `http://htaccessbook.com/members/`

The .htaccess templates include inline comments so that it's easy to follow along. These are great files to begin learning the basics or experimenting with more advanced techniques. The site-maintenance pack is explained in section 6.3.

1.6 Questions, comments, and errata

This book is brought to you by... lil' ol' me. And although I've tried to be as careful and thorough as possible with every detail, improvements are always possible. So if you discover any errors or have questions or comments, please let me know. Or if you need help with code and techniques from the book, visit the .htaccess Forums and post your question in the appropriate category. Taking the time to provide feedback is worth it, especially because you get the updated versions for free after improvements have been made. It's a win-win.

Log in to visit the Member's Area: http://htaccessbook.com/members/

Get help with code in the .htaccess Forums: http://htaccessbook.com/forums/

Send errata, questions, and comments about the book directly via email: help@htaccessbook.com

chapter 2

the basics

There are some basics of working with .htaccess that you should understand before working with any code. Once you've got it, you're ready to roll, and that's what this chapter is all about.

There are *three keys* for working with .htaccess:

1. **Backup** your files before making any changes
2. **Apply** the right code for the right job
3. **Test** all changes thoroughly

Of these, it's up to you to make sure that you're backing up your files and testing things thoroughly, and it's up to me to provide the "right code for the job" in the pages of this book. So with these three keys in hand, let's look at a few other important tools and techniques for working with .htaccess that will ensure success when it's time to get in and get the job done.

Remember, only use .htaccess when it's not possible to use Apache's main configuration file. http://htaccessbook.com/79

Beginner's Guide to the .htaccess file: http://htaccessbook.com/5

See section 10.3 for help with diagnosing and resolving errors if things aren't going as planned.

Comprehensive guide to .htaccess: http://htaccessbook.com/6

2.1 Required skills

If you know how to open a file, copy/paste, and upload files via FTP to your web server, you're in business. I've been doing this stuff for years, and will be right there with you, explaining everything you need to do it right the first time. Yes working with .htaccess is mission-critical stuff, but easily implemented with the help of this book.

2.2 Required software

Assumptions: Linux/Unix-based server running Apache, which is like 90% of the web, but you should ask your web-host if unsure. It's also recommended that you test thoroughly before implementing new techniques on a live website. You can either set up a private domain for testing changes, or test things on your local machine before going live.

Other than that, you'll need something to edit text-based .htaccess files, and a good FTP client to upload, download, and just look cool in general. If you have access to your web-host's server control panel (e.g., cPanel, Plesk, or similar), that's another useful option, but not necessarily required.

Lastly, as you're working with web pages, you'll definitely want a good browser. I recommend Chrome, Firefox or Opera as my first-round draft pick, but there are many other good browsers available online.

Unless noted otherwise, the techniques in this book require Apache **2.0** and greater. The current version of Apache is **2.4**.

For more information on setting up and running Apache locally on your own computer, see section 2.9.

The Apache Software Foundation and Server Project: http://www.apache.org/ and http://httpd.apache.org/

Need a browser? Here's a great round-up with in-depth information and statistics: http://htaccessbook.com/8

2.3 Conventions used in this book

Everything should be clearly identified and explained as you read along. The structure of the book looks like this:

- **Chapter 1 and 2:** Introduction and basics
- **Chapter 3 - 9:** Treasure-trove of .htaccess techniques
- **Chapter 10:** Bonus tips and troubleshooting guide

Within each of the "technique" chapters, you'll get an overview of the contents and a quick-jump menu on the first page of each chapter. Then on each page, you'll get important links, notes and tips in the footer-area. Shown at right are the different icons that we'll be using, and how such references will appear throughout the book.

.htaccess notes and tips

Links to online resources

Apache notes and tips

Important information

Additionally, as you may have noticed by now, many of the book's hyperlinks are "shortened" to make it easier for those with the printed version to type the URLs into their browser. It also enables the inclusion of more links in the footer-area, without things looking too crowded or weird. When used, these shortened URLs will look like this: `http://htaccessbook.com/1`, where "1" is a sequence of alphanumeric characters that corresponds to a specific link on the Web.

The footer items are ordered from left to right, top to bottom, according to the order in which the reference dots appear in the main text.

If the icon is blue, the information pertains to Apache in general, such as the `httpd.conf` file.

For example, here's a shortcut link to "The Ultimate Guide to .htaccess Files": http://htaccessbook.com/9

The red icons contain important information that you should be aware of while working with .htaccess files.

2.4 About the .htaccess file

Unlike scripting languages such as PHP that may use many different files, working with .htaccess involves only one: the .htaccess file itself. The .htaccess file may be located in the root directory or any subdirectory of your site, but is not always visible by default.

.htaccess files are present on many server configurations, so working with them involves downloading from the server and editing with your favorite text-editor or syntax highlighter. I use Dreamweaver (Mac & PC), Coda (Mac), and most frequently TextEdit (Notepad on Windows), depending on the situation. It's important not to use MS Word or any other program that applies any sort of auto-formatting. You want to keep it plain-text all the way. If in doubt, stick with TextEdit/Notepad or similar and go from there.

If you don't see any .htaccess files on the server (or elsewhere), you'll need to create one. This can be tricky because of the way the file is named. For example, the "dot" at the beginning of the file name means the file is hidden by default on Mac and Linux, and can be difficult to manage on PC/Win as well. Once you do have an .htaccess file available (and not hidden) on your machine, it's trivial to duplicate the file for use in other directories.

Rather than go through the steps involved to create an .htaccess file, let's do it the easy way. Just visit the footer link to grab a zipped copy of the blank .htaccess file. Once downloaded, make sure you can view hidden files, and then unzip for a ready-to-go .htaccess file.

Excellent article on "How To Create & Edit The .htaccess File For Your Site": http://htaccessbook.com/a

Blank .htaccess file (1KB .zip download): http://htaccessbook.com/members/

You can rename ".htaccess" files to anything you'd like, including names that don't begin with a dot. See section 3.10 for more information on renaming .htaccess files.

Apache guide to .htaccess files: http://htaccessbook.com/b

2.5 How .htaccess files work

If you're familiar at all with how CSS rules operate, you'll see that .htaccess directives work in a similar way. In CSS, for example, if you apply a font-size rule of `12px` to the `<body>` element, it will be applied to everything contained within: paragraphs, headings, and other child elements of the body tag will display their text in 12px-size font. It's a cascading effect.

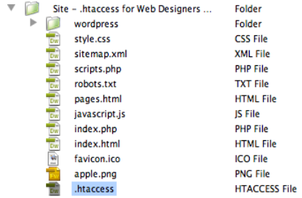
Looking at .htaccess rules, we see the same basic principle, only instead of applying to all child elements, .htaccess directives apply to all sub-directories. So for example, when you place an .htaccess file in the root directory of your site, the directives will be applied to everything contained therein. At right are screenshots and captions to further visualize the concept.

The ".htaccess cascade" flows down the directory structure, making it easy to apply directives to an entire site, as well as customize for any subdirectory along the way. And like CSS, .htaccess rules may be overridden by directives contained further downstream. After all, that is the purpose of .htaccess files in the first place — to enable per-directory configuration of your website.

.htaccess in root directory

When an .htaccess file is placed in the root directory of a site, its rules are applied to basically the entire site, meaning all of its subdirectories and files.*

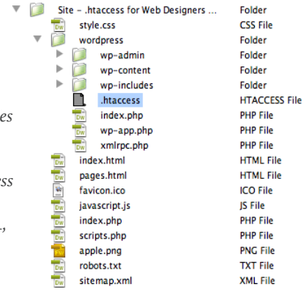
In this example, all root files are covered, as well as the entire WordPress installation.



.htaccess in subdirectory

If we move the .htaccess file to a subdirectory, such as `/wordpress/`, its rules are applied only to the files and folders contained therein.

In this example, only the wordpress directory and its contents are covered. The `style.css`, `index.html`, and other root files are not.



2.6 Basic structure and syntax

Okay, we're just about ready to dive into the techniques. Before doing so, let's look at the basic structure and syntax of the .htaccess file.

Structure

The structure of an .htaccess file is "open" in the sense that rulesets and single-line directives may be placed in any order in the file. As you can see in the "chapter-examples" file, the order of the different rulesets is generally unimportant. You can place stuff wherever you would like, however there are a few situations where playing with the order may get you different results. As we go through the book, I'll let you know whenever rule-order is important. 99.9% of the time you may organize things however you like.

Syntax

Just as with Apache's main configuration files, .htaccess files consist of single-line server directives. These single-line directives may work independently or together, for example:

W3C Cascading Style Sheets home page
http://htaccessbook.com/c

Apache HTTP Server Tutorial: .htaccess files
http://htaccessbook.com/b

In this book, and when dealing with .htaccess files, the "root directory" always refers to the web-accessible root directory, such as "`/httpdocs/`", "`/public_html/`", or "`/www/`". If in doubt, ask your web-host.

A potential downside to the .htaccess cascade is basically "how to stop it" from acting upon specific directories. For example, if you're protecting your site with .htaccess in the root directory, how do you disable protection for a 'public' directory? We'll see how to do it in section 7.3.

This book includes an .htaccess file with chapter-examples and inline-comments. Download here:
http://htaccessbook.com/members/

Apache guide to configuration files:
http://htaccessbook.com/e

Single-line directive (works independently)	Multi-directive ruleset (works collectively)

`RedirectMatch 301 / http://example.com/`	`RewriteBase /wordpress/`
	`RewriteRule ^index\.php$ - [L]`
	`RewriteCond %{REQUEST_FILENAME} !-f`
	`RewriteCond %{REQUEST_FILENAME} !-d`
	`RewriteRule . /wordpress/index.php [L]`

In the left column, we see a single directive that will redirect the entire site to "`example.com`". In the right column, we see the default set of directives for WordPress. Working together, these directives establish the "pretty-permalink" structure for WordPress URLs. So the "one-directive-per-line" rule is important, and we may combine directives for more advanced functionality. Additionally, you should keep the following things in mind:

Dealing with long lines — when working with directives that are many characters in length, you may use the backslash "\" as the last character on a line. This tells Apache that the directive continues on the next line. Here is an example:

```
RewriteCond %{QUERY_STRING} (benchmark|boot.ini|cast|declare|drop\
|echo.*kae|environ|etc/passwd|execute|input_file|insert|md5|mosconfig\
|scanner|select|set|union|update) [NC]
```

Here we see a long line of code split onto several lines using the backslash at the end of each line. As long as you're careful not to include any whitespace or other characters between the backslash and the end of the line, this is an excellent way to keep your .htaccess files easier to manage.

Case (in)sensitivity — for the most part, .htaccess directives are case-*insensitive*, but there are certain arguments and operators that are case-*sensitive*. If unsure, check the official Apache documentation.

Inline comments — when working with .htaccess, it's helpful (and encouraged) to leave descriptive comments along with your various directives. To do so, simply prepend a hash-symbol "#" to the beginning of the line, like so:

```
# this is a helpful comment
# that continues on this line
      # you may indent comments
      # whenever you would like
            # and as much as you'd like
```

Comments in .htaccess files must exist on their own line, and you may add as many comments as needed. If you place a comment *on the same line* as a directive, Apache will throw the dreaded 500 — Internal Server Error. You can indent your comments too, because Apache ignores any white space and blank lines that may appear before a directive.


See Chapter 6 for more info on redirecting files, directories, and entire sites.


Warning: precision is key when writing .htaccess directives. A single misplaced character will trigger a 500-status error.


For more information on WordPress permalinks, visit section 9.1. And for more, "The htaccess Rules for all WordPress Permalinks": http://htaccessbook.com/f


Apache HTTP Server Documentation http://httpd.apache.org/docs/


More about the 500 "Internal Server Error": http://htaccessbook.com/g


When writing comments, it's best to use only alphanumeric characters, underscores, and dashes. This helps to avoid any potential server parsing-errors.


Automatic htaccess file generator http://htaccessbook.com/7i

2.7 Character Definitions

This isn't an exhaustive list of characters, but rather sort of a cheat-sheet of the most commonly used regular expressions, flags, and status-codes. No need to memorize any of this — it's here as a quick-guide for easy copy, paste, & go. There's really not too many of them, and they're easily picked up as you work with .htaccess. So without further ado...

Character/Flag	Definition
#	Instructs the server to ignore the line. Used for including comments.
[F]	Forbidden: instructs the server to return a 403 Forbidden to the client.
[L]	Last rule: instructs the server to stop rewriting after the preceding directive is processed.
[N]	Next: instructs Apache to rerun the rewrite rule until all rewriting is complete.
[G]	Gone: instructs the server to deliver Gone (no longer exists) status message.
[P]	Proxy: instructs server to handle requests by mod_proxy.
[C]	Chain: instructs server to chain the current rule with the previous rule.
[R]	Redirect: instructs Apache to issue a redirect, causing the browser to request the new URL.
[NC]	No Case: defines any associated argument as case-insensitive.
[PT]	Pass Through: instructs mod_rewrite to pass the rewritten URL for further processing.
[OR]	Or: specifies a logical "or" that ties two expressions together such that either one proving true will cause the associated rule to be applied.
[NE]	No Escape: instructs the server to parse output without escaping characters.
[NS]	No Subrequest: instructs the server to skip the directive if internal sub-request.

Character/Flag	Definition
[QSA]	Append Query String: directs server to add the query string to the end of the expression.
[S=x]	Skip: instructs the server to skip the next "x" number of rules if a match is detected.
[E=var:value]	Environmental Variable: instructs the server to set the variable "var" to "value".
[T=MIME-type]	Mime Type: declares the mime type of the target resource.
[xyz]	Character class: any character within square brackets will be a match. For example, "[xyz]" will match any of the characters x, y, or z.
[xyz]+	Character class in which any combination of items within the brackets will be a match. For example, "[xyz]+" will match any number of x's, y's, z's, or any combination thereof.
[^xyz]	Not within a character class. For example, [^xyz] will match any character that isn't x, y, or z.
[a-z]	A dash "-" between two characters within a character class denotes the range of characters between them. For example, [a-zA-Z] matches all lowercase and uppercase letters.
a{n}	Exact number, n, of the preceding character, a. For example, x{3} matches exactly three x's.
a{n,}	Specifies n or more of the preceding character. For example, x{3,} matches three or more x's.
a{n,m}	Specifies a range of numbers, between n and m, of the preceding character, a. For example, x{3,7} matches three, four, five, six, or seven x's.
()	Used to group characters together, thereby considering them as a single unit. For example, (htaccess)?book will match "book", with or without the "htaccess" prefix.
^	Denotes the beginning of a regular expression. For example, "^Hello" will match any string that begins with "Hello". Without the caret "^", "Hello" would match anywhere in the string.
$	Denotes the end of a regular expression. For example, "world$" will match any string that ends with "world". Without the dollar sign "$", "world" would match anywhere in the string.

Character/Flag	Definition
`?`	Declares as optional the preceding character. For example, "`monzas?`" will match "`monza`" or "monzas". In other words, "`x?`" matches zero or one of "x".
`!`	Declares negation. For example, "`!string`" matches everything except "`string`".
`.`	A literal dot (or period) indicates any single arbitrary character.
`-`	Instructs Apache to NOT rewrite the URL. Example syntax: "`example.com - [F]`"
`+`	Matches one or more of the preceding character. For example, "`G+`" matches one or more `G`'s, while "+" will match one or more characters of any kind.
`*`	Matches zero or more of the preceding character. For example, use "`.*`" as a wildcard.
`|`	Declares a logical "or" operator. For example, "`(x|y)`" matches "x" or "y".
`\`	Escape special characters such as: `^ $ ! . * | ( ) [ ] { }`
`\.`	Indicates a literal dot (escaped).
`/*`	Zero or more slashes.
`.*`	Zero or more arbitrary characters.
`^$`	Defines an empty string.
`^.*$`	The standard pattern for matching everything.
`[^/.]`	Defines one character that is neither a slash nor a dot.
`[^/.]+`	Defines any number of characters that contains neither slash nor dot.
`http://`	This is a literal statement — in this case, the literal character string, "`http://`".
`^example.*`	Matches a string that begins with the term "`example`", followed by any character(s).

Character/Flag	Definition
`^example\.com$`	Defines the exact string, "`example.com`".
`-d`	Tests if string is an existing directory.
`-f`	Tests if string is an existing file.
`-s`	Tests if file in test string has a non-zero value.

Server status-codes

Lastly, here is a short-list of some of the most-commonly used status-codes (e.g., used when redirecting and rewriting URLs):

- **301** – Moved Permanently
- **302** – Moved Temporarily
- **403** – Forbidden
- **404** – Not Found
- **410** – Gone

For a complete list of status-code definitions, visit: http://htaccessbook.com/g

2.8 Other requirements

As you get started with .htaccess, things may not work as expected. For example, if you add the permalink-rules for WordPress, you may discover that either they don't work at all, or worse, that your site is throwing a 500-error. Similar situations may occur for other rules, so if things aren't working as expected, here is a short list of things to check.

Is .htaccess enabled on the server?
Your site may contain a hundred .htaccess files, but Apache will ignore all of them if the `AllowOverride` directive is set to none. This directive determines which directives will be honored if found in .htaccess files. If you have access to Apache's main configuration file, you may enable .htaccess by setting `AllowOverride` to "`All`" or by selectively enabling specific types of directives such as `AuthConfig`, `FileInfo`, `Indexes`, `Limit`, and/or `Options`.

```
<Directory "/usr/local/httpd/htdocs">
      AllowOverride All
</Directory>
```

Is Apache loading the required module?
By default, Apache loads only a core set of modules, and some web hosts modify which modules are loaded for various types of accounts. So if, say, WordPress permalinks aren't working after adding the required directives, you may want to check whether the required Apache module — **mod_rewrite** in this case — is being loaded into the server.

There are several ways to determine if Apache is loading the required module. First, if you're savvy with the command-line, you can list currently compiled modules with the "-l" command. Another way to check is to look at the main Apache configuration file (named "httpd.conf") and see if the required module is commented out or not. For example, here is how **mod_rewrite** looks as included by default in Apache's main configuration file:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

And here is what it will look like if it is not included (i.e., disabled):

```
# LoadModule rewrite_module modules/mod_rewrite.so
```

As discussed in section 2.6, the hash-symbol "#" is used as the conventional way of disabling a directive, but it's also okay to remove the directive from the configuration altogether, so heads up if you don't see the desired module.

IfModule directives

To help prevent errors caused by missing Apache modules, the techniques presented in this book are enclosed within "IfModule" directives. IfModules are conditional directives that "wrap" whatever module-specific rules you're using. They look like this:

```
<IfModule mod_rewrite.c>
      # mod_rewrite rules go here..
</IfModule>
```

This example would check to see if the **mod_rewrite** module is loaded before executing the enclosed directives. This prevents server-crashes when modules are not available. Wherever applicable, we'll use `<IfModule>` directives as a "best-practice" for techniques in this book.

Excellent guide to configuring Apache with the httpd.conf file: http://htaccessbook.com/h

See the blue "httpd.conf" sidebar in sections 3.2 and 4.5 for more information about this technique.

Getting into the command-line? Check out the "Linux Server SSH Cheat Sheet" http://htaccessbook.com/i

Another good resource for "Useful Apache Commands": http://htaccessbook.com/j

For more about the IfModule directive, check out its section in the Apache Docs: http://htaccessbook.com/k and also: http://htaccessbook.com/l

In some sections, `<IfModule>` directives are omitted to save space. You'll get further reminders :)

Note that you can also use "not-if" containers as a conditional check in your .htaccess files. We see an example of this on page 45. Here is an example of it's syntax, checking for the absence of mod_filter:

```
<IfModule !mod_filter.c></IfModule>
```

2.9 Testing locally vs. testing live

When designing websites, many designers I know work directly on a live server, either on a private test-domain (recommended) or even on live sites (not recommended). If you must go "guerilla-style" on a live site, use extreme caution, especially when working with .htaccess directives.

Implementing changes on a live site may be fine for stuff like CSS and HTML, but when it comes to .htaccess, testing live is not recommended. As discussed in section 2.6, .htaccess syntax is strict — a misplaced comma will trigger the dreaded 500 "Internal Server Error."

There are several alternatives to working on a live site. Least optimal is setting up a temporary maintenance-page*. The next best solution is to test on a private separate domain. And the safest approach is to test locally on your computer*. That way the development process is kept offline until everything is well-tested and ready.

To test .htaccess on your computer, you need to run Apache. If you want to test .htaccess along with your dynamic website (e.g., WordPress), you'll also need to run PHP*, MySQL*, and optimally a gaggle of other minor programs as well (e.g., APC, XCache, phpMyAdmin).

Fortunately there are some great software-bundles that automate much of the process, making it relatively quick and painless to get set up. Such software is usually named with an abbreviation of their included programs, generally "something-AMP", where "AMP" stands for Apache, MySQL, and PHP. Here are some of the most popular of these so-called "_AMP"-based server-software packages:

LAMP http://www.lamphowto.com/	For Linux, installing Apache, MySQL, and PHP is more like running specific commands to install what's known as "LAMP", with the "P" referring to Perl, PHP, or Python, depending on which components are installed.
MAMP http://www.mamp.info/	Mac OS X + Apache + MySQL + PHP. Truly easy to use, MAMP also comes in "PRO" flavor, a professional-grade version that does awesome stuff like multiple servers, external access, easier configuration and more.
XAMPP http://www.apachefriends.org/en/xampp.html	Actually four separate distributions, there's a XAMPP for Linux, Mac, Solaris, and Windows. The "X" in "XAMPP" refers to "cross-platform", and the extra "P" is for Perl, which makes this particular _AMP even more capable. Portable/USB versions also available.
WAMP http://www.wampserver.com/	Renamed WampServer, WAMP is exactly what you would expect: Apache, MySQL, and PHP for Windows. If you're a Windows user, you'll find WAMP a solid, flexible program that's pretty easy to use. Also available in portable/USB flavors.

In addition to these top suites, there's also JAMP, ZWAMP, and many others that include all sorts of programs and some that actually aren't named with an abbreviation*.

See section 6.3 for a modular site-maintenance method that's simple to enable/disable.

Apache/PHP/MySQL on Mac: http://htaccessbook.com/7b
Apache/PHP/MySQL on PC: http://htaccessbook.com/7c

PHP is a widely-used general-purpose scripting language
http://www.php.net/

MySQL: The world's most popular open source database
http://www.mysql.com/

Two decent Wikipedia pages on "_AMP"-based software:

- List of _AMP packages: http://htaccessbook.com/m
- Comparison of _AMPs: http://htaccessbook.com/n

Downloading the Apache HTTP Server
http://htaccessbook.com/7j

2.10 Chapter Summary

To help stay focused, and for quick-reference, here is a summary of key points presented in this important chapter covering the basics of .htaccess.

2.1 Required skills
If you are a web-designer, web-administrator, or work on the Web in any capacity, you should have all the skills needed to use .htaccess. An eye for detail is also good.

2.2 Required software
Working with .htaccess requires Apache, a plain-text editor, and a web-browser.

2.3 Conventions used in this book
Everything should be self-explanatory, with special icons in the footer-area* for notes, links, and other information. Also, code-examples are indicated along the edge of the page for easier reference.

2.4 About the .htaccess file
.htaccess files are used to configure your site at the directory-level. Syntax is extremely rigid. Small mistakes will trigger an error. Remember to make backups.

2.5 How .htaccess files work
.htaccess rules are executed in "cascading" fashion down the directory structure.

2.6 Basic structure and syntax
.htaccess directives are generally case-insensitive*, with each directive on its own line.

2.7 Character Definitions
In section 2.7, you'll find definitions of commonly used characters, status codes, flags, and more. Many of these characters are used in examples throughout the book.

2.8 Other requirements
For .htaccess to work, it must be enabled in the main configuration file and the required modules must be loaded.

2.9 Testing locally vs. testing live
Testing "guerilla style" on a live server is not recommended. Use a private test-domain or a local installation of "_AMP" instead.

That's Chapter 2 in a nutshell, so if anything on the list doesn't look familiar, you may want to take a moment to review before moving into the "meat" of the book. So, now that we've covered the basics, let's jump into some essential .htaccess techniques.

Yep, it's the now-infamous "footer-area" :)

.htaccess tips
http://htaccessbook.com/7t

Case-sensitivity (or lack thereof) depends on the directives and expressions being used. The point is to be aware of proper syntax.

more .htaccess tips
http://htaccessbook.com/7u

Apache Core Features
http://htaccessbook.com/7p

Apache Docs: Index of Modules
http://htaccessbook.com/7o

Behind the Scenes with Apache's .htaccess
http://htaccessbook.com/7l

chapter 3

essential techniques

.htaccess techniques may vary greatly from site to site, but there are a handful that are useful for virtually any website. From enabling functionality to logging server activity, these essential techniques culminate in a "universal" .htaccess starter template.

When beginning a new website, you can streamline production by utilizing a predefined set of "template" files — or "boilerplate" files — that are common to most any site on the Web. Such files include stuff like the robots.txt file, favicons, JavaScript libraries, CSS templates, and so on. The same principle may be applied when configuring a site with .htaccess: some directives are super-useful for most any setup. In this chapter, we'll cover these essential techniques and then combine them into a starter-template designed to kick-start development and speed-up production.

Such as the most-awesome HTML5 Boilerplate: http://html5boilerplate.com/

There's even a jQuery Boilerplate: http://jqueryboilerplate.com/

Boilerplate WordPress Theme: http://htaccessbook.com/o

And of course the .htaccess "boilerplate", aka the "starter" file: http://htaccessbook.com/members/

3.1 Enable mod_rewrite

As discussed in Chapter 2, certain servers may not have **mod_rewrite** enabled by default. The rewrite module is required for rewriting (redirecting) URLs from one page to another. To ensure that **mod_rewrite** is enabled on your server, install the temporary maintenance page and visit your site in a web-browser. The maintenance page requires **mod_rewrite** to work, so if you see the "We'll be right back..." message, that means you're good to go for URL-rewriting. If it's not working, then add the following line to the root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteEngine On
</IfModule>
```

To test that it's working, revisit the maintenance page.

3.2 Enable symbolic links

You'll notice in the .htaccess starter-template that there are several listed values for the "`Options`" directives, located near the top of the file. These options are used to configure certain features, such as CGI, SSI, and symbolic links, or "symlinks" if you're nasty.

Symbolic links are used to integrate external directories into the filesystem. By default, files and directories not strictly beneath the "`DocumentRoot`" (i.e., the web-accessible root-

Apache docs for mod_rewrite: http://htaccessbook.com/p

If things aren't working, see section 10.2 and 10.3 for troubleshooting and how to log mod_rewrite activity.

Learn more about the site-maintenance technique in section 6.3.

Apache docs on symbolic links: http://htaccessbook.com/r

httpd.conf
AllowOverride & FollowSymLinks

As discussed, for symbolic links to work, Apache must be given explicit permission. The .htaccess method makes use of the Options directive, which itself must be enabled from within the httpd.conf file. Example:

```
<Directory "/var/www/html">
        AllowOverride Options
</Directory>
```

For performance considerations, it is important to only enable `AllowOverride` in the specific directory in which it is required.

While working with the httpd.conf file, we may go ahead and enable symbolic links from that location, rather than via .htaccess. Just add this to your httpd.conf file:

```
<Directory "/var/www/html">
        Options FollowSymLinks
</Directory>
```

directory) is not a part of the Apache filesystem, and thus not configurable via .htaccess.

Apache provides several ways of bringing other parts of the filesystem under the `DocumentRoot`, including "`Alias`", "`ScriptAlias`", and "`ScriptAliasMatch`" directives, as well as via shell-induced symbolic links.

Regardless of which method is used, Apache will follow symbolic links only when given explicit permission, either in httpd.conf or .htaccess. See the blue "httpd.conf" sidebar for more information on using either of these techniques.

To enable symlinks via .htaccess, add the following directive to the target directory:

```
# enable symbolic links
Options +FollowSymLinks
```

If you know that your site won't be using any symbolic links, or if you've enabled them via the main configuration file, feel free to comment-out or remove this directive. In general it's good practice to disable any functionality that's not needed, but technically it's fine to repeat the `FollowSymLinks` directive. See the blue sidebar for more about httpd.conf.

3.3 Disable index views

By default, Apache will display the contents of any directory that doesn't include some sort of index file. For some directories, this may be a useful for public content, but most of the time it's undesirable. For example, directories that contain sensitive core files, such as WordPress' `/wp-admin/` and `/wp-includes/` — there's no reason to list the contents of these directories for the public.

To disable directory listings for all directories, add this line to the site's root .htaccess file:

```
# disable directory listing
Options All -Indexes
```

To test, visit any directory that doesn't contain an index file. If you need to enable directory listing for some specific directory, create an .htaccess file for it, and add the following lines:

Index of /Pink-Floyd/More

Name	Last modified	Size	Description
Parent Directory		-	
01 - cirrus minor.mp3	15-Jun-2012 20:40	12M	
02 - the nile song.mp3	15-Jun-2012 20:40	7.8M	
03 - crying song.mp3	15-Jun-2012 20:41	8.2M	
04 - up the khyber.mp3	15-Jun-2012 20:41	5.1M	
05 - green is the colour.mp3	15-Jun-2012 20:41	6.8M	
06 - cymbaline.mp3	15-Jun-2012 20:41	11M	
07 - party sequence.mp3	15-Jun-2012 20:41	2.7M	
08 - main theme.mp3	15-Jun-2012 20:41	13M	
09 - ibiza bar.mp3	15-Jun-2012 20:41	7.5M	
10 - more blues.mp3	15-Jun-2012 20:41	5.1M	
11 - quicksilver.mp3	15-Jun-2012 20:42	16M	
12 - a spanish piece.mp3	15-Jun-2012 20:42	2.5M	
13 - dramatic theme.mp3	15-Jun-2012 20:42	5.2M	
Seabirds.mp3	15-Jun-2012 20:42	731K	

By default, Apache displays the contents of directories that don't include an index file.

Error 403 — Forbidden

You don't have permission to access `Requested URL: http://bluefeed.net/Pink-Floyd/More/` on this server..

Apache returns a "403 — Forbidden" response when directory listing is disabled. This prevents scripts, bots, and humans from any meddling.

`The SymLinksIfOwnerMatch` directive may be used in place of FollowSymLinks — either works to enable symbolic links. Read more in the Apache Docs: http://htaccessbook.com/r

Quick tutorial explaining two ways to create symbolic links: http://htaccessbook.com/s

Straight-up post on "How to Create a Symlink": http://htaccessbook.com/t

There are several other "httpd.conf" sidebars throughout the book. Refer to the Table of Contents for more info.

WordPress protects these files with a blank index.php file, which is a good way to selectively disable directory-views.

Unless disabled by the web-host. Some hosts disable index-views as an added security measure.

```
# DISABLE DIRECTORY LISTING
Options All +Indexes
```

As with other `Options` directives, `AllowOverride` must be enabled in the httpd.conf file.

3.4 Specify the default language

Apache makes it easy to control the default language used by different directories. For example, if you provide translated versions of your web pages, each in their own directory, you can set the default language for each with Apache's `DefaultLanguage` directive.

This can be a huge time-saver when working with multilingual sites — no more messing with meta tags to set the language. To specify the default language for the entire site, place the following directive near the top of the root .htaccess file:

```
DefaultLanguage en
```

Here, we're specifying English as the default language using its two-digit abbreviation. This will cascade down the filesystem and apply to all directories and files therein.

As mentioned, the default language may be overridden in specific subdirectories. If we have a subdirectory for a French translation, for example, we would create an .htaccess file and add the following line:

```
DefaultLanguage fr
```

See the footer for more information and resources about setting the default language.

3.5 Specify the default character set

It's also possible to specify the default character-set (charset) for all of your HTML and plain-text content. Apache's `AddDefaultCharset` directive may be used to add a default charset parameter to the server-response header. Basically, that just means the server lets the browser know how the content is encoded.

The screenshot at right shows an example of this. First, the request to Google.com is made by the web browser. The server then responds with `UTF-8` as the charset parameter for the `Content-Type` header.

By default, Apache disables `AddDefaultCharset`. If you enable it using the "`On`" value, the default charset is ISO-8859-1. Otherwise, to specify your own default charset, such as UTF-8, add the following directive to the root .htaccess file, preferably somewhere near the top of the file:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:12.0) Ge
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: NID=60=CZGVl8Zz7mwHFTSKpQakTKy8MvejZIllvXM_ekt-E-
Cache-Control: max-age=0

HTTP/1.1 200 OK
Date: Mon, 18 Jun 2012 22:49:05 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 25341
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
```


See the blue httpd.conf sidebar on page 26 for basic information about enabling `AllowOverride`.

Note that multiple language variations may be specified like so: `DefaultLanguage el, em, en`

`DefaultLanguage` applies to all files in the directive's scope, which excludes files that have an explicit language extension, such as "`.ch`" or "`.de`": http://htaccessbook.com/u

Quick example showing how to override language-defaults for specific file-types: `AddLanguage en .html .css .js`

The screenshot above shows the Google.com returning the UTF-8 charset along with the `Content-Type` header.

ISO 8859-1 character set overview: http://htaccessbook.com/7d

The scoop about UTF-8: http://www.utf-8.com/

The charset should be an IANA registered charset value for use in MIME media-types. See http://htaccessbook.com/v

```
AddDefaultCharset utf-8
```

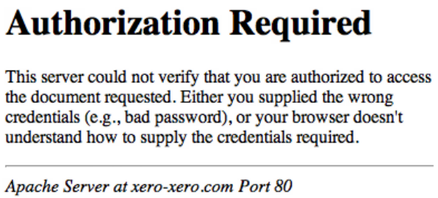
Specifying the default charset via server-response header should override any charset specified in the body of the response, such as those included via meta tag. If your site is in fact specifying default character set via meta tags, and you don't need the coverage for any plain-text files, it's better to disable `AddDefaultCharset` and roll with the meta tags.

```
AddDefaultCharset Off
```

3.6 Disable the server signature

Unless you have reason to do otherwise, disabling your server signature is a good way to keep sensitive information out of the wrong hands. Why broadcast sensitive server details such as which port you're using, your server name, and possibly other information?

Fortunately this behavior is disabled by default, but some hosts enable it for certain configurations. If you're sure you don't need to display that information, it should be disabled as a basic security measure. As seen in the "Authorization Required" screenshot, server-generated documents include the default server-signature displayed in the footer area.



For more info on setting the default charset with .htaccess, check out: http://htaccessbook.com/w

How To Use HTML Meta Tags http://htaccessbook.com/7e

How to disable (apache's) server signature http://htaccessbook.com/x

How (and why) to disable apache server signature http://htaccessbook.com/y

It's possible to customize the footer-line using the `ServerTokens` directive or by specifying "EMail" as the value for the `ServerSignature` directive.

Unless you have reason to do otherwise, it's best to disable this feature, preferably via the main configuration file, but it's also possible using the following line in the root .htaccess:

```
ServerSignature Off
```

Along with the other essential techniques in this chapter, this directive is included in the .htaccess starter-template included with this book. We'll get to that after seeing two more widely used .htaccess techniques.

3.7 Configure ETags

According to the Yahoo! Developer team, disabling ETags can improve site performance by decreasing response-sizes by around 12 Kilobytes each. There's a long, sordid story that goes with the "what", "why" and "how" of ETags, but let's not get into that here. Rather, let's stay focused on the task at hand: best practices and essential .htaccess techniques. And when it comes to ETags, it all depends on how your website is hosted. If you're hosting your site on a single server, Apache's default configuration should work fine. If, on the other hand, your site is hosted on a network of servers, ETags are probably *decreasing* performance and should be disabled.

The `ServerSignature` directive allows the configuration of a trailing footer-line under server-generated documents such as error-messages, directory-listings, and module output. More information in the Apache Docs: http://htaccessbook.com/z

Yahoo's "Best Practices for Speeding Up Your Web Site": http://htaccessbook.com/10

Steve Souders' "High Performance Web Sites": http://htaccessbook.com/86

Fortunately, Apache makes it easy to override default settings using the **FileETag** directive. For example, to **disable ETags**, add the following line to the root .htaccess:

```
FileETag none
```

Whenever possible, I like to keep core directives — such as `DefaultLanguage`, `ServerSignature`, and `FileETag` — located at the beginning of the .htaccess file. It makes sense mostly from a functional point of view, but really they may be placed anywhere.

3.8 Enable basic spell-checking

Apache has a built-in spelling-check module — ironically named "**mod_speling**" — that can "fix" basic spelling and capitalization errors in the URL request. The Apache documentation really explains it best:

> *[mod_speling] does its work by comparing each document name in the requested directory against the requested document name without regard to case, and allowing up to one misspelling (character insertion / omission / transposition or wrong character).*

For example, let's say a visitor misspells the URL to your site's "About" page, located at `http://example.com/about/`. With `CheckSpelling` enabled, one of the following things will happen:

- Apache can't find a matching document, and so delivers a "document not found" error.
- Apache finds something that "almost" matches the URL request, and redirects to it.
- Apache finds multiple possible matches and presents a list of options to the client.

`CheckSpelling` is disabled by default, but the general consensus is that it's useful for SEO. The following directive is also included in the starter-template, and may be added to the root .htaccess file to enable basic spell-checking on your site:

```
<IfModule mod_speling.c>
      CheckSpelling On
</IfModule>
```

"Mission accomplished," as it were. Further details are available in the Apache Docs.

3.9 Combining Options

Two of the techniques in this chapter — `Indexes` and `FollowSymlinks` — are enabled via the `Options` directive. It's perfectly fine to write them separately, like so:

```
Options -Indexes
Options +FollowSymlinks
```


Some servers require more "convincing" to disable ETags, so if FileETag isn't cutting it, try adding this snippet:

```
<IfModule mod_headers.c>
      Header unset ETag
</IfModule>
```

Read more about FileETag in the Apache Docs: http://htaccessbook.com/11

For more details about how mod_speling works, visit the Apache Documentation: http://htaccessbook.com/12

SEO = Search Engine Optimization http://htaccessbook.com/2i

The Options directive is part of the Apache core: http://htaccessbook.com/14

If you don't want the server trying to "guess" at which URL to serve, but would like to ensure that all URLs are returned in lowercase format, you may want to try the `CheckCaseOnly` directive:

http://htaccessbook.com/13

By default, the `Options` directive is set to "`All`", which is overridden by any `Options` values that are more specific. Similarly, when *multiple options* are specified, only the most specific is applied, again by default. The key to specifying multiple values is the plus-sign "+" or minus-sign "-", which instruct Apache to merge the options or remove them from the `Options` currently in place. This enables us to combine options into a single directive:

```
Options -Indexes +FollowSymlinks
```

In this fashion, as many options as needed may be combined. See the Apache Docs for further information and examples.

3.10 .htaccess starter-template

Any or all of the techniques in this chapter may be applied to your site, but as discussed in the chapter-introduction, they are all general and useful enough to be included in just about any .htaccess file, collectively as a foundation.

To help streamline development, I've combined these essential techniques into an .htaccess "boilerplate", or "starter" template. It's a simple and flexible template to customize and build upon. Anything that's not needed may be removed or commented-out with a hash-symbol "#". I use a similar file, tuned to my particular server setup, for new projects or implementing .htaccess on client sites.

The Options directive is part of the Apache core, as described in the official documentation: http://htaccessbook.com/14

The .htaccess starter-template contains only directives and techniques that are covered explicitly in this book. Log in to the Member's Area for current download: http://htaccessbook.com/members/

Installation

To install the .htaccess starter-template, you'll need a way to view hidden files. There are several ways to do this, either by configuring your operating system, installing an application, or simply viewing the file in a FTP or text-editing program that displays hidden files.

Once you can "see" .htaccess files on your system, follow these steps to install the starter-template:

1. Download the zipped starter-template
2. Unzip the template directory that contains the file
3. Copy to your site's root directory (e.g., `public_html`)
4. View/edit the .htaccess file as needed

Once you get the template-file uploaded to the server, remember to test for proper functionality for the different parts of your site. Then once everything is in place and working, it's time to optimize your site by tuning the .htaccess file to meet your specific requirements. And with that, let's jump into some more awesome .htaccess techniques.

httpd.conf
Rename the .htaccess file

Renaming the .htaccess file obfuscates its identity, which adds an extra layer of security to your website(s). Further, beginning the file-name with something other than a dot will make the files easier to work with on your local machine.

To rename the .htaccess file, add the following code to the httpd.conf file:

```
# rename htaccess files
AccessFileName ht.access
```

By default, Apache protects the .htaccess file from external access, but it may not protect renamed .htaccess files (such as "ht.access") by default. So just in case, you can explicitly restrict access using the following code:

```
<FilesMatch "^ht\.">
        Order deny,allow
        Deny from all
</FilesMatch>
```

Two ways to display hidden files on a Mac:
With Terminal: http://htaccessbook.com/15
With Houdini: http://htaccessbook.com/16

Most FTP programs display hidden files by default, but you may have to enable such in the options or settings.

Great guide for displaying hidden files on Windows: http://htaccessbook.com/17

To download the starter-template, visit the book's Members Area: http://htaccessbook.com/members/

chapter 4

optimizing performance

Apache enables some great techniques for improving the performance of your website. From conserving vital system resources to compressing and caching content, .htaccess files provide fine-grained control over many key aspects of your site.

Beyond providing excellent content or getting good links, improving the performance of your site is the best way to boost your site's visibility and success. Faster performing sites translate into better placement in the search-engine results, more traffic, and a better user-experience.

There are many ways to go about optimizing your site, including some awesome techniques implemented via .htaccess. In this chapter, you'll learn how to enable file-compression, optimize cache-control, and much more.

Performance optimization is about much more than .htaccess: "Best Practices for Speeding Up Your Web Site": http://htaccessbook.com/10

Maaaaannnnny optimization resources: http://www.websiteoptimization.com/

Essential reading: "WPO – Web Performance Optimization": http://htaccessbook.com/19

Apache Performance Tuning http://htaccessbook.com/7q

4.1 Essential techniques

In Chapter 3, four of the "essential" techniques are recommended for improving the performance of your site. I want to mention these methods in the performance chapter to keep things organized. If you're using the .htaccess starter-template, then these directives should be already included. If not, here they are once more:

```
ServerSignature Off
AddDefaultCharset UTF-8
DefaultLanguage en
FileETag none
```

Refer to Chapter 3 for more information on these four directives.

4.2 Enabling file compression

Compressing the size of your web-pages before sending them to the client results in faster page loading and a better experience for your visitors. You can do this with a scripting language like PHP, but it's more efficient to let Apache do the work using **mod_deflate**.

Although Apache includes the deflate module, some web-hosts disable it on their servers. Enabling it is easy if you have access to the main configuration file, otherwise you'll need to ask your web host to enable it for you. Once enabled, **mod_deflate** may be applied

See section 3.10 for more information about the .htaccess starter-template.

Apache Module mod_deflate http://htaccessbook.com/1a

To enable mod_deflate, find this line in httpd.conf: `# LoadModule deflate_module modules/mod_deflate.so` then uncomment the directive by removing the "#".

Note: Apache's main config file is named "httpd.conf". Grab a copy @ http://htaccessbook.com/members/

to any directory or to an entire site, enabling targeted compression of specific file-types, directories, or everything. As seen in the screenshot here, compressing your web pages with **mod_deflate** can reduce size by as much as 70% or more. That means faster page-loading and better experience for your visitors. From the many techniques available, here are some of the best ways to implement HTTP compression for your website, or any specific part of it.

Perishable Press is compressed via `mod_deflate`*, as verified @* `http://htaccessbook.com/1e`

As seen here, compressing your pages can reduce page size by more than 70%!

Basic configuration

This first method of configuring **mod_deflate** is well-known and used at web-hosts such as Media Temple and Rackspace. As always, it's recommended to place these directives in the main configuration file, but they also work great from the root .htaccess file.

```
<IfModule mod_deflate.c>
      <IfModule mod_setenvif.c>
            AddOutputFilterByType DEFLATE text/css text/html text/plain text/xml
            AddOutputFilterByType DEFLATE text/javascript application/javascript
            BrowserMatch ^Mozilla/4 gzip-only-text/html
            BrowserMatch ^Mozilla/4\.0[678] no-gzip
            BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
      </IfModule>
</IfModule>
```


As with any server process, mod_deflate does require a bit of CPU to do the compression, but overall CPU-load decreases because the server is processing much less data.

Rackspace Hosting
http://www.rackspace.com/

Media Temple (mt)
http://mediatemple.net/

It's specific to Media Temple but contains some good practical insight: http://htaccessbook.com/1b

When placed in the root .htaccess file, these directives tell Apache to compress all text, HTML, CSS, and JavaScript for your entire site. To limit compression to a subdirectory, create an .htaccess file and add the code there instead of in the root directory. The three `BrowserMatch` directives are there to help older browsers with the compressed content.

Configure compression with mod_filter

If available on your server, Apache's filter module provides more control over the configuration of **mod_deflate**. By using a "filter harness", **mod_filter** conditionally targets different types of content "based on any Request Header, Response Header or Environment Variable." This "smart" filtering gives you more control over configuration than either `AddOutputFilter` or `AddOutputFilterByType`, as used in the previous method.

```
<IfModule mod_deflate.c>
      <IfModule mod_filter.c>
            FilterDeclare  COMPRESS
            FilterProvider COMPRESS DEFLATE resp=Content-Type $text/css
            FilterProvider COMPRESS DEFLATE resp=Content-Type $text/html
            FilterProvider COMPRESS DEFLATE resp=Content-Type $text/javascript
            FilterProvider COMPRESS DEFLATE resp=Content-Type $text/plain
            FilterProvider COMPRESS DEFLATE resp=Content-Type $text/xml
            FilterProvider COMPRESS DEFLATE resp=Content-Type $application/javascript
            FilterChain    COMPRESS
            FilterProtocol COMPRESS DEFLATE change=yes;byteranges=no
      </IfModule>
</IfModule>
```

Apache version **2.1** and later

Apache Module mod_filter
http://htaccessbook.com/1c

Go deep into mod_filter, check out "An Architecture for Smart Filtering in Apache": http://htaccessbook.com/1d

In the previous code, **mod_filter** instructs **mod_deflate** to compress the same file types as the first method: text, HTML, CSS, and JavaScript. When placed in the root .htaccess file, these directives apply to the entire site. To compress only certain directories, place the code in the corresponding .htaccess file, or implement via `<Location>` or `<Directory>` directives in the main configuration file. To verify that compression is working, visit an online compression tool such as the one mentioned previously.

Compression tips and tricks

Either of the previous two methods should work great at compressing your web-pages, but they are fairly general, and more precise configuration may be required depending on the project. Without devoting the remainder of the book to this topic, there are some useful techniques that are worth mentioning here. See the footer for additional resources.

Simple way to compress only the basics

This technique is a simple way to compress only basic file-types without any additional rules for archaic browsers. This method is nice because it's just a single directive that may be customized with additional file-types. Just add the following lines to your .htaccess file:

```
<IfModule mod_deflate.c>
      AddOutputFilterByType DEFLATE text/css text/html text/plain text/javascript
</IfModule>
```


Online HTTP Compression Test
http://htaccessbook.com/1e

Solid write-up on HTTP Compression:
http://www.http-compression.com/

How To Optimize Your Site With GZIP Compression
http://htaccessbook.com/1f

GZIP compression? There's a module for that. But it's not included with Apache. http://htaccessbook.com/1g

Compress everything except images

Here is a set of directives that will compress everything except images:

```
<IfModule mod_deflate.c>
      <IfModule mod_setenvif.c>
            SetOutputFilter DEFLATE
            BrowserMatch ^Mozilla/4 gzip-only-text/html
            BrowserMatch ^Mozilla/4\.0[678] no-gzip
            BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
            SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip dont-vary
            Header append Vary User-Agent env=!dont-vary
      </IfModule>
</IfModule>
```

Note that if you're using Apache 2.0.48 or less, you should replace the "`BrowserMatch \bMSIE`" line with the following:

```
BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html
```

Help proxies deliver correct content

Apache has it's own mechanisms in place for dealing with proxy servers, but you can help ensure that proxies deliver the correct content (compressed or uncompressed) by adding the following rules along with your configuration directives for **mod_deflate**. Simply add the following rules to the same .htaccess file (just beneath the **mod_deflate** rules):

More information here:
http://htaccessbook.com/1a

All about proxy servers at Wikipedia:
http://htaccessbook.com/1h

```
<IfModule mod_deflate.c>
	<IfModule mod_headers.c>
		Header append Vary User-Agent env=!dont-vary
		Header append Vary Accept-Encoding
	</IfModule>
</IfModule>
```

Force compression of mangled headers

As discussed by the Yahoo! Developer team, "roughly 15% of visitors are not receiving compressed responses even though these user agents support compression."[•] The article then goes on to explain several potential solutions, which culminates in this technique[•]:

```
<IfModule mod_deflate.c>
	<IfModule mod_setenvif.c>
		<IfModule mod_headers.c>
			SetEnvIfNoCase ^(Accept-EncodXng|X-cept-Encoding|X{15}|~{15}|-{15})$ \
				^((gzip|deflate)\s*,?\s*)+|[X~-]{4,13}$ HAVE_Accept-Encoding
			RequestHeader append Accept-Encoding "gzip,deflate" \
				env=HAVE_Accept-Encoding
		</IfModule>
	</IfModule>
</IfModule>
```

This is a potentially useful technique, although I've not seen a lot of test-data or other evidence as to its actual effectiveness. So if you decide to use it, keep an eye on your server logs until you're sure that anything weird isn't happening traffic-wise.

Pushing Beyond Gzipping
http://htaccessbook.com/1i

Also featured in the HTML5 Boilerplate:
http://htaccessbook.com/1j

Just a reminder to always make a backup before making changes to .htaccess, and test thoroughly afterwards.

Compress additional file types

Not every type of file should be compressed, but there are good reasons why you would want to compress more than text, CSS, HTML, and JavaScript, as we've done so far. Here is a more-complete list[•] of files commonly used on the Web. Just grab what you need[•].

```
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/css
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/html
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/javascript
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/plain
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/x-component
FilterProvider COMPRESS DEFLATE resp=Content-Type $text/xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/javascript
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/json
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/x-javascript
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/atom+xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/rss+xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/xhtml+xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/vnd.ms-fontobject
FilterProvider COMPRESS DEFLATE resp=Content-Type $application/x-font-ttf
FilterProvider COMPRESS DEFLATE resp=Content-Type $font/opentype
FilterProvider COMPRESS DEFLATE resp=Content-Type $image/svg+xml
FilterProvider COMPRESS DEFLATE resp=Content-Type $image/x-icon
```

These directives are formatted for use with our second method, compressing with **mod_filter**[•]. To reformat for use with `AddOutputFilterByType`, remember to remove the "`$`".

(Near-complete) MIME Types List:
http://htaccessbook.com/1k

Apache Module mod_filter
http://htaccessbook.com/1c

For example, if you're working with SVG files and want to compress them, copy the following line from the list:

```
FilterProvider COMPRESS DEFLATE resp=Content-Type $image/svg+xml
```

And include it with your other compression directives.

Putting it all together

Now that we've seen some good techniques for configuring **mod_deflate**, let's combine them into one set of directives to rule them all. Copy and paste the following code into your httpd.conf (preferably) or .htaccess file[*]:

```
<IfModule mod_deflate.c>
	<IfModule mod_setenvif.c>
		<IfModule mod_headers.c>
			SetEnvIfNoCase ^(Accept-EncodXng|X-cept-Encoding|X{15}|~{15}|-{15})$ \
			^((gzip|deflate)\s*,?\s*)+|[X~-]{4,13}$ HAVE_Accept-Encoding
			RequestHeader append Accept-Encoding "gzip,deflate" \
			env=HAVE_Accept-Encoding
		</IfModule>
	</IfModule>
	<IfModule mod_filter.c>
		FilterDeclare  COMPRESS
		FilterProvider COMPRESS DEFLATE resp=Content-Type $text/css
		FilterProvider COMPRESS DEFLATE resp=Content-Type $text/html
		FilterProvider COMPRESS DEFLATE resp=Content-Type $text/javascript
		FilterProvider COMPRESS DEFLATE resp=Content-Type $text/plain
		FilterProvider COMPRESS DEFLATE resp=Content-Type $text/xml
		FilterProvider COMPRESS DEFLATE resp=Content-Type $application/javascript
		FilterChain    COMPRESS
		FilterProtocol COMPRESS DEFLATE change=yes;byteranges=no
	</IfModule>
```


Note: this technique continues on the next page.

.htaccess rules for site speed optimization
http://htaccessbook.com/7s


```
	<IfModule !mod_filter.c>
		AddOutputFilterByType DEFLATE text/css text/html text/plain text/xml
		AddOutputFilterByType DEFLATE text/javascript application/javascript
	</IfModule>
	<IfModule mod_setenvif.c>
		BrowserMatch ^Mozilla/4 gzip-only-text/html
		BrowserMatch ^Mozilla/4\.0[678] no-gzip
		BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
		SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip dont-vary
	</IfModule>
	<IfModule mod_headers.c>
		Header append Vary User-Agent env=!dont-vary
		Header append Vary Accept-Encoding
	</IfModule>
</IfModule>
```

We've already seen how each of the component parts work, so let's look at how those parts work together to compress your web-pages in optimal fashion. Here's the order of events:

1. Checks for all required modules before executing any directives
2. Attempts to force the compression of mangled headers
3. Checks for **mod_filter**, and sets compression for common file-types[*]
4. If no **mod_filter**, compression is set using `AddOutputFilterByType`
5. Helps archaic browsers handle (or not) compressed content
6. Helps proxies deliver the correct content (compressed or not)

Note: this technique begins on the previous page.

Notice in this technique the use of the "not" operator "!", which we're using here as a way of setting directives if mod_filter is not available.

For more info on any of the directives contained within this "all-in-one" technique, refer to their corresponding section in this chapter.

4.3 Optimizing cache-control

Another excellent way to improve site performance involves setting a "far-future expiration-date" for various types of files. Files that specify a healthy expiration date are cached by the browser and used for subsequent visits to your site. For example, the first time a browser loads a web-page, it loads the required files (e.g., `style.css`, `image.png`, `favicon.ico`, et al) into its cache. Then for subsequent visits, the browser checks the Expires Header for each file, and loads directly from cache anything that hasn't expired. Not having to request those assets again from the server decreases the load-time of your web-pages.

As with compressing content with **mod_deflate**, there are numerous ways to optimize cache-control by setting healthy expiration dates for certain types of files. In this section, we'll look at the best way of doing it with .htaccess, and then look at some alternate methods and useful techniques.

There's a lot to this topic, so I encourage exploration beyond the concise overview presented here. Despite the underlying complexity, however, optimizing cache-control with Apache's expires-module is relatively straightforward. Let's take a look...


See section 4.2 for information about mod_deflate.


Speed up your site with Caching and cache-control
http://htaccessbook.com/1l


Server Admin? Much more is possible. Check out mod_cache and the Apache Caching Guide:
http://htaccessbook.com/1m


Great overview/tutorial on caching:
http://htaccessbook.com/1n

Optimize cache-control with mod_expires

The conventional method of customizing cache-control utilizes Apache's expires-module, **mod_expires**. Once enabled in the main configuration file, **mod_expires** may be configured using three directives: `ExpiresActive`, `ExpiresByType`, and `ExpiresDefault`. As with many of the techniques in this book, these directives are best located in httpd.conf, but also work great when included in the site's root .htaccess file. Here's the magic bullet:

```
<IfModule mod_expires.c>
      ExpiresActive on
      ExpiresDefault                          "access plus 1 month"
      ExpiresByType text/html                 "access plus 1 seconds"
      ExpiresByType text/xml                  "access plus 1 seconds"
      ExpiresByType text/plain                "access plus 1 seconds"
      ExpiresByType application/xml           "access plus 1 seconds"
      ExpiresByType application/rss+xml       "access plus 1 seconds"
      ExpiresByType application/json          "access plus 1 seconds"
      ExpiresByType text/css                  "access plus 1 week"
      ExpiresByType text/javascript           "access plus 1 week"
      ExpiresByType application/javascript    "access plus 1 week"
      ExpiresByType application/x-javascript "access plus 1 week"
      <IfModule mod_headers.c>
            Header unset ETag
            Header unset Pragma
            Header unset Last-Modified
            Header append Cache-Control "public, no-transform, must-revalidate"
      </IfModule>
</IfModule>
```


More about using Apache mod_expires to control browser caching: http://htaccessbook.com/1p


See page 50 for more `ExpiresByType` directives.


To customize the cache-duration for different file-types, edit the "`access plus 1 month`" (or similar) with the desired expiration-date. You may specify any number of years, months, weeks, days, hours, minutes, or seconds. And of course much more is possible: http://htaccessbook.com/1o

Once in place, these directives will set Expires Headers for the specified file types. There are five main things happening with this technique:

- Enable the module using the `ExpiresActive` directive
- Set the default cache-duration using the `ExpiresDefault` directive
- Define the cache-duration for specific file-types using the `ExpiresByType` directive
- Remove the Last-Modified, Pragma, and ETag Headers
- Optimize the response with Cache-Control Headers for HTTPS ("`public`") and proxies ("`no-transform`"), and also to force revalidation of stale content ("`must-revalidate`").

Now, the key to optimizing cache-control of your content is to maximize the expiration-date for the various types of files available from your website. The expiration-dates and file-times specified in the above technique apply conventional cache-durations to the most common file-types. As is, it will improve cache-control, but further optimization is possible by "tuning" the `ExpiresByType` directives to the update-frequency of your specific files.

Tuning ExpiresByType directives

For most sites, the cache-durations specified in the above technique are plenty effective. But you can make them even better by optimizing the expires-date for each file type. For example, files that contain your blog posts, feeds, and other frequently changing content should be fetched fresh from the server every time. To ensure the browser does so, we specify a cache-duration of one second for HTML, JSON, and plain-text files.

For design-related files such as CSS and JavaScript that don't change frequently, there are two good ways to go about it. First, you can set a far-future expires-date via .htaccess and then use a "cache-busting" technique to force the browser to download updated files. This looks something like this when viewed in the source-code:

```
<link href="http://htaccessbook.com/css/style.css?ver=20120621">
```

If the query-string appended to the end of the URL has changed since the previous visit, most browsers will assume the file has been changed and will fetch the latest version. This trick enables us to set a far-future expiration-headers for CSS, JavaScript, and other design files, as is recommended for optimal performance.

If cache-busting isn't your style, the trick to tuning cache-durations for CSS, JavaScript, and other design-related files is choose an expiration-date that "finds the balance" between far-future and frequency of potential updates. For most sites, caching such files for a week is about right: visitors will get the updated files within a reasonable amount of time. If the site is brand-new, or undergoing a redesign, you could either shorten the cache-duration, or go with a cache-busting technique. After a design has "matured", the cache-duration for design files may be increased to a month or more — without the need for any cache-busting.
One more example of tuning `ExpiresByType` directives: setting longer cache-durations

Note that these directives are not required if included elsewhere. Also note that the scope of these directives is limited by `<IfModule>` directives.

Boring but essential: Header Field Definitions
http://htaccessbook.com/ir

Also see the cache-disabling .htaccess techniques later in this chapter.

Automatically Version Your CSS and JavaScript Files
http://htaccessbook.com/it

Query-strings work fine, but you may want to version the filename instead: http://htaccessbook.com/is

for images, video, and other media files. Generally, these files don't change once they've been uploaded to the server. Think about it, when was the last time you updated, say, a thumbnail-image for a blog post, or a video about making pasta carbonara the right way[•]? The same for other types of media files such as fonts, favicons, SVG, PDF, and so on — they're generally updated much less frequently, and so may be safely cached for longer periods of time. You could go a month or more, depending on what you've got on the server. This is the whole point of tuning your cache-control directives, to correlate as closely as possible with what's actually happening on the server.

Even so, if time is short or you're not interested in squeezing out every possible drop of site-performance, just rolling with the predefined values will establish some pretty solid cache-control for your site. The degree to which you want to fine-tune (or not) is your call.

Additional file-types for mod_expires

Before getting into some additional caching techniques, here are some additional `ExpiresByType` directives for commonly used file-types (continues on next page):

```
ExpiresByType image/x-ico                "access plus 1 month"
ExpiresByType image/x-icon               "access plus 1 month"
ExpiresByType image/gif                  "access plus 1 month"
ExpiresByType image/png                  "access plus 1 month"
ExpiresByType image/jpe                  "access plus 1 month"
ExpiresByType image/jpg                  "access plus 1 month"
```


Elliott Richmond's "Carbonara the right way": http://htaccessbook.com/1u

Apache Docs: Apache Module mod_expires http://htaccessbook.com/10

Whether to use the Expires header or Cache-Control max-age: http://htaccessbook.com/1v

More info about Cache-control vs. Expires http://htaccessbook.com/1w


```
ExpiresByType image/jpeg                 "access plus 1 month"
ExpiresByType video/ogg                  "access plus 1 month"
ExpiresByType audio/ogg                  "access plus 1 month"
ExpiresByType video/mp4                  "access plus 1 month"
ExpiresByType video/webm                 "access plus 1 month"
ExpiresByType font/truetype              "access plus 1 month"
ExpiresByType font/opentype              "access plus 1 month"
ExpiresByType application/x-font-woff    "access plus 1 month"
ExpiresByType image/svg+xml              "access plus 1 month"
ExpiresByType application/pdf            "access plus 1 month"
ExpiresByType application/vnd.ms-fontobject "access plus 1 month"
```

To include cache-control for any of these file-types, simple copy/paste the entire line into the **mod_expires** ruleset provided previously in this section. Adjust the cache-duration as needed for your specific file setup.

Cache-control for favicons

We've already seen two `ExpiresByType` directives (previous section) aimed at caching favicons[•], "`image/x-ico`" and "`image/x-icon`"; however, even with both of these rules in place, favicons may require further wrangling to cache in every browser.

The reason for this is that some icons have a special MIME-type[•] that must be explicitly targeted for caching. Fortunately, we can use Apache's `AddType` directive to first add the special MIME-type, and then use `ExpiresByType` to define its cache-duration. It's just the

Everything you ever wanted to know about favicons: http://htaccessbook.com/1y

Official information on the "special" MIME-type http://htaccessbook.com/1x

ticket for those hard-to-cache `favicon.ico` files. To implement this technique, add the following directives to your caching rules:

```
<IfModule mod_expires.c>
	<IfModule mod_mime.c>
		AddType image/vnd.microsoft.icon .ico
		ExpiresByType image/vnd.microsoft.icon "access plus 1 month"
	</IfModule>
</IfModule>
```

Alternate method for cache-control

The previous method of setting Expires Headers with the `ExpiresDefault` directive is generally the best way to go about optimizing cache-control, but there is another method that's worth mentioning because of its added flexibility. By using Apache's **mod_headers** to configure HTTP response-headers, it's possible to set custom Cache-Control Headers for just about any specific file, file-type, or location available.

To demonstrate the utility of this alternate method, I've replicated the same basic cache-control rules established with the recommended **mod_expires** technique. Using a combination of Apache's **mod_headers** and **mod_alias**, the following ruleset replicates the max-age directive of the Cache-Control HTTP header for the same file types. So without further ado, here is a good alternate method for cache-control via .htaccess:

Apache Docs: Apache Module mod_headers
http://htaccessbook.com/1z

So if mod_expires isn't available on your server, or if you need more flexibility in terms of matching files and configuring headers, the alternate method on the next page is a good solution.


```
<IfModule mod_headers.c>
	FileETag None
	Header unset ETag
	Header unset Pragma
	Header unset Cache-Control
	Header unset Last-Modified
	# default cache 1 year = 31556926 s
	Header set Cache-Control "max-age=31556926, public, no-transform, must-revalidate"
	<IfModule mod_alias.c>
		<FilesMatch "\.(htm|html|json|rss|txt|xhtml|xml)$">
			# cache markup for 1 second
			Header set Cache-Control "max-age=1, public, no-transform, must-revalidate"
		</FilesMatch>
		<filesMatch "\.(js|css)$">
			# cache for 1 week = 604800 seconds
			Header set Cache-Control "max-age=604800, public, no-transform, must-revalidate"
		</filesMatch>
		<FilesMatch "\.(gif|jpe|jpeg|jpg|png|ico)$">
			# cache image files for 1 month = 2629744 seconds
			Header set Cache-Control "max-age=2629744, public, no-transform, must-revalidate"
		</FilesMatch>
		<FilesMatch "\.(doc|eot|flv|mp4|ogg|pdf|svg|swf|ttf|woff)$">
			# cache fonts and media files for 1 month = 2629744 seconds
			Header set Cache-Control "max-age=2629744, public, no-transform, must-revalidate"
		</FilesMatch>
	</IfModule>
</IfModule>
```

As you can see, this method of setting cache-control headers is more complicated than doing it with **mod_expires**, but with the complexity there is also flexibility. You've got

Is Your Web Site Cache Friendly?
http://htaccessbook.com/20

Hypertext Transfer Protocol – HTTP/1.1
Caching in HTTP: http://htaccessbook.com/21

FilesMatch to target just about file or set of files on the server, and then you have fine-grained control over of the Cache-Control Header, including directives such as "max-age", "public", "no-transform", and "must-revalidate". So it's a great fallback for the recommended method of using **mod_expires**, and enables some useful .htaccess tricks.

Disable caching during site development

Using the alternate method of configuring cache-control, it's possible to disable file-caching for your site. This is useful during development, maintenance, and so forth*. To implement this temporary technique, replace any existing caching-rules with these:

```
# disable file-caching during site maintenance (temporary)
<FilesMatch "\.(css|flv|gif|htm|html|ico|jpe|jpeg|jpg|js|png|pdf|swf|txt)$">
	<IfModule mod_expires.c>
		ExpiresActive Off
	</IfModule>
	<IfModule mod_headers.c>
		FileETag None
		Header unset ETag
		Header unset Pragma
		Header unset Cache-Control
		Header unset Last-Modified
		Header set Pragma "no-cache"
		Header set Cache-Control "max-age=0, no-cache, no-store, must-revalidate"
		Header set Expires "Mon, 10 Apr 1972 00:00:00 GMT"
	</IfModule>
</FilesMatch>
```


During site-development, it's useful combine this technique with the site-maintenance technique in section 6.3.

Disable caching for scripts and other dynamic files

Scripts and other dynamic files are used to generate web-content such as HTML, and should never be cached by the browser. If there is reason to think that the browser is somehow caching dynamic files, throw down this tasty slab in the root .htaccess file:

```
# disable caching for scripts and dynamic files
<FilesMatch "\.(cgi|fcgi|php|pl|scgi|spl)$">
	<IfModule mod_expires.c>
		ExpiresActive Off
	</IfModule>
	<IfModule mod_headers.c>
		FileETag None
		Header unset ETag
		Header unset Pragma
		Header unset Cache-Control
		Header unset Last-Modified
		Header set Pragma "no-cache"
		Header set Cache-Control "private, \
		no-cache, no-store, proxy-revalidate, no-transform"
	</IfModule>
</FilesMatch>
```

Now that we've seen how to customize cache-control via .htaccess, let's move on with another good technique for optimizing the performance of your website.

Expires Http header: the magic number of YSlow
http://htaccessbook.com/22

4.4 Using cookie-free domains

To further improve performance, it's recommended to deliver your site's static components without using cookies, which aren't required for images, videos, and other static files. Ideally, you should be hosting your site's assets on a static subdomain or CDN that's 100% cookie-free. This benefits performance in several ways:

- Less data to process
- Reduces network traffic
- Facilitate proxy caching

.htaccess tastes better with cookies...

If you look at the source-code of big sites like Google and Bing, you'll find their static resources hosted on separate cookie-free domains. Here are a few ways to configure cookie-free hosting for static components:

- Don't use cookies for any request on your domain
- Host your site at `www.example.com` and static components at `static.example.com`
- Use a separate cookie-free domain or subdomain for static components

Google Developers "Web Performance Best Practices": http://htaccessbook.com/23

Yahoo Developers "Best Practices for Speeding Up Your Web Site": http://htaccessbook.com/10

HTTP cookies explained http://htaccessbook.com/2d

The Unofficial Cookie FAQ http://www.cookiecentral.com/faq/

Wherever you decide to host your static files, the key is to keep it 100% cookie-free. You can do this by not setting cookies anywhere on your static domain — keep it all static files only, no scripts, just images, videos, audio files, and so on. That should be all you need to do, but there are cases where further measures are required to eliminate cookies. Fortunately, Apache makes it easy with its headers module. Just include the following code in the root .htaccess file of your static domain (or subdomain):

```
<IfModule mod_headers.c>
      RequestHeader unset Cookie
      Header unset Set-Cookie
</IfModule>
```

This technique does two things to disable cookies on your static domain. First it strips all cookies from the request, and then it also stops the server from sending any cookies back to the client. Note that **mod_headers** must be enabled in httpd.conf for this to work.

4.5 Configuring environmental variables

To round out the chapter, let's look at how to configure environmental variables, which help the server control access, logging, and even communicate with external programs. Using the **env_module**, we can set environmental variables that are "available to Apache HTTP Server modules, and passed on to CGI scripts and SSI pages."

.htaccess Cookies: http://htaccessbook.com/2b
Cookie Stuffing: http://htaccessbook.com/2c
Using Cookies in PHP: http://htaccessbook.com/29
jQuery/PHP cookies: http://htaccessbook.com/28
Cookies in WordPress: http://htaccessbook.com/2a

Using cookies with jQuery: http://htaccessbook.com/27
Cookies with jQuery/JavaScript: http://htaccessbook.com/25
Javascript Cookie Library: http://htaccessbook.com/26

Environment Variables in Apache http://htaccessbook.com/2e

httpd.conf
Optimizing via AllowOverride

To prevent the server from having to scan every directory for .htaccess files, we can limit the scope of the `AllowOverride` directive by disabling it in the root directory and then selectively enabling for specific directories. Here is the basic idea:

```
# disable .htaccess by default
<Directory "/var/www/html">
        AllowOverride None
</Directory>

# enable .htaccess in this directory
<Directory "/var/www/html/wordpress">
        AllowOverride All
</Directory>

# enable select features only
<Directory "/var/www/html/other">
        AllowOverride FileInfo Options
</Directory>
```

Remember to specify the correct path if using this method to improve performance.

You can define environmental variables in the root .htaccess file using this syntax:

```
SetEnv env-variable value
```

So for example, to create an internal variable for a special path, we would do this:

```
SetEnv SPECIAL_PATH /foo/bin
```

Variables set by `SetEnv` are done so later in the request, so if you need to do stuff like rewriting and access-control, use the `SetEnvIf` directive instead. Here is an example where we set a variable whenever an XML file is requested:

```
SetEnvIf Request_URI "\.xml$" xml_request
```

We could then use the "`xml_request`" variable to custom-log all XML-requests:

```
CustomLog logs/xml_log common env=xml_request
```

That's environmental variables in a nutshell, and we'll see more examples elsewhere in the book, as well as more information about creating and customizing log files for various types of server-activity. For now, let's wrap it up with a couple of useful things you can do with the `SetEnv` directive.

Set the timezone

Your website is available around the world, but there are scenarios where it's useful to set your server's timezone to something specific. Here's how to synchronize the server with the same timezone as New York — just add to the root .htaccess file:

```
<IfModule mod_env.c>
      SetEnv TZ America/New_York
</IfModule>
```

Set the email address for the server administrator

Storing the administrator's email address in a variable is a useful way to utilize the `SetEnv` directive. Edit the email address in the following code, and add to the site's root .htaccess:

```
<IfModule mod_env.c>
      SetEnv SERVER_ADMIN default@example.com
</IfModule>
```

And of course much more is possible with the `SetEnv` and related directives, see the Apache documentation for all the gory details. For now, let's move on to the next chapter.

Apache Module mod_env
http://htaccessbook.com/2f

Apache Module mod_setenvif: SetEnvIf Directive
http://htaccessbook.com/2g

For example:
Section 4.2 — enabling file compression
Section 8.2 — serve browser-specific content
Section 10.2 — logging stuff

See section 10.2 for log-customization tricks :)

PHP Manual: List of Supported Timezones
http://htaccessbook.com/2h

chapter 5

improving SEO

"Content is King" as they say, but that doesn't mean you shouldn't strive to improve the Search Engine Optimization (SEO) of your site. And you don't have to be an expert in SEO to use .htaccess to better control the flow of traffic throughout your website.

Every aspect of your site contributes to its success or failure on the Web. Search-engines are continuously improving their ability to measure the quality of your site. They use complex algorithms to rank your site using every factor imaginable, from content and links to performance and security. As you optimize these factors to improve SEO, links and traffic to your site will increase. The key is to focus traffic on your content, maximizing quality signals while minimizing the negative ones. And focusing that traffic is what .htaccess is all about — from preventing duplicate content to cleaning up broken links, this chapter provides some excellent ways to improve the SEO of your site.

The Beginner's Guide to SEO
http://htaccessbook.com/2i

What Is Search Engine Optimization?
http://htaccessbook.com/2j

Small SEO Tools
http://www.smallseotools.com/

Free Online SEO Analyzer
http://htaccessbook.com/2o

5.1 Universal www-canonicalization

An important part of good SEO is minimizing duplicate-content. There are many sources of duplicate content, including variations in your site's URL. If you can access your homepage at "`example.com`" and "`www.example.com`", the search-engines may be dividing your page rank between the two versions of your site. It's better to pick one or the other and then enforce it as the canonical* URL for your site. With Apache, it's easy to remove or require* the "www" prefix for all of your site's URLs.

Remove the www

This technique redirects all requests to "non-www" URLs. It's "universal" code, meaning it's just plug-n-play, with no editing required — it just works. Just add the following chunk of code to any site's root .htaccess file:

```
# remove www
<IfModule mod_rewrite.c>
      RewriteEngine On
      RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
      RewriteRule ^(.*)$ http://%1/$1 [R=301,L]
</IfModule>
```

In the RewriteRule, a 301-response header is sent to indicate that the new URL is

A "canonical" URL is the *definitive* URL for a particular page or resource.

SEO Optimization Checker:
http://htaccessbook.com/2n

DigWP.com Poll: www vs no-www
http://htaccessbook.com/2p

Using Chrome's SEO Site Tools
http://htaccessbook.com/2l

permanent. Remember to make backup copies of any files you change, especially .htaccess. And be ready to test that everything is working immediately after uploading changes to your .htaccess file(s). It's just good practice. Also, to verify the 301 (permanent) server-response, check your site using an online server-header checker.

Require the www

If you would rather *require* the www-prefix, the code is similar:

```
# require www
<IfModule mod_rewrite.c>
      RewriteEngine On
      RewriteCond %{HTTP_HOST} !^www\. [NC]
      RewriteCond %{HTTP_HOST} ^(.*)$  [NC]
      RewriteRule ^(.*)$ http://www.%1/$1 [R=301,L]
</IfModule>
```

Again, this goes in the root directory, with no editing required. Notice also we're sending the 301-permanent header, so search-engines, apps, and discerning visitors will understand that you actually want the "www" included for your URLs.

To better understand how this technique operates, let's break it down, line-by line:

1. Check that the required module is available
2. Enable the rewrite engine (not required if enabled elsewhere in the .htaccess file)
3. Check for the www-prefix (match the request only if "www" is not included)
4. Capture the domain-name (via "`HTTP_HOST`") as a variable ("`$1`")
5. Capture the request-string and rewrite the URL to include the www-prefix

That's it in a nutshell. For a closer look at the process of URL-canonicalization, check out the articles mentioned in the footer-area of this section.

5.2 Redirecting broken links

Here's the scene: you have been noticing a large number of 404 errors (page not-found) referred by `example.com`. Upon further investigation, you realize `example.com` includes a number of misdirected links to your site. The links may resemble legitimate URLs, but because of typographical or markup errors, they are broken, leading to nowhere and producing a 404-error for every request. Ugh. So much potential wasted on broken links.

Or, another painful scenario would be a single broken link on a highly popular site. For example, you may have one of your best posts mentioned on the Google homepage, but the URL contains a simple typo, say something like this:

```
http://example.com/path/toyour/article/
```

Using this mistyped URL for the link, Google would be sending a ton of traffic to a nonexistent resource on your server, which would generate a ton of 404 errors. Of course, the first thing you should do when discovering such a link is contact the Webmaster of the linking site. Depending on the site, you may resolve the issue with a simple email. If that fails to work, it's time to take matters into your own hands.

Fortunately, we can use Apache's powerful **mod_rewrite** to redirect any broken links to the correct resource. Just add the following code to the site's root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_URI} ^/path/toyour/article [NC]
      RewriteRule .* http://example.com/path/to/your/article/ [R=301,L]
</IfModule>
```

As we'll see in section 6.1, many redirects are more easily accomplished using **mod_alias** and `Redirect` or `RedirectMatch`, which enable us to do it this way:

```
<IfModule mod_alias.c>
      RedirectMatch ^/path/toyour/article http://example.com/path/to/your/article/
</IfModule>
```

Either method works fine, but **mod_alias** only looks at the URI of the request, so you've got to know the exact mistyped URL for each broken link that you want to fix. In cases where external sites are causing many 404 (or other) errors, mod_rewrite can match against other variables, such as the HTTP referrer, which is `Google.com` in our wishful example.

Redirect all (broken) links from an external site

Let's say you've noticed in your access log that "`problem-domain.com`" is misconfigured and sending all sorts of misdirected traffic to your site. And not just one or two mistyped URLs — they're sending hundreds of visitors to nonexistent pages on your site. Sure you can try contacting the site's administrator and try to resolve the issue, and while you wait forever to hear a response, you can use **mod_rewrite** to handle the situation locally.

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_FILENAME} .*
      RewriteCond %{HTTP_REFERER} problem-domain\.com [NC]
      RewriteRule .* http://problem-domain.com/ [R=301,L]
</IfModule>
```

As-is, this technique simply redirects all traffic from the problem-domain back to itself. It's an elegant solution that should help get the attention of whoever should be fixing the issue. To use, just edit both instances of the "`problem-domain.com`" and place into root .htaccess.

Also, instead of returning traffic to `problem-domain.com`, it may be useful to redirect it to the

mod_rewrite – Apache HTTP Server
http://htaccessbook.com/p

Redirect All (Broken) Links from any Domain via HTAccess: http://htaccessbook.com/2x

Tip: to match both uppercase and lowercase letters, prepend the regex with "`(?i)`", for example:

```
RedirectMatch (?i)^/image(.*) /ftp/pub/image$1
```

Also works with the `AliasMatch` directive.

See section 10.2 to learn how to setup various types of access, error, and rewrite logs.

Note that the `RewriteEngine` must be "`on`" for mod_rewrite directives to work. See section 3.1.

home page, sales page, or other strategic location. Take a second to look at the `RewriteRule`. The ".*" matches all requests that satisfy the two rewrite-conditions. The URL "`http://problem-domain.com/`" is where the traffic is being sent. Change it to the URL of your homepage or wherever makes sense.

You could also send the requests to a script for custom processing, or even keep it simple by returning a 403 "Forbidden" response by replacing the current `RewriteRule` with this:

```
RewriteRule .* - [F,L]
```

A trick that I use in this situation is to redirect problem-traffic to a simple text-file that explains the situation, something like this:

> *Hello!*
> *The site that sent you here – problem-domain.com – is misconfigured and causing problems. Please contact the administrator of problem-domain.com to resolve the issue. Thanks.*

Then in my .htaccess file, I replace the `RewriteRule` in the previous technique with something like this:

```
RewriteRule .* http://perishablepress.com/note.txt [R=301,L]
```


You may be surprised at the effectiveness of this method — it's harder to ignore than an email.

Once everything is in place, verify that the code is working by visiting the problem-domain and clicking on any of the links to your site. If you get the desired response, you're in business. Now take a break while you wait for the site-administrator to return your email.

Redirect a few external links

To redirect only a few broken links instead of all traffic coming from a particular site, use Apache's excellent `Redirect` or `RedirectMatch` directives instead. For example, if some external site is linking to a post that doesn't exist, you can redirect the specific request:

```
Redirect 301 /some/post/that/doesnt/exist/ http://example.com/
Redirect 301 /some/post/that/once/existed/ http://example.com/
Redirect 301 /some/post/that/may/have/existed/ http://example.com/
```

Likewise, to redirect more than one variation of the nonexistent post, use `RedirectMatch`:

```
RedirectMatch 301 /some/post/that/ http://example.com/
```

The main difference between these directives is that the latter redirects any request that contains the string, "`/some/post/that`", whereas the former redirects only the specific request, "`/some/post/that/doesnt/exist/`". We'll explore redirecting stuff in Chapter 6 — for now, let's continue with some additional SEO techniques.

Online Redirect Checker
http://htaccessbook.com/2y

5.3 Cleaning up malicious links

One thing you may notice while examining your access logs involves query-strings appended to your URLs. It's not that common, but frustrating to see stuff like this:

```
http://example.com/?whatever
http://example.com/?something
http://example.com/?brandname
```

As seen in the screenshot, apparently Google considers such URLs as valid even though there is no matching resource or functionality for specific query strings. That is, the server resolves such requests, so Google includes them in the search index. Depending on who or what is linking to you, many of your site's pages could be indexed with some random query string appended to the URLs.

What's the worst that can happen? I suppose the worst that could happen is that someone could link to your site with a threatening or obscene query string. Here are some hypothetical examples:

Perishable Press
Perishable Press provides high-quality, in-depth articles on web design and ... owi
web **page**, replete with corporate linkage and poor grammar. ...
perishablepress.com/press/page/17/?scamdex

Perishable Press
Oct 1, 2008 ... New design in effect at **Perishable Press**. With this **17th** ... Apparer
of links at the bottom of each **page** — the menu, ...
perishablepress.com/press/page/10/?scamdex - Cached

Perishable Press
Oct 11, 2005 ... Return to the **Perishable Press** main index **page**. [Custom 500-Er
page] ... by Jeff Starr on Saturday, September **17**, 2005 ...
perishablepress.com/press/page/41/?scamdex - Cached

Perishable Press
Perishable Press provides high-quality, in-depth articles on web design and ... as
Meyer reset and a link to Eric's official CSS Reset **page**. ...
perishablepress.com/press/page/15/?scamdex - Cached

Perishable Press
Perishable Press provides high-quality, in-depth articles on web design and ... Her
this basic template looks in the current web **page 1** (using ...
perishablepress.com/press/page/2/?scamdex - Cached

Perishable Press
Perishable Press provides high-quality, in-depth articles on web design and ... You
download the Arabic version via the Coldform home **page**. ...
perishablepress.com/press/page/16/?scamdex - Cached

As discussed in my article, "Clean Up Malicious Links with HTAccess": http://htaccessbook.com/zz


```
http://starbucks.com/?overpriced-coffee
http://www.wireless.att.com/?horrible-service
http://www.house.gov/?corrupt-politics
```

Then as search-engines crawl and index these valid pages, the URL with the malicious query-string would begin replacing the original URLs in the search results. Granted this is all hypothetical, but as they say, "if it happened to me, it can happen to anyone". In my case, one "scamdex" link was all it took for Google to index all sorts of pages with the appended query-string. Fortunately, there are numerous ways to clean up sloppy and/or malicious incoming links. Here is how I did it with a simple slice of .htaccess:

```
<IfModule mod_rewrite.c>
      RewriteCond %{QUERY_STRING} querystring [NC]
      RewriteRule .* http://example.com/$1? [R=301,L]
</IfModule>
```

Just place into your web-accessible-root .htaccess file and replace the "`querystring`" with whatever is plaguing you, and also replace "`example.com`" with your site URL. Adding more query-strings is easy, just replace the `RewriteCond` with something like this:

```
RewriteCond %{QUERY_STRING} (apples|oranges|bananas) [NC]
```

It turns out that "scamdex" is a valuable resource for email-scam prevention: http://www.scamdex.com/

What removes the query-string from the URL? The question-mark "?" in the `RewriteRule`

Note that the `RewriteEngine` must be "`On`" for mod_rewrite directives to work. See section 3.1.

And then replace the fruit-names with whatever query-string(s) needed. After implementing this technique, search-engines should get the message and remove the specified URLs from the search-results.

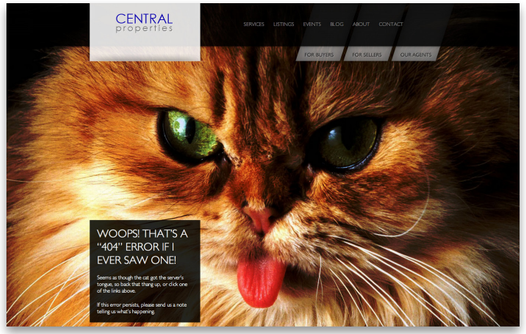
5.4 Cleaning up common 404 errors

As we'll see in the chapter on security, rogue scripts and malicious bots like to request the same nonexistent resources *over and over and over again*. Common examples are requests for `favicon.ico` and `robots.txt` files in random directories. Idiot bots will hammer your site looking for these common files in weird places.

Fortunately, they're easily dealt with using .htaccess. Let's look at a specific example, the commonly requested slew of "mobile" versions of your site:

```
http://example.com/iphone
http://example.com/mobile
http://example.com/mobi
http://example.com/m
```

If these pages don't exist on your site, bad bots will continue to drain server-resources while repeatedly



Guide to 404 error-pages:
http://www.404errorpages.com/

Fabulous 404 error-pages:
http://fab404.com/

Screenshot is of Central Properties' 404-page:
http://centralpropertiesdc.com/fab404

making these requests throughout your site's directory structure. Needless to say, this is an incredible waste of time, bandwidth, and server resources.

The best solution is for all bots to stop "assuming and guessing" at expected URLs, but that will never happen so it's up to us, the *League of Responsible Webmasters*, to handle the situation ourselves.

Deny all requests for non-existent mobile content

If your site is plagued with requests for nonexistent "mobile" content, slap this code into your site's root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_URI} /iphone/?$ [NC,OR]
      RewriteCond %{REQUEST_URI} /mobile/?$ [NC,OR]
      RewriteCond %{REQUEST_URI} /mobi/?$ [NC,OR]
      RewriteCond %{REQUEST_URI} /m/?$ [NC]
      RewriteRule .* http://example.com/ [R=301,L]
</IfModule>
```

As-is, these **mod_rewrite** rules redirect all matching "mobile" requests to the homepage. It works out of the box, but it's smart to fine-tune to match your site's actual traffic activity.

Note: If you are using WordPress, place the .htaccess rules *before* the permalink rules.

Stop 404 Requests for Mobile Versions of Your Site
http://htaccessbook.com/30

See section 6.2 for the full story on mod_rewrite.

Universal redirect for nonexistent files

Similar to the previous method, here is a more generalized way* to redirect virtually anything that's giving you grief, such as requests for nonexistent `favicon.ico` and `robots.txt` files. The following technique works in any directory:

```
<IfModule mod_rewrite.c>
      RewriteEngine on
      RewriteCond %{REQUEST_URI} !^/favicon\.ico [NC]
      RewriteCond %{REQUEST_URI} favicon\.ico [NC]
      RewriteRule .* http://example.com/favicon.ico [R=301,L]
</IfModule>
```

To implement this technique, replace both instances of "`favicon.ico`" with any file that you would like to redirect, and then edit the `RewriteRule` with the target URL. Here is a summary of the directives used in this technique:

1. Check for the required Apache module, **mod_rewrite**
2. Enable rewriting by activating the `RewriteEngine`
3. Check if the requested URI is the actual file
4. Match any nonexistent URI that contains the target file
5. Redirect any matching URI to the actual file
6. Close the conditional module container

We'll see an even easier way to redirect favicons and other common files using mod_alias in section 6.1.

This is a solid example of how **mod_rewrite** enables us to redirect just about any request, regardless of how complicated or specific. And speaking of redirecting stuff, that's exactly what's in store for the next chapter...

chapter 6

.redirecting stuff

In my experience working with .htaccess, the most commonly used technique involves redirecting stuff from point "A" to point "B". Whether it's an entire site or directory, specific URLs, or even conditional requests, .htaccess is an ideal way to do the job.

Apache provides two powerful modules for redirecting stuff with .htaccess: **mod_alias** and **mod_rewrite**. In many cases, **mod_alias**' simple `Redirect` and `RedirectMatch` directives are more than sufficient; and when they're not, **mod_rewrite**'s powerful `RewriteCond` and `RewriteRule` will get you there. In this chapter, you'll learn how to use these techniques to redirect virtually anything from anywhere to anywhere.

mod_alias	mod_rewrite
`Redirect`	`RewriteCond`
`RedirectMatch`	`RewriteRule`

Apache Module mod_alias
http://htaccessbook.com/33

Apache Module mod_rewrite
http://htaccessbook.com/p

6.1 Redirecting with mod_alias

One of the most useful techniques in my .htaccess toolbox involves URL redirection using **mod_alias**' `Redirect` and `RedirectMatch` directives. Here is the general syntax for `Redirect`:

```
Redirect [status] URL-path URL
```

So for example, if we want to redirect "`this-page.html`" to "`that-page.html`" using a 301 "Moved Permanently" server-response, we put this into the root .htaccess file:

```
Redirect 301 /this-page.html http://example.com/that-page.html
```

The key difference between `Redirect` and `RedirectMatch` is pattern-matching. With `Redirect`, there is a strict one-to-one relation between the matched request and the redirect-target. That is, `this-page.html` is the only request that will be redirected using `Redirect`. If we want to all pages in, "`/this-directory/`", we can use `RedirectMatch` instead:

```
RedirectMatch 301 /this-directory/ http://example.com/that-page.html
```

With this, any file or page located in `/this-directory/` is redirected to `that-page.html`. Using one of these techniques, many types of redirects are possible. Let's look at some useful techniques using **mod_alias**' `Redirect` and `RedirectMatch` directives.

Other status-codes may be used, such 302 "Temporary", or 410 "Gone" status. Also, instead of writing "301" in the directive, we could use the word "permanent" instead.

In section 6.2 we see how to redirect these same types of requests using mod_rewrite.

Apache mod_rewrite & mod_alias tricks you should know
http://htaccessbook.com/31

Rewriting & Redirecting with mod_rewrite & mod_alias
http://htaccessbook.com/32

Redirecting subdirectories to the root directory

A common redirect scenario involves redirecting a subdirectory to the root directory. Using the `Redirect` directive to do this, only requests for the "`/blog/`" directory itself are redirected to the homepage:

```
Redirect 301 /blog/ http://example.com/
```

If there are other files or pages contained within the `/blog/` directory, they will not be redirected. So to redirect those as well, we use `RedirectMatch` instead*:

```
RedirectMatch 301 /blog/ http://example.com/
```

Now any request containing "`/blog/`" will be redirected to the homepage*, including URLs such as these:

```
http://example.com/blog/
http://example.com/blog/this-file.html
http://example.com/blog/that-post/
http://example.com/user/blog/
http://example.com/user/blog/this-file.html
http://example.com/user/blog/that-post/
```


Just a reminder to include `<IfModule>` directives when possible. Some techniques in this section omit them for the sake of saving space.

With either method, `Redirect` or `RedirectMatch`, the target can be the homepage, any page on the site, or even any page on another site.

We want to redirect the first three of these URLs, but not the last three. Fortunately it's easy to restrict the regular-expression to match only requests for which `/blog/` is located in the root directory. We do this by prefixing the URL-path with the caret-symbol, "`^`":

```
RedirectMatch 301 ^/blog/ http://example.com/
```

The caret denotes the beginning of the URL request*, so for example, only the first three of our example URLs will be matched and redirected.

Removing a subdirectory from the URL

Following from the previous technique, it may be the case that you've actually moved the files contained in the `/blog/` directory, say to the root directory. Something like this:

```
http://example.com/this-file.html
http://example.com/that-post/
```

Moving content around like this is common, but when you do so, any links that point to the files in the old `/blog/` directory result in a 404-error. In redirecting such requests to their new locations, we're essentially removing the `/blog/` directory from the URL. Such that:

`http://example.com/blog/some-page/` **redirects to** `http://example.com/some-page/`

See the .htaccess Character Definitions in section 2.7 for more information about the caret "`^`" and friends.

To make this happen with .htaccess, we use `RedirectMatch`:

```
RedirectMatch 301 ^/blog/(.*) http://example.com/$1
```

Here, we are capturing as a variable the key part of the request, and then using it to specify the target URL. This demonstrated one of the great strengths of `RedirectMatch`: the ability to use parts of the request-string* to define the target location.

Redirect common 404-requests to canonical resources

That sounds like a mouthful, but we've already seen several examples of this*. There are many rogue bots on the Web that *hound* websites for nonexistent files. Examples include `robots.txt`, `favicon.ico`, `apple.png`, `readme.txt`, `sitemap.xml`, `humans.txt`, and many others, including imaginary files and other malicious URL-requests. Don't believe me? Check your logs and see for yourself. It's happening *constantly*, unless you do something about it.

Fortunately `RedirectMatch` enables us to eliminate common 404-requests by redirecting them to resources that actually exist. Let's use `favicon.ico` and `robots.txt` as examples*:

```
RedirectMatch 301 /(.*)/favicon.ico$ http://example.com/favicon.ico
RedirectMatch 301 /(.*)/robots.txt$ http://example.com/robots.txt
```

Each of these directives redirects all requests to the canonical resource, and works great at keeping traffic on target and helping lost bots get to where they're trying to go. For example, with the previous directives, all of the following requests will reach their goal:

```
http://example.com/some-directory/favicon.ico
http://example.com/another-directory/favicon.ico
http://example.com/that-old-post-you-wrote/favicon.ico
http://example.com/2012/06/29/favicon.ico
http://example.com/archives/favicon.ico
```

And that's going to help keep things running optimally on the server, conserving bandwidth, memory, and other resources while converting 404 and other errors into legitimate traffic. It's a win-win and well-worth the effort.

More rewriting tricks with mod_alias

By now, the basic technique of redirecting stuff with mod_alias should be clear. Once you understand the basic syntax and a little regex, you're ready to create your own redirects. Here are some further techniques to help save time while working on your own rules.

Redirect an entire website to any location

When redirecting one site to another, there are several ways to go about it:

- Redirect each request to the same resource at the new site

The request-string is the part of the URL included after the domain-name, for example: `http://example.com/request-string/`

In section 6.2 we see how to redirect these same types of requests using mod_rewrite.

Another way to write these directives is to use the negative-lookbehind, which is written as "`(?<!^)`". For example, these rules do basically the same thing:

```
RedirectMatch 301 (?<!^)/favicon.ico$ http://example.com/favicon.ico
RedirectMatch 301 (?<!^)/robots.txt$ http://example.com/robots.txt
```

SEO Redirects without mod_rewrite
http://htaccessbook.com/34

- Redirect all requests for the old site to the new site's homepage (or other specific page)
- Custom redirecting of old pages to new pages

The first case is simple and easily accomplished using **mod_alias**' `Redirect` directive. For example, to redirect "`domain.org`" and all of its files to the same files at "`example.com`", place the following directives into the root .htaccess file of `domain.org`:

```
Redirect 301 / http://example.com/
```

Then check a few specific pages to verify that they're redirecting properly. Note that we're sending a 301 "Permanent" server-response to let search-engines, browsers, and aware-visitors know that the change is permanent. To redirect with a "Temporary" response, change the "301" to a "302" or even "503" (Service Unavailable) to indicate site-maintenance.

The second case in our list involves redirecting an entire site (all pages, files, etc.) to some specific page. This is often used to funnel a site's page-rank to some domain for SEO purposes. The only difference in this case is that we use `RedirectMatch` instead of `Redirect`:

```
RedirectMatch 301 / http://example.com/specific-page.html
```

To send all the requests to the homepage, simply remove the "`specific-page.html`". Lastly, for the third case in our list — custom redirecting of old pages to new pages — some custom rules will need to be configured to meet your specific needs. Here are some techniques that should help.

Note that you can change the response for either `Redirect` or `RedirectMatch`.

Redirect a single file or directory

In the previous techniques, we're redirecting all requests from the old domain. But what if we just want to redirect a single file or page? Easy. Here are a few examples:

```
Redirect 301 /path/to/old-page.html http://example.com/
Redirect 301 /path/to/old-page.html http://example.com/path/to/new-page.html
RedirectMatch 301 ^/path/to/old-post/?$ http://example.com/path/to/new-post/
RedirectMatch 301 ^/path/to/old-post/?$ http://example.com/
```

At this point, these redirects should be self-explanatory, following the same general formula: "`[Redirect or RedirectMatch] [status] [old-resource] [new-resource]`". The one thing that's new here is the question-mark in the third directive. It's telling Apache that the trailing-slash "`/`" is optional and may be excluded before the end of the string, "`$`".

Redirecting multiple files

There are several common types of multiple-file redirects: redirect all files in a directory, redirect all files of a particular type, and redirect all files with a similar name. Let's take a look at each of these techniques before moving on to the next section.

Note that "`[new-resource]`" should be a full URL and "`[old-resource]`" is a request string or some part of it.

See section 2.7 for the .htaccess Character Definitions.

Redirecting all files in a directory
For example, to redirect all requests for files contained in a directory named "pancakes" to an identical directory in another location, add the following to the root .htaccess file:

```
RedirectMatch 301 /pancakes/(.*) http://example.com/$1
```

Redirecting all files of a particular type
For example, to redirect all requests for files with an ".html" or ".htm" extension to similar-named directories, add the following to the root .htaccess file:

```
RedirectMatch 301 /(.*)\.html?$ http://example.com/$1/
```

Redirecting all files with a similar name
For example, to redirect all requests for files that aren't in the root directory and include the string "sitemap" anywhere in the name, add the following to the root .htaccess file:

```
RedirectMatch 301 /(.*)/sitemap(.*)/?$ http://example.com/sitemap$1
```

Advanced redirecting with RedirectMatch

So far we've seen RedirectMatch put to good use, but there is much more that is possible. This is mostly the result of the built-in regular-expression pattern-matching, so the more you familiarize yourself with the "art of regex," the more sophisticated rewrites you'll be able to accomplish using RedirectMatch. Let's look at some examples of what's possible using mod_alias for redirects.

Combine multiple redirects into one

If you have multiple RedirectMatch rules all pointing to the same URL, it's possible to consolidate them into a single directive. Consider the following three redirect-rules:

```
RedirectMatch 301 ^/tag/notes/(.*) http://example.com/tag/asides/$1
RedirectMatch 301 ^/tag/links/(.*) http://example.com/tag/asides/$1
RedirectMatch 301 ^/tag/news/(.*) http://example.com/tag/asides/$1
```

Here we're basically merging three tags — "notes", "links", and "news" — into a single "asides" category. For each, we capture the key part of the request as a variable, and then use the variable to create the target URL. Doing it this way is fine, but you can optimize things by combining these three rules into one:

```
RedirectMatch 301 ^/tag/(notes|links|news)/(.*) http://example.com/tag/asides/$2
```

Here, we use the vertical-pipe "|" in specifying that either notes, links, or news should be matched in the URL-request. Note also the "$2" appended to the target-URL. Let's look at why we're doing that and how it works.

To include requests for "/pancakes" (no trailing slash), modify the directive like so:
`RedirectMatch 301 /pancakes(.*) http://example.com$1`

When matching a literal dot, as in the ".html" file-extension, escape it with a backslash "\".

Without the first "/(.*)/" before "sitemap", requests for the actual file will cause an infinite request-loop.

A cheat-sheet helps :) Here's a good one:
http://htaccessbook.com/35

Friendly neighborhood-reminder to use <IfModule> directives whenever possible. They are omitted in many of the techniques in this section to save space.

More information about regular expressions:
http://htaccessbook.com/7k

Also, remember to make backups before making any changes to your .htaccess file(s).

Using multiple variables with RedirectMatch

In the previous section, notice that the `RedirectMatch` rule captures two variables, each surrounded with parentheses:

- Variable 1 = `(notes|links|news)` = `$1`
- Variable 2 = `(.*)` = `$2`

This is why we append the "`$2`" to the target URL in our example. If we had used the first variable "`$1`" instead, the server would append the wrong variable to the target URL, causing a 404-error. When using `RedirectMatch`, it's important to understand the order in which variables occur in the directive. Here is the general syntax for multiple variables:

```
RedirectMatch 301 ^/path/(.*)/(.*)/(.*)/ http://example.com/$1/$2/$3/
```

That's the basic idea, and I've used up to nine variables in a single rule — it seems to break things once double-digit numbers are used for the variables. And when capturing the variables, the key is the parentheses, not the contents — you can use regular expressions to match against anything and use it as a variable. We see this in our previous example:

```
RedirectMatch 301 ^/tag/(notes|links|news)/(.*) http://example.com/tag/asides/$2
```

The ability to use variables like this makes `RedirectMatch` a flexible, convenient way of

In the first variable, pipe-separators are used as "or" conditions, so that "`$1`" is equal to "`notes`", "`links`", or "`news`". See section 2.7 for the Character Definitions.

To map other parts of the filesystem to the `public/www` directory, use Alias: http://htaccessbook.com/36

In this directive, the caret-symbol "`^`" denotes the beginning of the request-string, which is the part of the URL that appears after the domain-name.

redirecting stuff with .htaccess. But there are limits to its reach, so when more power is needed, we summon one of Apache's best features, **mod_rewrite**.

6.2 Redirecting with mod_rewrite

We've seen how **mod_alias**' `Redirect` and `RedirectMatch` enable a wide range of basic redirecting, but there are situations where more specificity is required. Whereas **mod_alias** techniques consist of *single* directives, **mod_rewrite** directives may be *combined*, providing much control over the rewrite-process and more sophisticated redirects.

Of course, **mod_rewrite** can also be used to achieve any of the redirects that can be done with `Redirect` or `RedirectMatch`. Here are some of the same techniques from the previous section, rewritten using the `RewriteRule` directive:

```
RewriteRule /pancakes/(.*) http://example.com/$1 [R=301,L]
RewriteRule /(.*)\.html?$ http://example.com/$1/ [R=301,L]
RewriteRule /sitemap(.*)/?$ http://example.com/sitemap$1 [R=301,L]
```

These rules work fine, but they're more complicated than `RedirectMatch`. As a general rule, **mod_alias** should be used whenever possible, and **mod_rewrite** used for everything else. Here is a rundown on some of the most-useful **mod_rewrite** techniques.

Apache mod_rewrite module http://htaccessbook.com/p

Forum thread about some of the differences between mod_alias & mod_rewrite: http://htaccessbook.com/84

For mod_rewrite directives to work, the module must be enabled with "`RewriteEngine On`" placed at the top of the .htaccess file (or in httpd.conf). It's not necessary to include the `RewriteEngine` directive more than once. See section 3.1 for more information.

Basic example of mod_rewrite

Before jumping into the straight-up **mod_rewrite** techniques, let's look at a basic example.

Let's say that you want to redirect your site's RSS feed to one that you've set up at FeedBurner. If we tried doing this with `RedirectMatch`, all requests for our feed would be redirected to FeedBurner, including FeedBurner itself. To avoid an *infinite loop* for FeedBurner while redirecting all other feed-requests, we use `mod_rewrite`:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_URI} ^/feed/ [NC]
      RewriteCond %{HTTP_USER_AGENT} !(FeedBurner|FeedValidator) [NC]
      RewriteRule .* http://feeds.feedburner.com/yourfeedname [L,R=302]
</IfModule>
```

Let's go through this line-by-line to see what's happening:

1. Check for the required rewrite module
2. Check the requested URI, to see if it begins with "`/feed/`"
3. Check the user-agent for the request, to see if it's from FeedBurner
4. If the request is not from FeedBurner, redirect to the FeedBurner feed
5. Close the conditional `<IfModule>` directive

For example, the Perishable Press feed is redirected to FeedBurner from http://perishablepress.com/feed/ to http://feeds.feedburner.com/perishablepress

Most of the redirects in this section send a 301 "Permanent" status-code, but when redirecting your feeds to a third-party service like FeedBurner, it's best to use a 302 "Temporary" status-code instead.

That's a basic example showing the general mechanics behind redirecting with **mod_rewrite**. Its syntax is very similar to **mod_alias**' directives, with a few notable exceptions:

- Multiple conditions may be used in determining whether or not to redirect
- The conditions may test against various server variables, such as `REQUEST_URI` and `HTTP_USER_AGENT` in our example
- Various flags and attributes are available for declaring stuff like alphanumeric-casing, server-status, and logical-operators

That's it in a nutshell, but **mod_rewrite** is a vast arena, with entire books written on the topic. To keep things focused, let's continue with some examples and explain things along the way. Yes, **mod_rewrite** does go deep, but for most of the stuff we'll be doing in .htaccess files, we've covered more than enough to dive into the good stuff.

Targeting different server variables

When a client requests a resource from the server, it includes with the URL a bunch of other server-variables, such as `HTTP_USER_AGENT`, `REMOTE_HOST`, and `REQUEST_URI`. While the general focus of a redirect is the requested URL, each of these variables may be used in the test string. This makes it possible to configure rewrite-rules for a wide range of redirect-scenarios. To better understand how to make use of server-variables, here are some examples showing some commonly used techniques.

Such as "The Definitive Guide to Apache mod_rewrite": http://htaccessbook.com/39

Note that many server-variables are easily spoofed. For example, many malicious scripts report a common user-agent to avoid getting blocked by firewalls. So the variables exist and may be used for redirecting stuff, but they shouldn't be taken as an indication of true identity.

Redirecting based on the request-method

Every time a client attempts to connect to your server, it sends a message indicating the type of connection it wishes to make. There are many different types of request methods recognized by Apache. The two most common methods are `GET` and `POST` requests, which are required for "getting" and "posting" data to and from the server. In most cases, these are the only request methods required to operate a dynamic website*. So if we wanted to create a custom log* to record any requests that aren't `GET` or `POST`, we could use this:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} ^(delete|head|trace|track) [NC]
      RewriteRule .* http://example.com/custom-log.php [L,R=302]
</IfModule>
```

Of course, the key to this technique is the `REQUEST_METHOD` in the rewrite condition. If it's on the list, Apache will redirect the request to `custom-log.php` for further processing.

Redirecting based on the complete URL-request

When a client connects to the server, it sends a full HTTP request-string that specifies the request method, request URI, and transfer-protocol version. Here is a typical example:

```
GET blog/index.html HTTP/1.1
```


"Dynamic" websites are those that use a scripting language such as PHP to interact with a database.

See section 10.2 for setting up custom logs.

Eight Ways to Blacklist with Apache's mod_rewrite: http://htaccessbook.com/3a

More great mod_rewrite recipes: http://htaccessbook.com/38

The benefit of checking the entire URL-request (as opposed to just checking the request-string*) is that there are additional parameters to evaluate. For example, if we wanted to redirect all requests that are using HTTP version 1.0, we would check `THE_REQUEST` variable with the following directives:

```
<IfModule mod_rewrite.c>
      RewriteCond %{THE_REQUEST} HTTP\/1\.0 [NC]
      RewriteRule .* http://example.com/custom-log.php [L,R=302]
</IfModule>
```

The new trick here is the escaping certain characters to make them literal. A backslash "\" is used to escape the forward-slash and the dot "." in the rewrite-condition.

Redirecting based on IP-address

Another great way to target specific requests is via the `REMOTE_ADDR` server-variable, which is basically the specified IP-address behind the request*. Here's the general technique:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REMOTE_ADDR} ^111\.222\. [OR]
      RewriteCond %{REMOTE_ADDR} ^123\.456\.789\.0$
      RewriteRule .* http://example.com/custom-log.php [L,R=302]
</IfModule>
```

Refresher course: the request-string is the part of the URL that appears *after* the domain-name. The full HTTP request-string consists of the request-method, requested URI, and HTTP version-number.

If you just want to block based on IP, we can use the methods described in section 7.6. Here is a sneak-peek:

```
<Limit GET POST PUT>
        Order Allow,Deny
        Allow from all
        Deny from 123.456.789
</Limit>
```

In this ruleset, the first rewrite-condition matches an entire range of IPs, while the second condition matches against a single IP-address. In the first case, the test-string is open-ended, so that any IP-address beginning with "`111.222.`" meets the condition. Conversely, the second directive terminates with a dollar-sign "`$`", denoting the end of the regex string and matching against the specific address, "`123.456.789.0`".

Redirect based on the query-string

Whereas static URLs summon pages, their appended query strings transmit data and pass variable-data throughout the domain. Query-string information interacts with scripts and databases, influencing behavior and determining results. They look something like this:

```
http://duckduckgo.com/?q=query+string
```

For dynamic websites, controlling query-string requests via .htaccess is insanely useful, especially when it comes to securing your site against malicious requests. Here is an example showing how to redirect based on the `QUERY_STRING` variable:

```
<IfModule mod_rewrite.c>
      RewriteCond %{QUERY_STRING} ^username [NC]
      RewriteRule .* http://example.com/custom-log.php [L,R=302]
</IfModule>
```

In doing so, query-strings are common targets for malicious scripts and bad bots. Section 7.8 explains how to secure your site against many types of query-string attacks.

Mod_Rewrite Variables Cheatsheet
http://htaccessbook.com/5u

Here we are checking the query-string for the presence of "`username`", and recording the results in a custom-log if found. This is one of many ways to use the query-string data.

Redirect based on the user-agent

Like many other server-variables, the user-agent string may be easily spoofed, so it's not always reliable. Even so, having it available to us for URL-rewriting is enormously useful.

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_USER_AGENT} (evil.bot|idiot.bot) [NC]
      RewriteRule .* http://example.com/custom-log.php [L,R=302]
</IfModule>
```

Here we are recording in a custom-log, any requests claiming a user-agent of "`evil.bot`" or "`idiot.bot`", where the literal dot "." in either rewrite-condition matches any character.

As you can imagine, this is a powerful technique for protecting your site against scrapers, spammers, malicious bots, and other nefarious scumbags.

Redirecting based on other server-variables

Out of all the server-variables not yet covered, there are a few more worth mentioning here: `REQUEST_URI`, `HTTP_COOKIE`, and `HTTP_REFERER`. Let's check 'em out...

See Chapter 7 for some excellent techniques for securing your site against all sorts of malicious requests.

Nice list of server-variables available to the RewriteCond directive: http://htaccessbook.com/3b

REQUEST_URI

The `REQUEST_URI` variable is included in the full request-string, or `THE_REQUEST`. Checking the specific request-URI is useful for canonicalization and SEO endeavors. Here is an example whereby the `REQUEST_URI` is checked for suspicious characters:

```
RewriteCond %{REQUEST_URI} (%0A|%0D|%27|%3C|%3E|%00) [NC]
RewriteRule .* http://example.com/custom-log.php [L,R=302]
```

HTTP_COOKIE

The `HTTP_COOKIE` variable may also be checked for malicious characters:

```
RewriteCond %{HTTP_COOKIE} (%0A|%0D|%27|%3C|%3E|%00) [NC]
RewriteRule .* http://example.com/custom-log.php [L,R=302]
```

HTTP_REFERER

Likewise, we can use the deliberately misspelled `HTTP_REFERER` variable to evaluate the referring URI for the presence of forbidden characters:

```
RewriteCond %{HTTP_REFERER} (%0A|%0D|%27|%3C|%3E|%00) [NC]
RewriteRule .* http://example.com/custom-log.php [L,R=302]
```

Now that we've seen how to target different aspects of the request, let's move on to some more-specific examples of redirecting with **mod_rewrite**.

For example, the "`blog/index.html`" part of the full request-string: "`GET blog/index.html HTTP/1.1`".

Reminder to use `<IfModule>` containers whenever possible. They are omitted in this section to save space.

Canonic... what? It's one of those zany words that says it best. A "canonical" URL is the *definitive* URL for a particular page or resource. Canonicalizing your website is the process of eliminating duplicate content by delivering canonical versions of pages.

Send visitors to a subdomain

This technique uses **mod_rewrite** to force all requests to a subdomain. Edit the "`subdomain`", "`example`", and "`com`" to match your subdomain, domain, and top-level domain respectively. Then add to your root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_HOST} !^subdomain\.example\.com$ [NC]
      RewriteRule (.*) http://subdomain.example.com/$1 [L,R=301]
</IfModule>
```

Redirect only if the file or directory is not found

The "`-d`" (directory) and "`-f`" (file) attributes enable us to redirect the request when the file or directory is not found on the server. To use, edit the path with the desired target-location and add to your site's root .htaccess.

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_FILENAME} !-d [OR]
      RewriteCond %{REQUEST_FILENAME} !-f
      RewriteRule (.*) http://example.com/custom-log.php [L,R=302]
</IfModule>
```

Note that these rewrite-conditions may be used separately. Don't forget to remove the "`[OR]`" flag when only one `RewriteCond` is used, or on the last line for multiple conditions.

See section 2.7 for the .htaccess Character Definitions.

Crazy Advanced Mod_Rewrite Debug Tutorial
http://htaccessbook.com/7f

Browser-sniffing based on the user-agent

The list of user-agents approaches infinity as time goes on, so there's no point in trying to sniff them all, but it can be useful to target some of the major browsers such as Chrome, Firefox, Internet Explorer, Opera, and Safari. And here's precisely that: a set of directives targeting each of the five major user-agents:

```
RewriteCond %{HTTP_USER_AGENT} Chrome [NC]
RewriteRule .* http://example.com/chrome.html [L,R=302]

RewriteCond %{HTTP_USER_AGENT} Firefox [NC]
RewriteRule .* http://example.com/firefox.html [L,R=302]

RewriteCond %{HTTP_USER_AGENT} MSIE [NC]
RewriteRule .* http://example.com/msie.html [L,R=302]

RewriteCond %{HTTP_USER_AGENT} Opera [NC]
RewriteRule .* http://example.com/opera.html [L,R=302]

RewriteCond %{HTTP_USER_AGENT} Safari [NC]
RewriteRule .* http://example.com/safari.html [L,R=302]
```

In this manner, you may sniff out as many (or as few) browsers as needed. See the footer area for more information and resources regarding user-agents and browser-sniffing.


Handy tool for checking your user-agent:
http://whatsmyuseragent.com/


HTTP_USER_AGENT Index
http://htaccessbook.com/3e


No more CSS hacks: Browser sniffing with .htaccess
http://htaccessbook.com/3f


Using .htaccess to Redirect Obsolete Browsers
http://htaccessbook.com/3g

Redirect search queries to Google's search engine

Here's a neat trick that I originally posted at Perishable Press®. The following ruleset will redirect all specified search-requests to Google's search engine.

```
RewriteCond %{QUERY_STRING} search [NC]
RewriteRule (.*) http://www.google.com/search?q=$1 [L,R=302]
```

The first `RewriteCond` checks the query-string for the desired search-string ("`search`" in this example), so edit that to match the string that you are using, and then add the two directives to the root .htaccess file.

Redirect a specific IP-address to a custom page

This technique redirects all requests for a specific page when requested from a specific IP-address. In other words, when a visitor coming from `123.456.789` requests the page "`requested-page.html`", the visitor will be redirected to "`just-for-you.html`".

```
RewriteCond %{REMOTE_HOST} 123\.456\.789
RewriteCond %{REQUEST_URI} /requested-page\.html
RewriteRule .* /just-for-you.html [R=301,L]
```

To use this redirect, edit the IP-address, requested-page, and target-page. Add to the root .htaccess file or relevant directory and enjoy the results.


More fun with RewriteCond and RewriteRule
http://htaccessbook.com/3h

6.3 Site-maintenance mode

"*This site is getting an update. Please check back soon.*" Every web-designer and site-administrator needs a good "*We'll be right back*" page for updates, upgrades, and general site-maintenance. There are plenty of robust, full-featured site-maintenance solutions available on the Web, so let's make one that's as simple as possible.

Features

Here is a list of minimal features required for an awesome "site-maintenance" page:

- Simple as possible — modular, plug-n-play
- Allows access to specific/multiple IP addresses
- Redirects everyone else to a temporary maintenance page
- Sends a 503 "Service Temporarily Unavailable" message
- Sends a "Retry-After" header that specifies when to try again
- Simple configuration, easily customizable

With these basics covered, you can use your skills as a web-designer to customize the appearance and functionality of the maintenance message. Let's begin with the absolute simplest way to handle site-maintenance using .htaccess. Here is the magic code to add to your site's root .htaccess file:

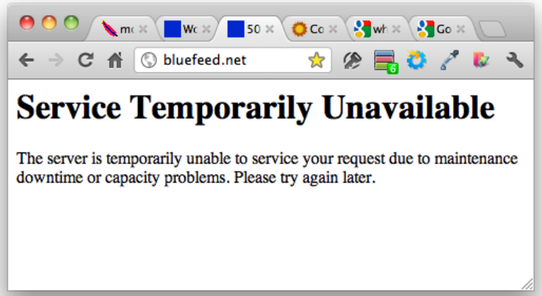
```
GET / HTTP/1.1
Host: bluefeed.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:12.0) Gec
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;c
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0

HTTP/1.1 503 Service Temporarily Unavailable
Date: Sun, 01 Jul 2012 00:05:10 GMT
Server: Apache
Retry-After: 3600
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 85
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

```
# TEMP MAINTENANCE PAGE
<IfModule mod_rewrite.c>
      RewriteEngine On
      RewriteCond %{REMOTE_ADDR} !^123\.456\.789
      # RewriteCond %{REMOTE_ADDR} !^111\.222\.333
      RewriteRule .* - [R=503,L]
</IfModule>
<IfModule mod_headers.c>
      Header always set Retry-After "3600"
</IfModule>
```

That's the magic ticket, and with no other files required. Just add to .htaccess before starting site-maintenance, and then remove (or comment-out) after maintenance is complete.

Nothing could be easier, really. But you do need to edit at least one of the IP-addresses with that of your own. To allow access to additional IPs, uncomment and edit the second `RewriteCond`, and you're good to go. More IPs may be added with new lines.



This technique sends Apache's default 503 "Service Temporarily Unavailable" page, as shown here. We'll see how to customize this and more in the next section.

You can download a zipped version of the site-maintenance file from the Members Area: http://htaccessbook.com/members/

In-depth article on how to close your site for maintenance with .htaccess: http://htaccessbook.com/3i

Although 503 works great, another possibility is the 307 "Site Closed for Maintenance" status-code. A 307 response lets search engines know that it's temporary.

Screenshot shows the site-maintenance code working by sending the 503 response-header.

It may be necessary to place the maintenance rules at the top of your file, before any other directives. If in doubt, verify that it's working by visiting your site via proxy.

Many places online to check your IP address, here is a good one: http://www.whatismyip.com/

During development, it can be useful to disable caching for various types of files. See section 4.3 for more info.

Another good post on redirecting during site-maintenance: http://htaccessbook.com/3l

Customizing

Now that we've got a solid .htaccess technique for redirecting visitors during temporary site-maintenance, let's look at a few ways to customize things.

Send a custom message in plain-text

Apache's default 503-response mentions "capacity problems," which may be taken to mean that something is wrong with the site. If you're not concerned with how the page looks, the easiest way to remedy this is to add the following line to the .htaccess technique on page 97:

```
ErrorDocument 503 "Maintenance mode: update in progress, please check again soon."
```

That directive basically overrides the default Apache response. Any plain-text message may be sent using this method. Unfortunately, even simple markup elements like `<strong>` or `<em>` are not supported, so if you need to do more than plain-text, you can specify any online-resource for the value of your custom `ErrorDocument`.

Use a custom maintenance.html page

Most awesome *websites* also have an awesome *maintenance* page. So create a slick design, save it as "`maintenance.html`", and upload it (along with any CSS/JavaScript files) to the root directory of your site. Next, we need to add two directives to the .htaccess technique given on page 97. The first designates our custom `maintenance.html` file for all 503-responses:

See screenshot on previous page :)

Apache Docs: ErrorDocument directive
http://htaccessbook.com/3j


```
ErrorDocument 503 /maintenance.html
```

Additionally, we need a directive that "un-blocks" requests for `maintenance.html` and its associated files (e.g., CSS, JavaScript), allowing them to be served for all 503-responses:

```
RewriteCond %{REQUEST_URI} !/maintenance [NC]
```

This will allow requests for "`maintenance.html`" and any associated files located in a directory named "`maintenance`". Here is the final code to add to your site's root .htaccess:

```
<IfModule mod_rewrite.c>
      RewriteEngine On
      RewriteCond %{REMOTE_ADDR} !^123\.456\.789
      RewriteCond %{REQUEST_URI} !/maintenance [NC]
      RewriteRule .* - [R=503,L]
</IfModule>
ErrorDocument 503 /maintenance.html
<IfModule mod_headers.c>
      Header always set Retry-After "3600"
</IfModule>
```

With this code in place, visitors will enjoy your customized maintenance page instead of the bleak server message generated by Apache. And that's a good thing.

In the next chapter, we plunge into the *heart* of the book, where you'll learn how to use .htaccess to improve the *security* of your website.

chapter 7

tighten security

Securing your website is mission-critical, and there are many excellent techniques available. Securing your site is all about controlling access — who gets what, who goes where — and that's precisely what .htaccess is all about, enabling strong protection against nefarious scumbags.

Security is the most important aspect of your website. It's the foundation on which everything else is built. If your site is online and available to the public, it may be impossible to secure at 100%, but you can make it very, very difficult for even the most-determined hackers to do any damage. A solid security strategy consists of many *layers* that work collectively to protect your site against malicious activity. These layers of security begin at the server-level and continue all the way up to the UI (user-interface). In this chapter, we focus on the server-level, applying layers of protective techniques to tighten the security of your site.

10 Ways To Beef Up Your Website's Security
http://htaccessbook.com/3m

Ensure basic Web site security with this checklist
http://htaccessbook.com/3n

7.1 Basic security techniques

At this point in the book, we've seen how to use many of the Apache modules that are also used to create strong security measures. In this section, we'll cover some basic security techniques such as improving default settings, preventing file access, and requiring SSL.

So rather than jumping right into the meaty stuff, we'll ease into it with some basic things you can do to improve security. Feel free to use some, all, or none of these techniques, depending on whether or not it makes sense for your particular setup.

Prevent unauthorized directory browsing

As discussed in the Essentials Chapter, it's smart to disable directory-views unless they're specifically needed. The easiest way to check if directory-views are enabled on your site is by visiting any directory on your site that doesn't have an index file*. If you see a listing of the directory's contents, you should lock that down to prevent unwanted access.

Apache makes it easy to disable directory-listings from the .htaccess file*. Here are some

For example, any directory that doesn't include an "`index.html`", "`index.htm`", "`index.php`", or similar.

By default, Apache returns a 403 "Forbidden" status-code when denying access to a directory.

Security tips for web developers
http://htaccessbook.com/30

httpd.conf
Disable .htaccess files

As useful as they are, there are situations where disabling .htaccess files makes good sense. For example, if you have access to the httpd.conf file, disabling .htaccess prevents other users from overwriting default Apache directives.

Disabling .htaccess files also improves performance because the server no longer has to traverse the directory structure with every request.

If this sounds like your situation, and you are able to do so, add the following directives to httpd.conf to completely disable .htaccess files on the server:

```
<Directory />
        Options None
        AllowOverride None
        Order allow,deny
        Allow from all
</Directory>
```

examples showing how to disable, enable, and customize directory-views. To apply any of the directives to a specific directory, place in its .htaccess file. Otherwise root-directory .htaccess is the way to go.

Disable directory-views

```
Options -Indexes
```

Enable directory-views

```
Options All +Indexes
```

Enable directory-views, disable file-views

```
Options All +Indexes
IndexIgnore *
```

Enable directory-views, disable specific files

```
Options All +Indexes
IndexIgnore *.wmv *.mp4 *.avi *.etc
```

Disable listing of sensitive files

```
IndexIgnore .htaccess .??* *~ *# HEADER* README* _vti*
RCS CVS *,v *,t
```


Apache Docs: IndexIgnore Directive
http://htaccessbook.com/3p

Prevent access to specific files

To restrict access to a specific file, edit the file name, "`secret.jpg`", with the name of the file that you wish to protect on the server. Place the code into an .htaccess file contained in the same directory as the protected file.

```
<Files secret.jpg>
      Order Allow,Deny
      Deny from all
</Files>
```

The `Files` directive can also do regular-expressions, so if you need to prevent access to multiple files, just edit the following example with the names of your files:

```
<Files ~ "(secret.jpg|private.html)">
      Order Allow,Deny
      Deny from all
</Files>
```

Prevent access to specific types of files

To restrict access to a variety of file types, edit the file-types in the `FilesMatch` directive with those you want to protect. Place the code into whichever .htaccess file makes sense.

If you have access to Apache's main configuration file, you can restrict the portion of the filesystem to which the `<Files>` directive applies. Alternatively you can simply place the rules in the .htaccess file contained in the target directory.

Note that when targeting a literal filename with `<Files>`, no quotes or tildes "~" are required, just the name of the file. Conversely, if you're targeting multiple files using a regular-expression, it must wrapped in quotes and preceded by a tilde (see the second method on this page).

```
<FilesMatch "\.(htaccess|htpasswd|ini|phps|fla|psd|log|sh)$">
      Order Allow,Deny
      Deny from all
</FilesMatch>
```

Disguise script extensions

To enhance security, disguise scripting languages by replacing actual script extensions with dummy extensions of your choosing. For example, to change the ".foo" extension to ".php", add the following line to your .htaccess file and rename all affected files accordingly:

```
<IfModule mod_mime.c>
      # serve foo files as php files
      AddType application/x-httpd-php .foophp

      # serve foo files as cgi files
      AddType application/x-httpd-cgi .foocgi
</IfModule>
```

Disguise all file extensions

For another obfuscating layer of security, you can rename all of your files to whatever you want, with any extension you want, and then tell Apache to serve them as PHP, Perl, Python, or whatever scripting language you prefer. Two examples and you're good to go.

Apache Module mod_mime
http://htaccessbook.com/3r

AddType not enough? Check out ForceType:
http://htaccessbook.com/3s

The scene: A directory full of extension-less PHP files. The Goal: Serve them as PHP files.
The solution: Apache's ForceType directive placed in the .htaccess file of the same directory.

```
# serve all files as PHP
ForceType application/x-httpd-php
```

Another example, let's say you have a directory containing a bunch of .jpe, .jpeg, .jpg, and possibly some .jgep and .jpge files as well*, and you want to ensure that they're all served as the JPG media-type. Easy, just add the following directive to the nearest .htaccess file:

```
# serve all files as JPG
ForceType image/jpg
```

Require SSL/HTTPS

Here is an excellent method for requiring SSL* on a specific port. To implement, edit the port number "80" if you're using something different, and then add to the site's root .htaccess file.

```
<IfModule mod_rewrite.c>
      RewriteCond %{SERVER_PORT} ^80$
      RewriteRule ^(.*)$ https://%{SERVER_NAME}%{REQUEST_URI} [R=301,L]
</IfModule>
```

Maybe someone was in a hurry when naming the files. Just roll with it.

Apache SSL in htaccess examples
http://htaccessbook.com/3t

Limit size of file-uploads

One way to help protect your server against DOS[1] attacks is to limit the size of file-uploads. Here, we are limiting file-upload size to 5 megabytes. For this directive, file-sizes are expressed in bytes. Note that this code is only useful if you actually allow users to upload files to your site, say via a form or something. Place this line in the nearest .htaccess file:

```
LimitRequestBody 5242880
```

```
# REFERENCE (megabytes to bytes)
# 100 megabytes = 104857600 bytes
#  20 megabytes =  20971520 bytes
#  10 megabytes =  10485760 bytes
#   5 megabytes =   5242880 bytes
#   3 megabytes =   3145728 bytes
#   2 megabytes =   2097152 bytes
```

Quick reference chart for common size-conversions

7.2 Disable trace and track

As discussed previously[2], there are different HTTP-methods used to connect to your server. When enabled, two of these methods, `TRACE` and `TRACK`, are useful in debugging connections, but they also pose a potential security-risk. By exploiting certain browser vulnerabilities, an attacker may manipulate the `TRACE` and `TRACK` methods to intercept your visitors' sensitive data.

The solution, of course, is to disable these two methods on the server, and enable them only when needed and appropriate measures have been taken.

To disable `TRACE` and `TRACK` methods on your Apache-powered webserver, add the following directives to either your main configuration file or root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
      RewriteRule .* - [F]
</IfModule>
```

This technique disables `TRACE` and `TRACK` by checking the `REQUEST_METHOD`, and returning a 403 "Forbidden" response[3] for any `TRACE` and `TRACK` requests.

To take this strategy further, we could limit server-responses to `GET` and `PUT` methods only. `GET` and `PUT` are generally the only methods required, but check first with your host just to be safe. In the previous code, change the `RewriteCond` directive with this one:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK|OPTIONS|HEAD)
      RewriteRule .* - [F]
</IfModule>
```


[1] DOS = Denial of Service
http://htaccessbook.com/3u

[2] If LimitRequestBody isn't working on your server, try adding the following code to a file named "`php.ini`" located in your site's root directory:

```
post_max_size = 4M
upload_max_filesize = 50M
```

[3] As indicated by the "`[F]`" flag at the end of the `RewriteRule`. See section 2.7 for Character Definitions.

7.3 Prevent hotlinking

Hotlinking is **bandwidth-theft**, someone stealing your resources for their own benefit. For example, let's say you posted a brilliant photo of a solar-eclipse on your website. In fact, it's so awesome that other sites start using it too, without your permission, and at your expense. Instead of getting your permission and hosting the file on their own server, lazy and/or ignorant people will just link directly to your image from their web-pages. In doing so, they are effectively *stealing your content* and making you pay for it with your bandwidth, memory, and other resources. Fortunately, it's simple to prevent hotlinking images and other types of files with a simple slab of .htaccess.

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_REFERER} !^$
      RewriteCond %{HTTP_REFERER} !^http(s)?://(www\.)?example.com [NC]
      RewriteRule \.(gif|jpe?g?|png)$ - [NC,F,L]
</IfModule>
```

With this code, the first rewrite-condition checks for an empty value for the `HTTP_REFERER` variable. The second rewrite-condition checks if the referrer is from "`example.com`". If both of these conditions are true, the request is legit, and the server returns the requested image file. If both conditions are not met, then the server responds to the request with a 403-status, thereby preventing use of your files on somebody else's domain.

Usage and customization

To use this anti-hotlinking technique, edit the "`example.com`" to match your domain, and place the code in either of the following locations:

- The root .htaccess file (to protect your entire site)
- The nearest .htaccess file to the files you want to protect (e.g., the `/images/` directory)

As-is, these anti-hotlinking directives protect all GIF, JPG, and PNG images. If you have other types of files, such as music or video files, you can protect them as well by adding their respective file-extensions to the `RewriteRule` directive. Here is an example:

```
RewriteRule \.(gif|jpe?g?|png|mp3|mp4|wmv|flv|avi)$ - [NC,F,L]
```

Once your files are protected, the server will return a 403 "Forbidden" response whenever they are requested from a site other than your own.

To allow other sites — such as FeedBurner, Google Reader, et al — to access your files, determine any URLs involved and add a `RewriteCond` for each of them. For example, to

Creating the Ultimate htaccess Anti-Hotlinking Strategy
http://htaccessbook.com/3v

Anti-hotlink code generator
http://htaccessbook.com/3w

Allow Google Reader Access to Hotlink-Protected Images
http://htaccessbook.com/3x

Allow Feedburner Access to Hotlink-Protected Images
http://htaccessbook.com/3y

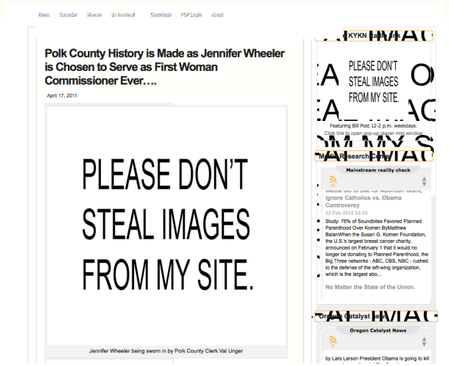
allow Google Reader to access your images, add the following directives to the anti-hotlink code, just beneath the first rewrite-condition:

```
RewriteCond %{HTTP_REFERER} !^http://www.google.com/
reader/(m/)?view/ [NC]
```

Another great way to customize this technique is to deliver a "*don't steal*" image instead of a 403-response. A simple 403-response prevents hotlinking, but it happens quietly behind the scenes and may not get the attention of the hotlinking site's admin. So why not return a "special" image of your choosing for all those greasy hotlink-requests.

```
RewriteRule \.(gif|jpg)$ http://example.com/goofy.jpe
[NC,F,L]
```

Using the previous `RewriteRule` in our anti-hotlinking technique, sites that steal your files, will display the image of your choice[•]. It's a great way to get your point across and have some fun. Shown at right are two recent examples of this technique delivering results and much satisfaction[•].



Note that we remove the "`.jpe`" file-extension from the list. This enables us to serve the "`goofy.jpe`" image without causing an infinite request-loop.

I block content-theives for sport. Shown in the screenshots are two sites that were caught red-handed using the techniques explained in this section. Good times.

Allow hotlinking from a specific directory

After implementing the anti-hotlinking technique, you may want to disable protection for a specific directory[•]. By default, .htaccess rules apply to the directory in which they're located, as well as all subdirectories contained therein. So for example, if you're protecting your entire domain against hotlinking, you may have a directory full of logos and banners for which hotlinking is acceptable. In such a case, the solution is elegant:

```
RewriteEngine off
```

Add that rule to the .htaccess file that's contained in the hotlink-able directory. It disables the rewrite module for that directory, so any anti-hotlink rules will not apply.

Disable hotlinking in a specific directory

As mentioned, the easiest way to disable hotlinking in a particular directory is to add the prescribed directives to the .htaccess file of that directory. Of course, if you'd rather not create another .htaccess file, and just use the one located in the root directory, replace the current rewrite-rule with this one:

```
RewriteRule /protected/(.*)\.(gif|jpe?g?|png)$ - [NC,F,L]
```

Just edit the "protected" with the name of your directory and you're good to go[•].

Three Ways to Allow Hotlinking in Specific Directories
http://htaccessbook.com/3z

What's up with "`jpe?g?`" in the RewriteRule? That's a regex to match all three types: `.jpe`, `.jpeg`, and `.jpg`.

7.4 Password-protect directories

The safest way to host private content is to secure its directory with password-protection. Apache makes it possible to do this with .htaccess, and provides everything needed to customize the configuration to suit your needs.

Although there is much to the story, most cases require only basic password-protection, which you'll learn in this chapter, along with some useful techniques and variations. Before we begin, there are few things you need to know about Apache's various password-protection directives.

Password-protection works in cascading fashion

First, these password-protection tricks apply to the directory in which they are placed. For example, to password-protect your entire site, you would place one of these tricks in the web-accessible root .htaccess file for your site. These directives are applied down the directory structure, in cascading fashion, such that all sub-directories are also protected.

Password-protection requires two files

The second thing you need to know is that, in most cases, there are two parts to any password-protecting a directory: **1)** *the .htaccess file*, and **2)** *the .htpasswd file*. The .htaccess file will contain the password-directives, while the .htpasswd file will contain the required username and an encrypted version of your password.

There are several ways to generate your .htpasswd file. If you are comfortable with Unix, you can simply run the "**htpasswd**" command. For example, entering the following command will create a working password file in the `/home/path/` directory:

```
htpasswd -bc /home/path/.htpasswd username password
```

Placing the password file above the web-accessible root directory is a good security measure. If you examine the file after it has been created, the only thing it will contain is a line that looks similar to this:

```
username:Mx1lbGn.nkP8
```

Instead of running a Unix command, you may prefer to use one of the 200,000 online services providing an online password generator. Regardless of how or where you decide to create your .htpasswd file, keep its location in mind for use in its associated .htaccess file. And yes, you may use one .htpasswd file for multiple .htaccess files used to protect multiple directories.

.htpasswd - Manage user files for basic authentication
http://htaccessbook.com/40

See "HTAccess Password-Protection Tricks" for more in-depth information: http://htaccessbook.com/42

Here is an excellent online tool for generating the necessary elements for a password-protected directory:
http://htaccessbook.com/41

Some hosts enable setup of password-protected content through their control-panel, such as Plesk or cPanel.

The password-prompt dialogue is customizable

The third important thing that you should know before diving into some sweet tricks is that you may customize the message shown on the password prompt by editing the following line in each of the examples in this article:

```
AuthName "Username and password required"
```

By changing the text inside of the quotes, you may use any language you wish for the password prompt. So with those three essential points in mind, let's see some choice techniques to protect your directories with .htaccess.

Basic password protection

To password-protect any directory, place this code in its .htaccess file*:

```
<IfModule mod_authn_file.c>
      AuthName "Username and password required"
      AuthType Basic
      AuthUserFile /home/path/.htpasswd
      <Limit GET POST>
            Require valid-user
      </Limit>
</IfModule>
```


The `<IfModule>` containers in this section assume you are running Apache 2.2 or better. If you're running a version less than 2.2, change the "`mod_authn_file`" to "`mod_auth`".

Apache Module mod_auth (for versions less than 2.2)
http://htaccessbook.com/43

Apache Module mod_authn_file (versions 2.2 or better)
http://htaccessbook.com/44

To use this basic password-protection technique, edit the `AuthUserFile` path to match the location of your .htaccess file. That's about as basic as it gets. Let's move on to something more interesting.

Allow open-access for specific IP-address(es)

To allow open-access for single or multiple IPs* while password-protecting for everyone else, add the following code to the .htaccess file of the directory you would like to protect:

```
<IfModule mod_authn_file.c>
      AuthName "Username and password required"
      AuthType Basic
      AuthUserFile /home/path/.htpasswd
      Require valid-user
      Order Deny,Allow
      Deny from all
      Allow from 111.222.333.444
      Allow from 555.666.777.888
      Satisfy Any
</IfModule>
```

This technique is great during project development, where you want open access with the ability to give others access via password. Edit, remove, or replicate the "`Allow from`" directives to suit your needs*.

HTAccess Privacy for Specific IPs
http://htaccessbook.com/49

In addition to providing access to your team, you may also want to allow access for certain web-services such as validators and the like. Some examples to add to the list:

```
Allow from validator.w3.org
Allow from jigsaw.w3.org
Allow from google.com
```

Password protect specific files

Want to password-protect specific files on the server? Just include the authorization directives in a `Files` container, like so:

```
<IfModule mod_authn_file.c>
      <Files "protected.html">
            AuthName "Username and password required"
            AuthType Basic
            AuthUserFile /home/path/.htpasswd
            Require valid-user
      </Files>
</IfModule>
```

When password-protecting multiple files, use regular-expression syntax for the `Files` directive, replacing the previous one with something like this:

```
<Files ~ "(secret.jpg|private.html|passwords.txt)">
```

In similar fashion, we can password protect all files of a specific type, by writing*:

```
<FilesMatch "\.(avi|flv|mp4|mpeg|mpg|mov|swf|wmv)$">
```

This technique is useful for protecting, say, a directory full of premium videos.

These two directives are essentially the same:

```
<Files ~ "(expression)">
<FilesMatch "(expression)">
```


Allow access to specific files

When password-protecting site content during development, it's nice to leave a page open for visitors to know what's up. Here is how to do just that, allowing access to "`hello.html`":

```
<IfModule mod_authn_file.c>
      AuthName "Username and password required"
      AuthType Basic
      AuthUserFile /home/path/.htpasswd
      Require valid-user
      <Files "hello.html">
            Order Deny,Allow
            Deny from all
            Allow from 123.456.789
            Satisfy any
      </Files>
</IfModule>
```

Once in place, `hello.html` will be accessible to the public, while everyone else except the allowed IP will be prompted for the password. As seen in previous password-techniques, additional IPs may be whitelisted, and additional files protected.

Apache makes it possible to configure just about any password-protection setup needed to protect your files easily and securely. For more in-depth information, see the footer links.

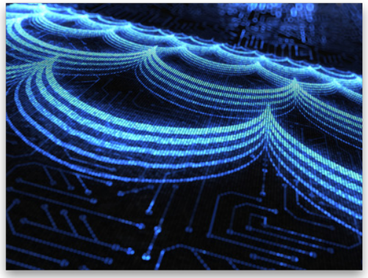
Authentication and Authorization
http://htaccessbook.com/45

Apache Web Login Authentication
http://htaccessbook.com/47

How to Password Protect a Directory on Your Website
http://htaccessbook.com/46

Password Protection for WordPress
http://htaccessbook.com/48

7.5 Block proxy servers



Most of them are horrible, but there must be a *gazillion* proxy services on the Web. They make it easy for visitors to access your site remotely, disguising their true IP address, country, and other data. In my experience, proxies bring some good traffic, but they can also be used by troublemakers to cause problems. Trolls, for example, will get banned by IP, but then return via proxy to continue their idiocy. Further, evil bots and malicious scanning happens via proxy script.

Trying to block proxies by IP is practically futile, as is trying to block by domain-name — the blacklist would grow to be endless, eventually irrelevant, and virtually useless. Rather than blocking proxy-servers by their *identity*, it's better to target their *activity*. By simply denying the various HTTP protocols used by proxy servers, it is possible to block *many* proxy connections.

.htaccess proxy firewall

The simplest way to block a large percentage of proxy visits is to apply a simple firewall. Apache's mod_rewrite enables us to evaluate requests for signs of proxy behavior. By setting up rewrite-conditions for commonly used proxy response-headers, it's possible to filter out much of the "lower-level" proxy noise. The proxy firewall is certainly effective for a cut-n-paste solution, but it won't block everything, especially the "higher-level" or more sophisticated proxies services such as the notoriously hard-to-block site, `hidemyass.com`.

That said, to add the proxy firewall to your site, include the following code in the root .htaccess file. Once uploaded to your server, test its effectiveness by visiting your site via any proxy service. It won't block them all, but compared to blacklisting a million proxies by domain-name or IP-address, it's a lightweight, concise, and effective way to reduce the net volume of proxy visits to your site. Here's the code to add to your root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP:VIA}                  !^$ [OR]
      RewriteCond %{HTTP:FORWARDED}            !^$ [OR]
      RewriteCond %{HTTP:FORWARDED-FOR}        !^$ [OR]
      RewriteCond %{HTTP:X-FORWARDED}          !^$ [OR]
      RewriteCond %{HTTP:X_FORWARDED_FOR}      !^$ [OR]
      RewriteCond %{HTTP:PROXY_CONNECTION}     !^$ [OR]
      RewriteCond %{HTTP:XPROXY_CONNECTION}    !^$ [OR]
      RewriteCond %{HTTP:HTTP_PC_REMOTE_ADDR}  !^$ [OR]
      RewriteCond %{HTTP:HTTP_CLIENT_IP}       !^$ [OR]
      RewriteCond %{HTTP:USERAGENT_VIA}        !^$
      RewriteRule .* - [F]
</IfModule>
```


A delightful romp! "Blacklist Candidate Number 2008-04-27" by yours truly: http://htaccessbook.com/4a

IPs are spoofed easily. And even true IPs change constantly, so don't waste time unless it makes sense.

Blocking by domain or host-name is useful for specific threats, but it's futile trying to block them all.

Difficult to block, but not impossible... using a scripting language such as PHP, it's possible to block even the most notorious proxies... skip ahead to see one way of doing it.

Note that not all proxies reveal the information targeted in these directives, but many of them continue to do so.

Current list of active proxy sites: http://proxy.org/

How to Block Proxy Servers via htaccess http://htaccessbook.com/83

No editing is required, unless you want to send proxy visitors something other than a 403 "Forbidden" response. For example, to send proxy visitors to a script, use this `RewriteRule`:

```
RewriteRule .* http://example.com/proxy-log.php [NC,F,L]
```

Allow only specific proxies

Allowing visits from specific proxy-servers is simply a matter of creating additional rewrite-conditions to "whitelist" proxies by domain-name. For example, if we wanted to allow "`proxy-service.com`", "`another-proxy.com`", and "`proxy.service.com`", we could use this:

```
RewriteCond %{HTTP_REFERER} !(.*)proxy-service.com(.*)
RewriteCond %{HTTP_REFERER} !(.*)another-proxy.com(.*)
RewriteCond %{HTTP_REFERER} !(.*)proxy.service.com(.*)
```

These new conditions are additive, so we're not including an "`[OR]`" flag for any of them. You can edit the domain names to match those on your list, and add or remove new conditions as needed. Once everything looks good, add these directives to the proxy-block rules, between the `RewriteRule` and the last `RewriteCond`.

With this code in place, you will enjoy protection against unwanted proxies while allowing open access to the proxy servers or other referring domains of your choice.

See section 2.7 for character definitions, server-status codes, and free beer. Just kidding about the free beer.

Controlling Proxy Access with HTAccess
http://htaccessbook.com/82

Block Tough Proxies
http://htaccessbook.com/81

Block tough proxies

If the .htaccess method isn't catching some of those tougher proxies, it's worth mentioning that you can use PHP to stop them. If you can create a PHP file, you can use this simple solution:

```
<?php if(@fsockopen($_SERVER['REMOTE_ADDR'], 80, $errstr, $errno, 1))
die("Proxy access not allowed"); ?>
```

No editing required, just include with any PHP web page(s). I'm sure similar techniques are possible using PERL and other languages, but I've digressed aplenty here, so let's move on.

7.6 Controlling IP access

Controlling access based on IP-address is an effective strategy when dealing with *specific* security threats.

For example, let's say you're running a help-forum for the latest tech gadget. Most visitors are cool, but inevitably you're going to get a few trolls who insist on causing problems. We've already seen how to block trolls who are savvy enough to use a proxy, but some trolls *aren't* tech-savvy and are easily blocked via IP.

Such as the previously discussed proxy:
http://www.hidemyass.com/

PERL = Practical Extraction and Report Language
http://www.perl.org/

PHP = PHP: Hypertext Preprocessor
http://php.net/

Specific threats involve persistent IPs and other request data. In such cases it makes sense to block by IP address.

Beyond the individual-IP scenario, the techniques in this section enable you to deny access to entire blocks of IPs. Although I personally don't recommend it, some webmasters get so sick and tired of bad traffic that they'll block entire servers, regions, and even countries. It's all possible with a few lines of .htaccess — let's dive into some examples and take a look.

Blocking and allowing specific IPs

Let's begin with the basics. To block a single, specific IP-address, add the following directives to your site's root .htaccess file, editing the values of the IPs to match those you would like to block. As usual, feel free to remove or replicate as many lines as needed.

```
<Limit GET POST PUT>
      Order Allow,Deny
      Allow from all
      Deny from 123.456.789.000
</Limit>
```

When placed in the root directory's .htaccess file, this code will block all requests coming from that specific IP address. With that code in place, it's easy to block additional IPs:

```
      Order Allow,Deny
      Allow from all
      Deny from 123.456.789.000
      Deny from 111.222.333.444
```


Major IP Addresses Blocks By Country
http://htaccessbook.com/4b

See also "Access Control" in the Apache Docs:
http://htaccessbook.com/4c

Here, we instruct Apache to allow everyone except the denied IPs. Without getting into the tedious logic involved with this technique, suffice it to say that getting the order and syntax of these directives is *critical* to their operation. Here is a quick **rule-of-thumb**:

- **When blocking** specific IPs, use "`Order Allow,Deny`" then "`Allow from all`".
- **When allowing** specific IPs, use "`Order Deny,Allow`" then "`Deny from all`".

To help illustrate the distinction, consider the inverse of our previous technique:

```
      Order Deny,Allow
      Deny from all
      Allow from 123.456.789.000
      Allow from 111.222.333.444
```

This does the exact opposite: instead of allowing all while blocking two IPs, we're denying all and *allowing* two IPs. If you study each method, you'll see why order is important.

Denying and allowing ranges of IPs

There are several effective ways to block a range of IP addresses. The first method blocks an IP-range specified by their CIDR number. It's a useful technique for blocking mega-spammers such as RIPE, Optinet, and others. The second method blocks IPs directly based on wildcard IP-values. Let's take a look...

Tip: save a little space by blocking multiple IP-values on the same line, separated by a space. Some examples:

```
Deny from 123.456.789 111.222.333 999.888.777
Deny from 123.456. 111.222. 999.888.
Deny from 123. 111. 999.
```

Further information about "Order, Allow, and Deny":
http://htaccessbook.com/4d

CIDR = Classless Inter-Domain Routing
http://htaccessbook.com/4e

Denying and allowing based on CIDR number

If, for example, you find yourself adding line after line of `Deny` directives for IPs beginning with the same first few numbers, choose one of them and try a "whois lookup". Listed within the whois results will be the CIDR value representing every IP address associated with that particular network. Thus, blocking via CIDR is an effective way to prevent all instances of the offender's IP from accessing your site. Here is a generalized example for blocking by CIDR:

```
<Limit GET POST PUT>
      Order Allow,Deny
      Allow from all
      Deny from 10.1.0.0/16
      Deny from 80.0.0/8
</Limit>
```

Likewise, to allow an IP-range by CIDR number:

```
<Limit GET POST PUT>
      Order Deny,Allow
      Deny from all
      Allow from 10.1.0.0/16
      Allow from 80.0.0/8
</Limit>
```

Here is a good whois-lookup tool:
http://htaccessbook.com/4f

CIDR = Classless Inter-Domain Routing
http://htaccessbook.com/4e

To use either of these techniques, edit the CIDR number accordingly and place the code in the site's root .htaccess file. Remember to test thoroughly, either directly or by analyzing your server's access logs.

Denying and allowing based on wildcard IP-values

Another effective way to block an entire range of IPs involves truncating digits until the desired range is represented. As an IP address is read from left to right, its value represents an increasingly specific address.

For example, a fictitious IP address of `99.88.77.66` would designate some uniquely specific IP-address. Now, if we remove the last two digits (`66`) from the address, it would represent any address beginning with the remaining digits. That is, `99.88.77` represents `99.88.77.1`, `99.88.77.2`, ... `99.88.77.99`, ... etc.

Likewise, if we then remove another pair of digits from the address, its range suddenly widens to represent every IP address `99.88.x.y`, where "x" and "y" represent any valid set of IP address values. Following this logic, it is possible to block an entire range of IP addresses to varying degrees of specificity.

That's all pretty abstract, so let's look at a specific example. Consider the following set of .htaccess directives:

Tip: to deny or allow access to a specific directory, place these directives in its .htaccess file (instead of root).

The moral of the story is that you should exercise caution and attention to detail whenever blocking multiple IPs.

In this example, you would block 256*256 = 65,536 unique IPs. That sounds like a lot, but it's less than .002% of the 4,294,967,296 possible unique addresses.

```
<Limit GET POST PUT>
      Order Allow,Deny
      Allow from all
      Deny from 99.88.77.
      Deny from 66.55.
      Deny from 44.33.
      Deny from 22.
</Limit>
```

In the first `Deny` directive, we're blocking every IP that begins with "`99.88.77.`", while in the second directive, we're blocking all IPs that begin with "`66.55.`", and so forth. It's a powerful way to block or allow entire IP-ranges. Allowing only specific IPs would look like this:

```
<Limit GET POST PUT>
      Order Deny,Allow
      Deny from all
      Deny from 999.888.777.666
      Deny from 555.444.333.222
      Deny from 111.222.333.444
      Deny from 123.456.789.000
</Limit>
```

This is a very powerful technique, so again, please test thoroughly after implementation.

IP address information at Wikipedia
http://htaccessbook.com/4g

Sending blocked IPs to a custom page

Running a private site is all about preventing unwanted visitors. Here is a quick and easy way to allow access to multiple IP addresses while redirecting everyone else to a custom message page. Here is the basic procedure for this technique:

- First deny access to everyone, then allow access only to the specified addresses.
- Serve everyone who doesn't have access a customized 403 (Forbidden) message.
- Ensure that everyone has access to the customized 403 (Forbidden) message.

All you need is an .htaccess file and a list of IPs for which you would like to allow access. Edit the following code as needed and place into the root .htaccess file of your domain:

```
<Limit GET POST PUT>
      Order Deny,Allow
      Deny from all
      Allow from 123.456.789
      Allow from 456.789.000
</Limit>
ErrorDocument 403 path/custom-message.html
<Files path/custom-message.html>
      Order Allow,Deny
      Allow from all
</Files>
```

See section 8.1 for more information about the `ErrorDocument` directive.

To prepare this code for use on your site, do these three things:

- Edit the IPs to suit your needs, removing or replicating "`Allow`" directives as needed.
- Edit both instances of "`path/custom-message.html`" with the actual path and name.
- That's it. Copy/paste into your site's root htaccess file, upload and test thoroughly.

How does it work? The first ruleset allows only the specified IPs. Then in the middle there, the `ErrorDocument` directive is used to specify a custom error-page, which will be seen by visitors who are denied access. The third ruleset allows everyone access to the custom page.

Miscellaneous rules for blocking IP-addresses

Here are few miscellaneous rules for blocking various types of IP-addresses. For each of the following techniques, edit the `Deny` or `Allow` directives with the desired IP values.

Block a partial-domain via network/netmask values

Here is an example of denying access based on specific network/netmask values:

```
<Limit GET POST PUT>
      Order Allow,Deny
      Allow from all
      Deny from 99.1.0.0/255.255.0.0
</Limit>
```


TCP/IP basics: IP address, Netmask, Network
http://htaccessbook.com/4h

Limit access to Local Area Network (LAN)

Here is an example of denying access to a specific Local Area Network (LAN):

```
<Limit GET POST PUT>
      Order Deny,Allow
      Deny from all
      Allow from 192.168.0.0/33
</Limit>
```

Deny access based on domain-name

```
<Limit GET POST PUT>
      Order Allow,Deny
      Allow from all
      Deny from example\.com
</Limit>
```

Block domain.com but allow subdomain.domain.com

```
<Limit GET POST PUT>
      Order Deny,Allow
      Deny from example.com
      Allow from subdomain.domain.com
</Limit>
```

Wikipedia: "Local Area Network"
http://htaccessbook.com/4i

7.7 Whitelisting access

"Whitelisting" is the *opposite* of "blacklisting". When you create a blacklist of evil IPs, they will be blocked while everyone else enjoys open access. When you create a whitelist of good IPs, only they will be allowed access while everyone else is blocked from your site.

Whitelisting is a proven method of controlling comment-spam, bandwidth-theft, and content-scraping, but there are pros and cons to consider.

The major upside is that whitelisting works. Like some trendy club, if a visitor isn't on "the list" they're not getting in. Another upside is that maintaining a whitelist is easier than maintaining a blacklist, although both require periodic updating as things change.

The big downside to going the whitelist-route, is that *false-negatives* are going to be an issue. There are thousands of different user-agents, and they tend to change as software evolves. Unless your whitelist accounts for every browsing device, denying access to legitimate users is inevitable. It's also possible for hackers to "fake" a legit user-agent, so whitelisting may be effective, but it's no guarantee.

Bottom line: whitelisting is proven to be effective, but only makes sense in certain situations:

- You've got a good idea of which browsers people are using to visit your site
- You're willing to accept a percentage of false-positives as a trade-off for tight security
- You're able to check and update your whitelist as needed to stay current

If that sounds like you, here is an example of an extremely restrictive whitelist that blocks everyone except for the major search engines (Google, Yahoo, MSN, Ask) and popular browsers (Chrome, Firefox, IE, Opera, Safari).

```
<IfModule mod_authn_file.c>
      # Google
      BrowserMatchNoCase Googlebot            allow_access
      BrowserMatchNoCase Mediapartners-Google allow_access

      # Yahoo
      BrowserMatchNoCase Slurp                allow_access
      BrowserMatchNoCase Yahoo-MMCrawler      allow_access

      # MSN/Bing
      BrowserMatchNoCase msnbot               allow_access
      BrowserMatchNoCase SandCrawler          allow_access
```

Code continues on next page...

How to whitelist in .htaccess
http://htaccessbook.com/4k

Opt-in or Blacklist?
http://htaccessbook.com/4j

List of User-Agents (Spiders, Robots, Crawlers, Browsers)
http://www.user-agents.org/

Invite Only: Visitor Exclusivity via the Opt-In Method
http://htaccessbook.com/4l

Best-practice is to not include any extra whitespace for formatting directives, but it doesn't hurt anything and can help to clarify what's happening in the code.

Browser Statistics
http://htaccessbook.com/4n

How to Verify the Four Major Search Engines
http://htaccessbook.com/7m

Top 15 Most Popular Search Engines
http://htaccessbook.com/4m

```
        # Ask
        BrowserMatchNoCase Teoma                allow_access
        BrowserMatchNoCase Jeeves               allow_access

        # Browsers
        BrowserMatchNoCase Chrome               allow_access
        BrowserMatchNoCase Mozilla              allow_access
        BrowserMatchNoCase MSIE                 allow_access
        BrowserMatchNoCase Opera                allow_access
        BrowserMatchNoCase Safari               allow_access

        <Limit GET POST PUT HEAD>
              Order Deny,Allow
              Deny from all
              Allow from env=allow_access
        </Limit>
</IfModule>
```

To use this technique, check your access-logs and site-statistics for other commonly used user-agents. After customizing the list, be prepared to test thoroughly and monitor your traffic-logs for any undesirable activity, false-positives or otherwise. If you take the time to do your research, whitelisting good user-agents is an effective security measure.

2010 User-Agent Blacklist
http://htaccessbook.com/7n

List of User-Agents (Spiders, Robots, Crawlers, Browsers)
http://www.user-agents.org/

7.8 Blacklisting access

Blacklisting bad traffic is one of my favorite ways of protecting websites. Each server-request brings with it many variables that may be evaluated for access, giving you fine-grained control over the traffic of your site.

In this section, you'll learn how to use .htaccess to better secure your site using a variety of methods. These methods may be customized and refined according to the specifics of your particular setup.

Blacklist via the request-method

This first blacklisting method evaluates the client's request-method. Every time a client attempts to connect to your server, it sends a message indicating the type of connection it wishes to make. There are many different types of request methods recognized by Apache. The two most common methods are `GET` and `POST` requests, which are required for "getting" and "posting" data to and from the server. In most cases, these are the only request methods required to operate a dynamic website. Allowing more request-methods than are necessary increases your site's vulnerability. Thus, to restrict the types of request-methods available to clients, we add the following to the site's root .htaccess file:

My articles about blacklisting stuff:
http://htaccessbook.com/40

8 Ways to Blacklist
http://htaccessbook.com/3a

Conditional GET Request
http://htaccessbook.com/4q

Publishing Pages with PUT
http://htaccessbook.com/4p

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} ^(delete|head|trace|track) [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

The key to this rewrite method is the `REQUEST_METHOD` in the rewrite-condition. First we invoke some precautionary security measures, and then we evaluate the request method against our list of prohibited types. Apache will then compare each client request-method against the blacklisted expressions and subsequently deny access to any forbidden requests. Here we are blocking `DELETE` and `HEAD` because they are unnecessary, and also blocking `TRACE` and `TRACK` because they violate the same-origin rules for clients. Of course, I encourage you to do your own research and establish your own request-method policy.

Blacklist via the referrer

Blacklisting via the HTTP referrer is an excellent way to block referrer spam, defend against penetration tests, and protect against other malicious activity. The HTTP referrer is identified as the source of an incoming link to a web-page. For example, if a visitor arrives at your site through a link they found in the Google search results, the referrer would be the Google page from whence the visitor came. Sounds straightforward, and it is.

Unfortunately, one of the biggest spam problems on the Web involves the abuse of HTTP referrer data. In order to improve search-engine rank, spambots will repeatedly visit your site using their spam domain as the referrer. The referrer is generally faked, and the bots frequently visit via `HEAD` requests for the sake of efficiency. If the target site publicizes their access logs, the spam sites will receive a rank-boost from links in the referrer statistics.

Fortunately, by taking advantage of **mod_rewrite**'s `HTTP_REFERER` variable, we can forge a powerful, customized referrer blacklist. Here's our example:

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_REFERER} ^(.*)(<|>|'|%0A|%0D|%27|%3C|%3E|%00).* [NC,OR]
      RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?adult(-|.).*$  [NC,OR]
      RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?poker(-|.).*$  [NC,OR]
      RewriteCond %{HTTP_REFERER} ^http://(www\.)?.*(-|.)?drugs(-|.).*$  [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

Same basic pattern as before: check for the availability of the rewrite module, enable the rewrite engine, and then specify the prohibited character strings using the `HTTP_REFERER` variable and as many rewrite conditions as necessary. In this case, we are blocking a series of potentially malicious characters in the first condition, and then blacklisting any referrer containing the terms "`adult`", "`poker`", or "`drugs`". Of course, we may blacklist as many referrer strings as needed by simply emulating the existing rewrite-conditions. Just don't get carried away — I have seen some referrer blacklists that are over 8000 lines long!

W3C: Same Origin Policy
http://htaccessbook.com/4r

When adding new rewrite-conditions, it's important to omit the "[OR]" flag for the last RewriteCond, otherwise the match is impossible and the rewrite won't happen.

The Ultimate Referrer Blacklist, Featuring Over 8000 Banned Referrers: http://htaccessbook.com/4s

To send blocked referrers to a specific location (file or web-page), replace the RewriteRule with something similar to this (edit the target to whatever you'd like):

```
RewriteRule .* http://example.com/target/ [F,L]
```

Blacklist via cookies

Protecting your site against malicious cookie exploits is greatly facilitated by using Apache's `HTTP_COOKIE` variable. HTTP cookies are chunks of data sent by the server to the web client upon initialization. The browser then sends the cookie information back to the server for each subsequent visit. This enables the server to authenticate users, track sessions, and store preferences. A common example of the type of functionality enabled by cookies is the shopping cart. Information about the items placed in a user's shopping cart may be stored in a cookie, thereby enabling server scripts to respond accordingly.

Generally, a cookie consists of a unique string of alphanumeric text and persists for the duration of a user's session. Apache's **mod_cookie** module generates cookie values randomly and upon request. Once a cookie has been set, it may be used as a database key for further processing, behavior logging, session tracking, and much more.

Unfortunately, this useful technology may be abused by attackers to penetrate and infiltrate your server's defenses. Cookie-based protocols are vulnerable to a variety of exploits, including cookie poisoning, cross-site scripting, and cross-site cooking. By adding malicious characters, scripts, and other content to cookies, attackers may exploit vulnerabilities. The good news is that we may defend against most of this nonsense by using Apache's `HTTP_COOKIE` variable to blacklist characters known to be associated with malicious cookie exploits. Here is an example that does the job:

See section 4.4 for more information on using .htaccess to optimize cookie-behavior for subdomains. Plus there's a ton of tasty cookie-links in the footer :)

Evil Incarnate, but Easily Blocked
http://htaccessbook.com/4v

Related info: "Cookie Protected Directories"
http://htaccessbook.com/4t


```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_COOKIE} ^.*(<|>|'|%0A|%0D|%27|%3C|%3E|%00).* [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

This is as straightforward as it looks. Check for the required rewrite module, enable the rewrite engine, and deny requests for any `HTTP_COOKIE`s containing the specified list of prohibited characters. In this list you will see characters generally required to execute any sort of scripted attack: opening and closing angle brackets, single quotation marks, and a variety of hexadecimal equivalents. Feel free to expand this list with additional characters as you see fit. As always, recommendations are welcome in the .htaccess Forums.

Blacklist via the user-agent

Blacklisting via user-agent is a commonly seen strategy that yields *questionable* results. The concept of blacklisting user-agents revolves around the idea that every browser, bot, and spider that visits your server identifies itself with a specific user-agent character string. Thus, user-agents associated with malicious, unfriendly, or otherwise unwanted behavior may be identified and blacklisted in order to prevent against future access. This is a well-known strategy that has resulted in some extensive and effective user-agent blacklists.

Of course, the *downside* to this method involves the fact that user-agent information is easily forged, making it difficult to know for certain the true identity of blacklisted clients.

Questions about .htaccess? Visit the .htaccess Forums:
http://htaccessbook.com/forums/

Check your user-agent and full header:
http://www.httpuseragent.com/

Such as the classic "The Ultimate htaccess Blacklist"
http://htaccessbook.com/4w

Simulate any user-agent or bot:
http://htaccessbook.com/4u

By simply changing their user-agent to an unknown identity, malicious bots may bypass every blacklist on the Internet. Many evil "scumbots" indeed do this very thing, which explains the incredibly vast number of blacklisted user-agents. Even so, there are certain limits to the extent to which certain user-agent strings may be changed. For example, GNU's Wget and the cURL command-line tool are difficult to forge, and many other clients have hard-coded user-agent strings that are difficult to change.

On Apache servers, user-agents are easily identified and blacklisted via the `HTTP_USER_AGENT` variable. Here is an example:

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_USER_AGENT} ^$                                          [OR]
      RewriteCond %{HTTP_USER_AGENT} (<|>|'|%0A|%0D|%27|%3C|%3E|%00)             [NC,OR]
      RewriteCond %{HTTP_USER_AGENT} (HTTrack|clshttp|archiver|loader|email) [NC,OR]
      RewriteCond %{HTTP_USER_AGENT} (winhttp|libwww\-perl|curl|nikto|miner) [NC,OR]
      RewriteCond %{HTTP_USER_AGENT} (wget|harvest|scan|grab|extract|python) [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

This method works just like the others: check for **mod_rewrite**, enable the rewrite-engine, and proceed to deny access to any user-agent that includes any of the blacklisted character-strings in its name. As with our previous blacklisting techniques, here we are prohibiting angle brackets, single quotation marks, and various hexadecimal equivalents.

GNU Wget: http://htaccessbook.com/7v
cURL and libcurl: http://curl.haxx.se/

How to Write Valid URL Query String Parameters
http://htaccessbook.com/4x

For an alternate way of blacklisting user-agents, check out the "`[USER AGENTS]`" section of the 5G Blacklist:
http://htaccessbook.com/5g

What characters are allowed unencoded in query strings?
http://htaccessbook.com/5a

Additionally, we include a handful of user-agent strings commonly associated with server attacks and other malicious behavior. We certainly don't need anything associated with libwww-perl hitting our server, and many of the others are included in just about every user-agent blacklist that you can find. There are tons of other nasty user-agent scumbots out there, so feel free to beef things up with a few of your own.

Blacklist via the query-string

Protecting your server against malicious query-string activity is extremely important. Query-strings interact with scripts and databases, influencing behavior and determining results. This relatively open channel of communication is easily accessible and prone to external manipulation. By altering data and inserting malicious code, attackers may penetrate and exploit your sever directly through the query string.

Fortunately, we can protect our server against malicious query-string exploits with the help of Apache's invaluable `QUERY_STRING` variable. By taking advantage of this variable, we can ensure the legitimacy and quality of query-string input by screening out and denying access to a known collection of potentially harmful character strings. Here is an example that will keep our query-strings squeaky clean *(code continues on next page)*:

```
<IfModule mod_rewrite.c>
      RewriteCond %{QUERY_STRING} (localhost|loopback|127\.0\.0\.1)               [NC,OR]
      RewriteCond %{QUERY_STRING} (\.|\*|;|<|>|'|"|\)|%0A|%0D|%22|%27|%3C|%3E|%00) [NC,OR]
      RewriteCond %{QUERY_STRING} (md5|benchmark|union|select|insert)             [NC,OR]
```

libwww-perl (also called LWP) is a set of Perl modules that enable scripts to request resources for the Web.

Apache RewriteRule and query string
http://htaccessbook.com/4y

The query-string is the portion of the request that appears after the first question-mark "?", for example:
`http://domain.tld/anything/?q=this-is-the-query-string`

Apache .htaccess query string redirects
http://htaccessbook.com/4z

```
      RewriteCond %{QUERY_STRING} (cast|set|declare|drop|update) [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

As you can see, here we are using the QUERY_STRING variable to check all query-string input against a list of prohibited alphanumeric characters strings. This strategy will deny access to any URL-request that includes a query-string containing localhost references, invalid punctuation, hexadecimal equivalents, and various SQL* commands. Blacklisting these entities protects us from XSS*, remote shell attacks, and SQL injection.

Blacklist via the request

The next blacklisting method is based on the client's request. When a client attempts to connect to the server, it sends a full HTTP request string that specifies the request method, request URI, and transfer-protocol version. Note that additional headers sent by the browser are not included in the request string. Here is a typical example:

```
GET blog/index.html HTTP/1.1
```

This complete request-line* may be checked against a list of prohibited characters to protect against malicious requests and other exploitative behavior. Here is an example of sanitizing client requests by way of Apache's THE_REQUEST variable:

```
<IfModule mod_rewrite.c>
      RewriteCond %{THE_REQUEST} (\\r|\\n|%0A|%0D) [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

Here we are evaluating the entire client-request string against a list of prohibited entities. While there are many character strings common to malicious requests, this example focuses on the prevention of HTTP response-splitting*, XSS attacks*, cache-poisoning*, and similar dual-header exploits. Although these are some of the most common types of attacks, there are many others. I encourage you to check your server logs, do some research, and sanitize accordingly.

Blacklist via request-URI

Use of Apache's REQUEST_URI variable is frequently seen in conjunction with URL canonicalization. The REQUEST_URI variable targets the requested resource specified in the full HTTP request string. Thus, we may use Apache's THE_REQUEST variable to target the entire request string (as discussed above), while using the REQUEST_URI variable to target the actual request URI. For example, the REQUEST_URI variable refers to the "blog/index.html" portion of the following, full HTTP-request line:

```
GET blog/index.html HTTP/1.1
```


SQL = Structured Query Language
http://htaccessbook.com/51

XSS = Cross-Site Scripting
http://htaccessbook.com/50

In this example, "blog/index.html" is the requested URI. In terms of variables, REQUEST_URI is the request-string, and THE_REQUEST is the entire HTTP-request.

```
REQUEST_URI = blog/index.html
THE_REQUEST = GET blog/index.html HTTP/1.1
```

Introduction to HTTP Response Splitting
http://htaccessbook.com/52

Cache Poisoning
http://htaccessbook.com/54

XSS (Cross Site Scripting) Cheat Sheet
http://ha.ckers.org/xss.html

HTTP Cache Poisoning via Host Header Injection
http://htaccessbook.com/53

For canonicalization purposes, this is exactly the type of information that must be focused on and manipulated in order to achieve precise, uniform URLs. Likewise, for blacklisting malicious request activity such as the kind of nonsense usually exposed in your server's access and error logs, targeting, evaluating, and denying malicious URL requests is easily accomplished by taking advantage of Apache's `REQUEST_URI` variable.

As you can imagine, blacklisting via `REQUEST_URI` is an excellent way to eliminate scores of malicious behavior. Here is an example that includes some of the same characters and strings that are blocked in the 5G Blacklist:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_URI} (,|;|:|<|>|”>|”<|/|\\\.\.\\) [NC,OR]
      RewriteCond %{REQUEST_URI} (\=|\@|\[|\]|\^|\`|\{|\}|\~) [NC,OR]
      RewriteCond %{REQUEST_URI} (\’|%0A|%0D|%27|%3C|%3E|%00) [NC]
      RewriteRule .* - [F,L]
</IfModule>
```

Again, same general pattern of directives as before, only this time we are specifying forbidden characters via the `REQUEST_URI` variable. Here we are denying any URL requests containing invalid characters, including different types of brackets, various punctuation, and some key hexadecimal equivalents. The blacklist should be customized according to your specific needs.

HTTP/1.1 Protocol (see section 5.1.2: Request-URI)
http://htaccessbook.com/55

Protect your site with the Perishable Press 5G Blacklist
http://htaccessbook.com/5g

Dealing with blacklisted visitors

In each of these blacklisting techniques, we respond to all blacklisted visitors with the server's default "403 Forbidden" error. It's super-efficient in terms of system-resources, but there is much more that you can do with all of that blacklisted traffic. Here are a few ideas.

Redirect to homepage

More subtle than the 403-error, this redirect strategy routes blocked traffic directly to the home page. To use, replace the `RewriteRule` directive with the following code:

```
RewriteRule .* http://example.com/ [F,L]
```

Redirect to an external site

The possibilities here are endless, but think twice about the destination, as any scum that you redirect to another site will be seen as coming from your own. Even so, here is the code that you would use to replace the `RewriteRule` directive in any of the examples above:

```
RewriteRule .* http://external-domain.tld/some-target/page.html [F,L]
```

Redirect them back to their own site

This is one of my favorites. It's like having a magic shield that reflects attacks back at the attacker. Send a clear message with the following `RewriteRule`:

Protect Your Site with a Blackhole for Bad Bots
http://htaccessbook.com/56

How to build a Bot Trap
http://htaccessbook.com/58

Ajax-Powered Error Logs
http://htaccessbook.com/57

Bot-trap - A Bad Web-Robot Blocker
http://htaccessbook.com/59

```
RewriteRule .* http://%{REMOTE_ADDR}/ [F,L]
```

Custom processing

For those of you with a little skill, it is possible to redirect your unwelcome guests to a fail-safe page that explains the situation to the client while logging all of the information behind the scenes. This is perhaps the most useful approach for understanding your traffic and developing an optimal security strategy. The code would look something like this, depending on your file name and its location:

```
RewriteRule .* /home/path/blacklisting-script.php [F,L]
```

Blacklisting with mod_alias

So far, most of the blacklisting techniques use Apache's powerful rewrite-module. It's true that **mod_rewrite** gives us all flexibility needed to employ strong firewalls and blacklists, there are cases where it's simpler and equally effective to use **mod_alias**' `RedirectMatch` instead. Primarily, `RedirectMatch` directives may be used when targeting the request-URI* for potential blacklisting. And this is often the case, where blocking malicious request-strings is all that's needed to neutralize a wide-range of nefarious activity.

Basic example of blacklisting with RedirectMatch

As seen in section 6.1, `RedirectMatch` gives us a simple and effective way to redirect virtually any request-string. Whereas the similar `Redirect` directive doesn't allow for use of regular-expressions, `RedirectMatch` does precisely that. For example:

```
RedirectMatch 403 (https?|ftp|php)\://
```

This simple directive blocks any request that contains character-strings such as "`http://`", "`ftp://`", "`php://`", and so forth. Such strings are not allowed* in URL-request, so blocking them eliminates hundreds or thousands of ill requests. Thus the utility of `RedirectMatch`: it's a powerful, efficient tool for crafting strong blacklist rules.

The 5G Blacklist

For a more advanced example of using `RedirectMatch` to block bad traffic, allow me to share the **mod_alias** directives used in my own personal firewall, the 5G Blacklist* *(code continues on next page)*.

```
<IfModule mod_alias.c>
	RedirectMatch 403 (https?|ftp|php)\://
	RedirectMatch 403 /(cgi|https?|ima|ucp)/
	RedirectMatch 403 /(Permanent|Better)$
	RedirectMatch 403 (\=\\\'|\=\\%27|/\\\'/?|\)\.css\()$
	RedirectMatch 403 (\,|//|\)\+|/\,/|\{0\}|\(/\(|\.\.\.|\+\+\+|\||\\\"\\\")
	RedirectMatch 403 \.(cgi|asp|aspx|cfg|dll|exe|jsp|mdb|sql|ini|rar)$
	RedirectMatch 403 /(contac|fpw|install|pingserver|register)\.php$
	RedirectMatch 403 (base64|crossdomain|localhost|wwwroot|e107\_)
	RedirectMatch 403 (eval\(|\_vti\_|\(null\)|echo.*kae|config\.xml)
	RedirectMatch 403 \.well\-known/host\-meta
```


See section 6.1 for redirecting with `RedirectMatch`.

Protect your site with the complete 5G Blacklist:
http://htaccessbook.com/5g

```
        RedirectMatch 403 /function\.array\-rand
        RedirectMatch 403 \)\;\$\(this\)\.html\(
        RedirectMatch 403 proc/self/environ
        RedirectMatch 403 msnbot\.htm\)\.\_
        RedirectMatch 403 /ref\.outcontrol
        RedirectMatch 403 com\_cropimage
        RedirectMatch 403 indonesia\.htm
        RedirectMatch 403 \{\$itemURL\}
        RedirectMatch 403 function\(\)
        RedirectMatch 403 labels\.rdf
        RedirectMatch 403 /playing.php
        RedirectMatch 403 muieblackcat
</IfModule>
```

Here we see `RedirectMatch` used to create a strong firewall to protect your website. There's a lot happening in this code, so I'll refer you to the original post and it's companion articles that explain the process of blacklisting in-depth.

For now, consider this technique an example of what's possible with a small block of .htaccess directives. As-is, this code eliminates a *wide range* of malicious requests, preventing any nonsense before it has a chance to cause harm. It's not foolproof, but it's widely tested and used on all sorts of awesome sites around the Web.

Building the 5G Blacklist
http://htaccessbook.com/5c

Building the 4G Blacklist
http://htaccessbook.com/5b

Building the 3G Blacklist
http://htaccessbook.com/5d

Want to help test the 6G Blacklist? Here it is, 6G Beta:
http://htaccessbook.com/6g

chapter 8

enhance usability

The usability of your site is determined by a number of factors, but is determined largely by the User-Interface (UI), which is generally thought of as happening on the client-side. In this chapter, we'll take a fun look at how .htaccess can facilitate a better User-Experience (UX) at the server-level.

Admittedly, there are not a million usability improvements you can make with .htaccess, but there's enough to warrant a good chapter on the topic. In this chapter, I'm looking at usability in the broadest sense, insofar as the user-experience begins when the visitor requests a page from your server. When all is going well, visitors are looking at your deftly styled pages displaying your best content. But things go wrong, and visitors may end up staring at a 404-page. Or maybe visiting a broken page during a site-update. The techniques in this chapter are aimed at optimizing these types of scenarios to maximize the usability of your site.

God have mercy.
http://www.usability.gov/

UX Design @ UXmatters
http://htaccessbook.com/5h

UX Design at Smashing Magazine
http://htaccessbook.com/5f

User-Experience and UX Design
http://uxdesign.com/

8.1 Serve custom error pages

We all know how important it is to deliver sensible, helpful 404-error pages to our visitors. There are many ways of achieving this functionality, including the well-known .htaccess trick used to locally redirect users to custom error-pages. Using the `ErrorDocument` directive, we can serve custom pages for the most-common types of errors.

To implement this technique, edit each line with the path and file name of your custom error-document.

```
ErrorDocument 400 /errors/400.html
ErrorDocument 401 /errors/401.html
ErrorDocument 403 /errors/403.html
ErrorDocument 404 /errors/404.html
ErrorDocument 500 /errors/500.html
```

These directives basically tell Apache to deliver the designated documents for their associated error-types. Many webmasters and developers employ this trick to ensure that visitors receive customized error pages that are generally more user-friendly or design-specific than the rather unfriendly Apache defaults. Serving custom error-pages is an excellent way to enhance overall site usability and accessibility, but there are several alternate methods and ways to customize the technique that are worth checking out.

Note: your custom error-pages must be larger than 512 bytes in size or they will be completely ignored by Internet Explorer: http://htaccessbook.com/5i

Custom HTTP Errors via htaccess
http://htaccessbook.com/5j

Change the default error message

You don't need to design a complete set of customized error-documents just to get your point across. Simply changing the default error message to something more useful is a great way to spice things up with minimal fuss. Instead of pointing to a file, we can deliver any plain-text message wrapped in quotes, like so:

```
ErrorDocument 400 "Ooops - Bad request!
ErrorDocument 401 "Speak friend and enter
ErrorDocument 403 "Strictly fabidden..
ErrorDocument 404 "Missing in action..
ErrorDocument 500 "Server gone wild..
```

You can change the custom message to anything you want, but it must be plain text. This technique is ideal for hardcore sites with an audience that is a bit more "error-savvy" than the typical visitor. Note that the initial quotation mark (") specifies the character string as text and is not included in the message itself. No closing quotation mark is required.

Redirect errors to a custom script

Moving in the other direction, you may wish to incorporate some dynamic functionality into your error messages. For example, if you had a script that customized the error message by employing, say, referrer, client, and query data, you could redirect your error pages like so:

About the ErrorDocument Directive
http://htaccessbook.com/3j


```
ErrorDocument 400 /errors/redirection.php
ErrorDocument 401 /errors/authentication.php
ErrorDocument 403 /errors/explanation.php
ErrorDocument 404 /cgi-bin/not_found.pl
ErrorDocument 500 /cgi-bin/server_error.pl
```

Redirect to an external URL

Returning to the most common error-page customization-method, it should be noted that the custom error pages do not need to be located on the same domain. In fact, you can redirect your HTTP-errors to virtually any URL:

```
ErrorDocument 400 http://example.com/400.html
ErrorDocument 401 http://example.com/401.html
 :
 :
```

Keep in mind that any environmental variables or other request-specific information will not survive the external redirect. One last reminder about serving custom error pages: make sure that they are greater than **512 bytes** in size or Internet Explorer will ignore them.

Provide a universal error-page

One last technique for serving custom error-pages is to serve the same custom page regardless of the type of error. With the right scripting in place, consistent, informative

404 Error Pages: Reloaded
http://htaccessbook.com/5k

error-pages can be a great way to improve the user-experience of your site. Here is one way to do this using Apache's mod_rewrite:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_FILENAME} !-f
      RewriteCond %{REQUEST_FILENAME} !-d
      RewriteRule .* /path/errors.php [L]
</IfModule>
```

8.2 Serve browser-specific content

In the chapter on redirecting stuff, we learn one way of serving browser-specific pages using mod_rewrite and the HTTP_USER_AGENT variable. Sniffing the user-agent string and delivering custom content is a useful way to handle difficult design scenarios. Internet Explorer is a ripe example, as designs almost invariably look different than in other browsers. In this section, we're taking browser-sniffing beyond the whole-page and into the design itself, enabling us to highly customize the user-experience virtually anywhere.

Detecting the user-agent with .htaccess

There are many techniques available on the Web for detecting the user-agent. Here is one of the best methods* that I have seen for doing it with .htaccess, whereby the user-agent is stored as an environmental variable that may be used when scripting your web-pages:

No more CSS hacks: Browser sniffing with .htaccess
http://htaccessbook.com/3f


```
<IfModule mod_setenvif.c>
      SetEnvIfNoCase User-Agent (Opera|Chrome|Version|Firefox|MSIE)[\/|\s](\d+)\.
      \browser=$1 version=$2
      SetEnvIf browser Version browser=Safari
      SetEnvIf browser MSIE browser=IE
</IfModule>
```

Using Apache's mod_setenvif*, this technique checks for the five major browsers* and stores their user-agent and version-number as variables on the server. As you can imagine, being able to access that information from anywhere on the page brings powerful customization options. Let's take a quick look at how to do this with PHP.

Serving customized content with PHP

Once the user-agent and version-number are available to you from within the web-page, you can verify their existence with a bit of PHP and HTML:

```
<ul>
      <li>User-agent: <?php echo $_ENV['browser']; ?></li>
      <li>UA version: <?php echo $_ENV['version']; ?></li>
</ul>
```

Then once you see how this works, you're ready to customize based on user-agent and/or version-number. For more information on this technique, visit the original article*.

Apache Module mod_setenvif
http://htaccessbook.com/5l

Browser Wars 2011
http://htaccessbook.com/5m

No more CSS hacks: Browser sniffing with .htaccess
http://htaccessbook.com/3f

8.3 Improving directory-views

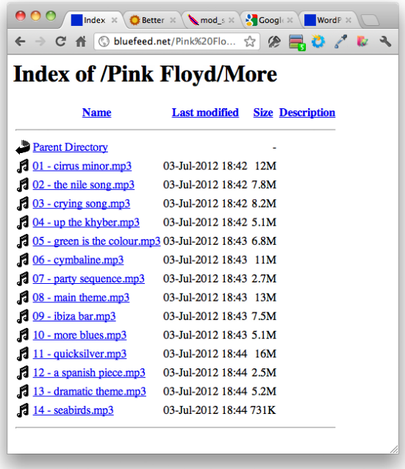
Displaying "index-less" directory-views is a great way to share files, but the drab, bare-bones interface is difficult to integrate into existing designs. While there are many scripts available to customize the appearance and functionality of default directory navigation, most of these methods are too complicated, too invasive, or otherwise insufficient for expedient directory styling.

In this section, you'll learn how to use the built-in functionality of Apache's **mod_autoindex** to style and enhance your default directory views with a smorgasbord of stylistic and functional improvements.

Before diving in...

Default directory-views are very common on the Web. Any directory that does not contain a default index file, such as `index.html`, `index.php`, `default.asp`, or something similar, may present its contents via default directory-view.

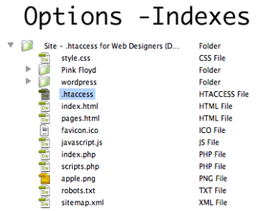
As discussed, the best strategy for directory-views is to disable them site-wide, and then enable *locally* in specific directories. See the screenshot for a visual of this concept.

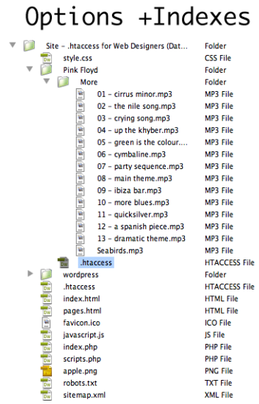


Shown here is the default directory-view, generated by Apache when Indexes are enabled.

Basic customization

Apache's `IndexOptions` directive provides many useful options for configuring and customizing just about everything. Rather than explain them all here, let's start with the basics and move on from there. Here's a list of customization techniques that we want to achieve:





- **Fancy indexing** — display additional file info, including file icons, descriptions, etc.
- **Folders first** — always display any subdirectories before individual files
- **Auto-width** — we want the name-column and description-column to automatically resize according to the length of the longest file/folder name
- **Suppress the HTML preamble** — by default, Apache automatically includes the opening HTML elements (e.g., `<html>`, `<head>`) for the markup comprising the directory listing; when using a custom header and footer file, we want to disable the default markup.

Better Default Directory Views with HTAccess
http://htaccessbook.com/5n

See section 7.1 for information about how and why to disable directory-views via .htaccess.

For a complete list, check out the official mod_autoindex manual at the venerable Apache website.
http://htaccessbook.com/50

In the screenshot, we disable directory-views via the root .htaccess file to protect the site. Then to enable directory-views for our download-folder, we add "`Options +Indexes`" in the local .htaccess file for that directory.

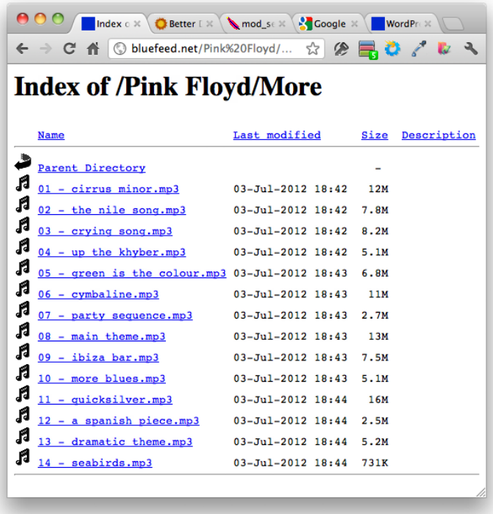
Amazingly enough, all of this functionality is invoked with a single directive:

```
IndexOptions IgnoreCase FancyIndexing FoldersFirst NameWidth=* DescriptionWidth=*
SuppressHTMLPreamble
```

Notice the two options for specifying the two column-widths: `NameWidth` and `DescriptionWidth`. Note also the use of the wildcard-operator (*) for the keen auto-width columns. Of course, any value (in pixels) may be specified here, depending on your specific needs. And, while we're at it, let's specify the default directory display-order:

```
IndexOrderDefault Descending Name
```

After placing these directives within the previously specified `<IfModule>` container, our "Pink Floyd" directory view looks pretty much the same, as seen in this screenshot. As you can see, not much has changed in terms of appearance, but don't worry, we'll get to that next...



See section 2.7 for .htaccess character definitions.

IndexOrderDefault Directive
http://htaccessbook.com/5p

Top Ten Pink Floyd Songs for Audiophiles
http://htaccessbook.com/5q

Customizing markup

The key to customizing the look and feel of the default directory listing is the ability to modify and customize the `<head>` region of the document markup. Even though they look plain and boring, default directory views are displayed via good 'ol fashioned HTML. Sadly formatted HTML perhaps, but HTML nonetheless. Each directory view consists of three different "chunks" of HTML:

- **The Header** — by default automatically generated by Apache
- **The Directory Listing** — necessarily generated by Apache
- **The Footer** — referred to as the "Readme" file

Fortunately, the Apache Wizards have provided a way to override the default header and footer files, thereby enabling us to include our own elements within the document `<head>`, like CSS styles, JavaScript, or anything else we desire. Likewise with the footer, we may add any sort of custom content, links, information, notes, or whatever. To specify custom header and footer files, we add this to the .htaccess file contained in our custom directory:

```
# SPECIFY HEADER FILE
HeaderName header.html

# SPECIFY FOOTER FILE
ReadmeName footer.html
```

As seen in section 7.1, we can prevent these HTML files from being displayed along with other files, for example:

```
IndexIgnore header.html footer.html
```

Each of these two files, `header.html` and `footer.html`, must be created and properly located according to the specified path. In this case, we are simply placing the files in the same directory as the .htaccess file, so no other path information is required. These files may be placed anywhere on the server, however, so long as they are accessible via correct file paths. Given our preconfigured options, here is the minimum amount of markup that must be included within the custom header file:

```
<html>
	<head>
		<title>Pink Floyd - More</title>
	</head>
	<body>
		<h1>Pink Floyd - More</h1>
```

Technically, we only need the opening `<html>` and `<body>` elements, but it is nice to include a title for the document as well. Before styling our document, let's have a quick look at the custom footer file we will be using:

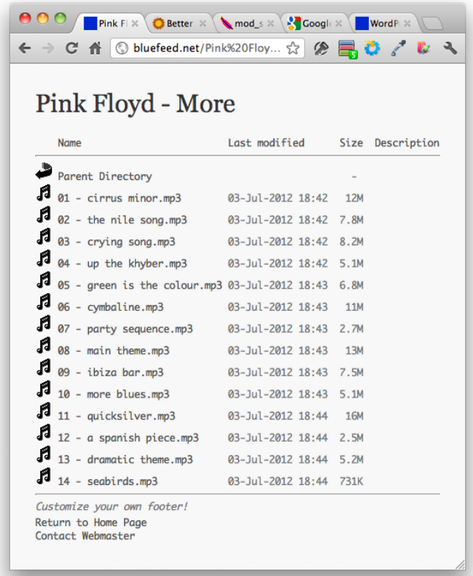
```
		<pre><em>Customize your own footer!</em></pre>
		<pre><a href="http://example.com/">Return to Home Page</a>
		<a href="http://example.com/contact/">Contact Webmaster</a></pre>
	</body>
</html>
```


Basic HTML Coding (cheat-sheet)
http://htaccessbook.com/5r

HTML5Rocks
http://www.html5rocks.com/

Note: remove the tab-indents from before the `<pre>` tags to get their contents to align left (as shown in the screenshot).

With the custom header and footer in place, our directory-view is looking better, but we can improve it further with a little CSS. To do so, we simply add our desired CSS styles to the `<head>` region. Before applying CSS styles, however, it helps to know which HTML elements we have at our disposal. With `FancyIndexing` enabled, directory listing markup includes the following elements: `<body>`, `<img>`, `<pre>`, `<h1>`, `<hr>`, `<a>`.



Customizing with CSS

To finish our customization, let's add the following CSS to the `<head>` section of our `header.html` file:

```
<style type="text/css">
body { background: #f8f8f8; color: #333; margin: 30px; }
h1 { font: 30px Georgia, serif; margin: 12px 0; }
a:link, a:visited { color: #555; text-decoration: none; }
a:hover, a:active { color: #cc0000; text-decoration: underline;
} pre { color: #777; font: 12px Monaco, monospace; margin: 3px
0; }
</style>
```

That will get you the view shown in the screenshot, and of course much more is possible if you use your imagination and enjoy the process.

CSS Basics
http://www.cssbasics.com/

Fully customizable CSS3 generator
http://www.css3.me/

As shown in the screenshot, the design for this example is pretty basic, and that makes sense because it is, after all, a directory listing. More elaborate designs should probably be given their own web-page, where it's going to be a lot easier to achieve complex design goals.

8.4 More usability enhancements

In addition to the previous techniques, here are some further .htaccess tricks for a wide range of usability enhancements. From spell-checking the requested URL to customizing Internet Explorer, these UX-techniques prove handy enough to warrant their own section.

Basic spell-checking for requested URLs

As explained in section 3.8, this code will auto-correct simple spelling errors in the URL:

```
<IfModule mod_speling.c>
      CheckSpelling On
</IfModule>
```

That only covers basic errors, but is useful to help even sloppy visitors get what they need.

Display source-code for dynamic files

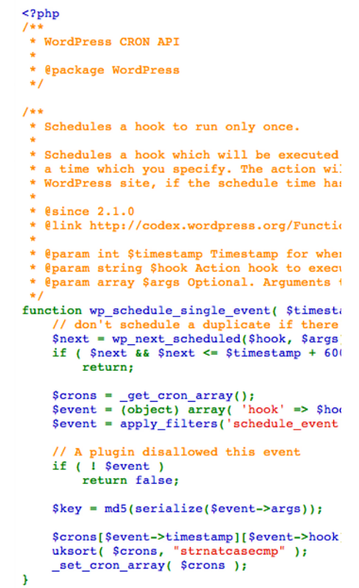
Instead of executing PHP, Python, and other dynamic file-types, it may be useful to display the source-code for public use. The easiest way to do this with PHP-files is to rename them to something like, "`filename.source.phps`". Files ending with "`.phps`" will be displayed as syntax-highlighted source-code. And if your file-names contain a common character-string, you don't have to rename anything. For example, if we have a series of files named like this:

- `filename-01-source.php`
- `filename-02-source.php`
- `filename-03-source.php`

We could display the source-code for these files by adding the following .htaccess file to the root .htaccess file:

```
<Files ~ "(.*)-source\.php">
      RemoveHandler .php .phtml .php3
      RemoveType .php .phtml .php3
</Files>
```

The key to this technique is to remove both the `Handler` and `Type`, which should work for just about any type of file. It's important to match only those files for which you would like to display contents. As mentioned, this code only works when placed in the root .htaccess file, so proper file-matching is critical to keep things secure.



```
<?php
/**
 * WordPress CRON API
 *
 * @package WordPress
 */

/**
 * Schedules a hook to run only once.
 *
 * Schedules a hook which will be executed
 * a time which you specify. The action wi
 * WordPress site, if the schedule time ha
 *
 * @since 2.1.0
 * @link http://codex.wordpress.org/Functio
 *
 * @param int $timestamp Timestamp for whe
 * @param string $hook Action hook to exec
 * @param array $args Optional. Arguments
 */
function wp_schedule_single_event( $timest
    // don't schedule a duplicate if there
    $next = wp_next_scheduled($hook, $args
    if ( $next && $next <= $timestamp + 60
        return;

    $crons = _get_cron_array();
    $event = (object) array( 'hook' => $ho
    $event = apply_filters('schedule_event

    // A plugin disallowed this event
    if ( ! $event )
        return false;

    $key = md5(serialize($event->args));

    $crons[$event->timestamp][$event->hook
    uksort( $crons, "strnatcasecmp" );
    _set_cron_array( $crons );
}
```

Force download of specific file-types

Here is a useful method for delivering multimedia file downloads to your users. Typically, browsers will attempt to play or stream such files when direct links are clicked. With this method, provide a link to a multimedia file and a

Apache Module mod_speling
http://htaccessbook.com/12

5 Easy Ways to Display Syntax Highlighted PHP Code
http://htaccessbook.com/55

http://htaccessbook.com/76
http://htaccessbook.com/77

This particular technique requires that the code be placed in the root .htaccess file. Of course, httpd.conf also works, so `<Location>` and `<Directory>` directives would provide more control over which directories are affected.

Screenshot shows a PHP script as displayed via this technique. Many scripts are made available by simply displaying its source-code.

dialogue box will provide users the choice of saving the file or opening it. Here are a few htaccess rules demonstrating the technique (edit extensions to match your specific files):

```
<IfModule mod_mime.c>
      AddType application/octet-stream .avi
      AddType application/octet-stream .mpg
      AddType application/octet-stream .wmv
      AddType application/octet-stream .mp3
</IfModule>
```

Block access during at specific times

If for some reason you need to block or redirect traffic during a specific time of day, you can use a wide variety of `TIME_XXX` variables available for rewrite-conditions. As stated in the Apache Documentation, "in conjunction with the special lexicographic comparison patterns "`<STRING`", "`>STRING`" and "`=STRING`" we can do time-dependent redirects." Here is an example that prevents access to any file during the midnight hour.

```
<IfModule mod_rewrite.c>
      RewriteCond %{TIME_HOUR} ^12$
      RewriteRule .* - [F,L]
</IfModule>
```

Similarly, we can redirect all requests for our site's RSS feed between 1:00-2:00am:

The article "Serve Alternate Content based on Time" contains all the gory details: http://htaccessbook.com/5t

Time-Dependent Rewriting
http://htaccessbook.com/5v


```
<IfModule mod_rewrite.c>
      RewriteCond %{TIME_HOUR}%{TIME_MIN} >0100
      RewriteCond %{TIME_HOUR}%{TIME_MIN} <0200
      RewriteRule ^feed.xml /update.html [F,L]
</IfModule>
```

Quick IE tips

Rounding out our chapter on usability, here are two quick tips for Internet Explorer.

Remove the IE imagetoolbar

```
<FilesMatch "\.(html|htm)$">
      <IfModule mod_headers.c>
            Header set imagetoolbar "no"
      </IfModule>
</FilesMatch>
```

Minimize CSS image-flicker in IE6

```
<IfModule mod_expires.c>
      ExpiresActive On
      ExpiresByType image/gif A2592000
      ExpiresByType image/jpg A2592000
      ExpiresByType image/png A2592000
</IfModule>
```

More ways to disable the IE Image Toolbar:
http://htaccessbook.com/5w

More about the IE image-flicker :
http://htaccessbook.com/5x

chapter 9

.htaccess tricks for WordPress

Many aspects of WordPress' default functionality may be enhanced or optimized with .htaccess. You can make your site's URLs look clean and simple, redirect your RSS feeds to services such as FeedBurner, and even improve the security of your site.

Out of the box, WordPress doesn't include any .htaccess files. It works just fine without them too. But if you want to optimize the SEO of your site with better-looking URLs, or want to reduce the volume of comment-spam, a few .htaccess directives are all that's needed to make it happen.

So chapter 9 is all about WordPress tricks and techniques. One could actually fill an entire book with .htaccess methods for WP, but in my experience most of the best techniques fit neatly into a single, tightly written chapter.

Self-hosted WordPress:
http://wordpress.org/

WP as a hosted-service:
http://wordpress.com/

Digging into WordPress
http://digwp.com/

Smashing WordPress
http://wp.smashingmagazine.com/

9.1 Optimizing WordPress Permalinks

By default, WordPress generates URLs that look like this:

```
http://example.com/?p=219
```

Although short URLs are all the rage these days, WordPress' default, query-string URLs aren't as stylish as the permalink-based URLs:

```
http://example.com/killa/
```

WordPress makes it easy to use the permalink-format for your URLs. Just visit **Settings > Permalinks** in the Admin, and then choose your permalink-format. The final step is to add the required .htaccess code.

 # WordPress installed in root directory

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteBase /
 RewriteRule ^index\.php$ - [L]
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteCond %{REQUEST_FILENAME} !-d
 RewriteRule . /index.php [L]
</IfModule>
```

 # WordPress installed in subdirectory

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteBase /wordpress/
 RewriteRule ^index\.php$ - [L]
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteCond %{REQUEST_FILENAME} !-d
 RewriteRule . /wordpress/index.php [L]
</IfModule>
```

Permalinks for non-WP sites? Check out "Making "clean" URLs with Apache and PHP" http://htaccessbook.com/5y

WP Codex: Using Permalinks
http://htaccessbook.com/6d

Note: if you're using the sub-directory code, edit both instances of "wordpress" to that of your sub-directory.

The htaccess Rules for all WordPress Permalinks
http://htaccessbook.com/f

Canonical permalinks with www or non-www

As discussed in section 5.1, it's smart for SEO to canonicalize your site's URLs either with or without the www-prefix. There are pros and cons, but a recent poll shows most people prefer the non-www versions for a variety of reasons. So, to serve only non-www versions of your URLs, insert this code before the first `RewriteCond` in WordPress' permalink rules:

```
RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
RewriteRule ^(.*)$ http://%1/$1 [R=301,L]
```

Or, if you're too powerful and need to serve only the www-versions of your URLs, use this code instead, also placed before the first `RewriteCond` in WordPress' permalink rules:

```
RewriteCond %{HTTP_HOST} !^www\. [NC]
RewriteCond %{HTTP_HOST} ^(.*)$  [NC]
RewriteRule ^(.*)$ http://www.%1$1 [R=301,L]
```

No editing is required for either of these add-ons. Just include them as-is into your permalink rules and you're all set.

Cleaning-up dead-end permalinks

In creating its permalink structure, WordPress leaves a few dead-end URLs that should be cleaned-up. Specifically, the following URLs generate a 404 "Not Found" error:

Poll: www. or no www.?
http://htaccessbook.com/2p

htaccess Combo Pack: WordPress Permalinks and non-www Redirect: http://htaccessbook.com/2t

Reminder to backup files before making changes :)


```
http://example.com/tag/
http://example.com/search/
http://example.com/category/
```

To fix this, we add the following three **mod_alias** directives to .htaccess:

```
RedirectMatch 301 ^/tag/$ http://example.com/
RedirectMatch 301 ^/search/$ http://example.com/
RedirectMatch 301 ^/category/$ http://example.com/
```

Remember to edit the "`example.com`" with your own domain-name or preferred URL.

Optimize date-based permalinks

Shorter URLs are thought to be optimal for SEO. Including the "`/year/month/day/`" is a good idea for date-based archive-pages, but not for your single posts and pages. The easiest way to remove the dates from your URLs is to visit **Settings > Permalinks** and then:

- **Change this**: `/%year%/%monthnum%/%day%/%postname%/`
- **To this**: `/%postname%/`

WordPress is smart enough to redirect all of your *old* URLs to the new ones, but if your site's been online for a while, you'll have to redirect any *incoming links* that you want to

Note that `<IfModule>` containers should be used whenever possible. They are omitted in some of the examples to save space. See section 2.8 for more info.

Optimizing WordPress Permalinks
http://htaccessbook.com/5z

Optimize Your Dated WordPress Permalinks
http://htaccessbook.com/60

keep. If you do *nothing* after changing the format of your permalinks, all external links will generate 404-errors, and all of that precious link-equity will be wasted. To avoid this, use the following .htaccess to redirect all of those old links to your shiny new ones[*]:

```
RedirectMatch 301 ^/([0-9]+)/([0-9]+)/([0-9]+)/(.*)$ http://example.com/$1/$4
```

This will redirect all requests for "`year/month/day`" URLs:

```
http://example.com/2007/01/04/example-post/
http://example.com/2008/02/05/another-post/
http://example.com/2009/03/06/amazing-post/
```

..to the new and improved permalink-format, which for this technique includes only the year:

```
http://example.com/2007/example-post/
http://example.com/2008/another-post/
http://example.com/2009/amazing-post/
```

Of course, other redirects are also possible, based on how your old and new permalinks are formatted. The basic technique is the same, but the regular-expression may vary depending on what you're trying to do. Here are some quick examples to get you going.

Here, each of the "`([0-9]+)`" regexes match any number, however, in order for the redirect to occur, the entire pattern must match, so for example:

`http://example.com/anynumber/anynumber/anynumber/anycharacters`

This is great for matching WP-URLs, but if you're using other PHP-scripts (e.g., e-commerce or social-media), check that they don't use similarly formatted URLs.

This is just a heads up — it should be fine 99% of the time, and there are ways to customize if necessary.

Redirect year/month/day permalinks to post-name only

```
RedirectMatch 301 ^/([0-9]+)/([0-9]+)/([0-9]+)/(.*)$ http://example.com/$4
```

Redirect year/month permalinks to year/post-name only

```
RedirectMatch 301 ^/([0-9]+)/([0-9]+)/(.*)$ http://example.com/$1/$3
```

Redirect year/month permalinks to post-name only

```
RedirectMatch 301 ^/([0-9]+)/([0-9]+)/(.*)$ http://example.com/$3
```

Remember, when using any of these techniques to change the "`example.com`" to your domain-name. And of course, test thoroughly.

Redirecting WordPress Date Archives

This technique is useful for redirecting your date-based archive-pages after removing the installation-subdirectory from your URLs[*]. For example, if your old date-archives looked like this:

```
http://example.com/wordpress/2007/
http://example.com/wordpress/2008/02/
http://example.com/wordpress/2009/03/06/
```

..and you've since removed the "`/wordpress/`" directory[*], you can use the following .htaccess

Many users move the WP-installation from the root directory to a subdirectory. This is optimal for a number of reasons, such as enabling other PHP-scripts or apps in the root directory, simplifying the URL-format, and streamlining the directory structure.

When WP is installed in a subdirectory, it's possible to configure your URLs to exclude the name of the subdirectory from the URL. http://htaccessbook.com/61

WP Codex: Changing The Site URL
http://htaccessbook.com/6e

snippet to redirect to the new date-archive URLs:

```
http://example.com/2007/
http://example.com/2008/02/
http://example.com/2009/03/06/
```

Here is the code, just replace the "example.com" with your domain-name:

```
<IfModule mod_alias.c>
RedirectMatch 301 ^/wordpress/([0-9]+)/?$         http://example.com/$1/
RedirectMatch 301 ^/wordpress/([0-9]+)/page/(.*)$ http://example.com/$1/page/$2

RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)/?$         http://example.com/$1/$2/
RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)/page/(.*)$ http://example.com/$1/$2/page/$3

RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)/([0-9]+)/?$         http://example.com/$1/$2/$3/
RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)/([0-9]+)/page/(.*)$ http://example.com/$1/$2/$3/page/$4
</IfModule>
```

There are three sections to this code: *yearly archives*, *monthly archives*, and *daily archives*. The only editing that you need to do is change the "wordpress" in each line to match your sub-directory's name, and then also replace all instances of "example.com" with your domain-name. Remember to test thoroughly and keep an eye on your access logs after implementing these rules. Incidentally, this technique is working great at my web-development site, Perishable Press.

When you configure WordPress to serve your site from the root-directory (e.g., at http://example.com/), it automatically excludes the name of any installation directory from permalink-URLs.

So for new WP-installs with no incoming links, there is no need to implement this technique. It's only needed for sites that want to keep incoming link equity from external sites after moving or re-configuring things.

Optimizing WordPress Permalinks with htaccess
http://htaccessbook.com/62

Eliminate all date-based archives

You may have asked at some point, "why bother at all with date-based archives?" After all, it's redundant info that may contribute to "duplicate-content" penalties with the search-engines. Many WP-themes tend to ignore date-based archives entirely, and popular SEO-plugins provide easy ways of no-indexing/removing them from search engines. So why not just eliminate date-archives entirely, say by redirecting all of it to your home page, where the additional link equity may provide some nice SEO-benefits. Two steps to do it:

Step 1. Add code to the root .htaccess file

```
<IfModule mod_alias.c>
 RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)?/?([0-9]+)?/?$    http://example.com/
 RedirectMatch 301 ^/wordpress/([0-9]+)/([0-9]+)?/?([0-9]+)?/?page http://example.com/
</IfModule>
```

As discussed in the previous section, edit or remove the subdomain to match your own, and then change the "example.com" to whatever makes sense (e.g., your home page). Test thoroughly using as many different archive/permalink formats as possible.

Step 2. Clean-up all instances of date-archive URLs

Once the date archives have been redirected via .htaccess, you should go through your theme and remove/edit any date-based archive links and/or code. A typical location for

Although it may sound like a good idea, make sure you take the time to examine your access-logs and statistical data to get an idea if anyone is actually visiting your date-based archives. For new sites, however, there is no traffic going to these pages, so this a great technique to keep link-equity focused on your main content.

such would be `archive.php`, `archives.php`, `index.php`, and of course `date.php` if it actually exists. Once your theme is cleaned up, you're all set, but should keep an eye on any weird activity in your traffic logs. As straightforward as it seems, it's a big move, so be smart.

Redirect any removed or missing pages

Whenever you delete a post or page on your site, the search-engines panic, and seem to request those pages over and over and over again. That's pretty wasteful, so let's redirect missing pages to someplace useful, like the homepage of the site. This is simple to do with a few lines of .htaccess, as seen here for a handful of removed pages:

```
<IfModule mod_alias.c>
      RedirectMatch 301 /pancake/?$ http://example.com/
      RedirectMatch 301 /bananas/?$ http://example.com/
      RedirectMatch 301 /oatmeal/?$ http://example.com/
</IfModule>
```

These redirects share a common pattern, so we can optimize our code by rewriting this in single-line format:

```
RedirectMatch 301 /(pancake|bananas|oatmeal)/?$ http://example.com/
```

Notice that we're redirecting these page requests to the site's home page, which you may edit to be whatever URL you wish. This is good for SEO as it preserves any page rank that the removed pages may have accumulated. You can funnel that love wherever you would like, such as a sales page or other key resource.

Make dead pages go away

For pages that you would rather not redirect, but rather just declare them as officially "gone", well there's a code for that. As explained by Mark Pilgrim, you can return a 410 "Gone" status-code to all requests for pages that you'd rather forget about. For example:

```
RedirectMatch gone /nu\-
RedirectMatch gone /nuer
RedirectMatch gone /renu
RedirectMatch gone /nuwest
RedirectMatch gone /nustyle
```

As before, we can take advantage of **mod_alias**' pattern-matching skillz to combine rules into a single line, like so:

```
RedirectMatch gone /(nuer|renu|nuwest|nustyle)
```

So with this technique, we're telling search engines that these resources are literally "gone". This is an *effective* way to eliminate pages from the search engines, so be careful.

Tip: this technique is easily modified to redirect, say, `/pancake/splural/` to `http://example.com/splural/`, effectively removing "`/pancake/`" from the URL.

```
RedirectMatch 301 /pancake/(.*) http://example.com/$1
```

At his late, great site, `diveintomark.org`.

In this technique, "`410`" may be used in place of "`gone`", which may be useful to keep things organized in the file.

A note about the pattern-matching — it's a little bit different than in our previous example. By omitting the regex "`/?$`", at the end of the pattern, this technique will match any URL that begins with "`/nustyle`", such as these examples:

```
/nustyle-1
/nustyle-1-2-3
/nustylebananaspiders
```

Admittedly the term "`nustyle`" is rare enough to avoid unwanted redirects, but more common terms require some fiddling with the pattern-matching. Remember to test.

Redirect entire category to another site

On a recent project, I needed to redirect a specific category to an external site. To do this, I created a new "monkey" category at the external site, and re-published all posts from the old site's monkey-category at the new site. After replicating everything, redirecting the entire category required but a single line of .htaccess:

```
RedirectMatch 301 ^/category/monkeys/?(.*)$ http://example.com/category/monkeys/$1
```

That's about as clean as it gets. The only trick when redirecting category (and other) URLs to other WordPress sites is getting the post permalinks to match exactly. It takes some time, especially if you tend to punctuate post titles with apostrophes and such.

Where weird, random images are shared online:
http://echunks.com/

Smarter Slugs ~!@$%^&*()={}<>[]?
http://htaccessbook.com/63

9.2 WordPress MultiSite

WordPress' MultiSite-mode is a great way to hosts multiple sites with a single installation of WordPress — one set of files, one database, and you can host as many sites as you want. Setting-up and configuring MultiSite can be a chore, but certainly worth it, in my opinion. As you'll discover there are numerous steps that must be followed, including at some point, adding the proper .htaccess code for MultiSite, which is this:

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]

# uploaded files
RewriteRule ^([_0-9a-zA-Z-]+/)?files/(.+) wp-includes/ms-files.php?file=$2 [L]

# add a trailing slash to /wp-admin
RewriteRule ^([_0-9a-zA-Z-]+/)?wp-admin$ $1wp-admin/ [R=301,L]

RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]
RewriteRule  ^([_0-9a-zA-Z-]+/)?(wp-(content|admin|includes).*) $2 [L]
RewriteRule  ^([_0-9a-zA-Z-]+/)?(.*\.php)$ $2 [L]
RewriteRule . index.php [L]
```

After MultiSite is set-up and working properly, this code is also available to you on the **Tools > Network** options page. Of course, things don't always go according to plan, hence

Whenever possible, `<IfModule>` directives should be used unless you're sure that the modules are available. Some examples in this section exclude them to save space.

Official WP-guide to setting up MultiSite
http://htaccessbook.com/65

htaccess Code for WordPress Multisite
http://htaccessbook.com/64

the reason for including this code. Note that these directives should be placed in your primary site's root .htaccess file. There it will do the rewrite-work for your entire network of sites. No editing of the code itself is necessary — just plug-n-play.

WordPress MultiSite Subdomains on MAMP

Developing WordPress sites on a local MAMP server provides flexibility, privacy, and security throughout the development process. Setting up a WordPress environment on MAMP is definitely worth the effort, especially if you're building and testing multiple sites using WordPress' built-in MultiSite functionality.

The easiest and recommended way of setting up MultiSite is to use sub-directories. So when you create a new site named "business", it will be located at "`http://localhost/business/`". Here's a mini-tutorial on how to use sub-domains for your network.

Step 1. Edit the Mac hosts file

After installing MAMP, change the default Apache port to 80. Next, add your sub-domains to the Mac hosts file. To do this, open Terminal and type "**sudo pico /etc/hosts**" (without the quotes), and then enter your password at the prompt. Use the arrow keys to scroll down to the end of the hosts file and add the following lines:

```
127.0.0.1       example.com
127.0.0.1       site1.example.com
```



See section 2.9 for more about "_AMP" and .htaccess.


Developing WordPress Locally With MAMP
http://htaccessbook.com/7h


How to edit the hosts file in Mac OS X
http://htaccessbook.com/66


```
127.0.0.1       site2.example.com
```

Edit these entries to match the domain and sub-domains that you want to create with MultiSite. Add as many sub-domains as needed, now or later. Then save the file and exit by typing Ctrl+O, Enter, and then Ctrl+X.

Step 2. Edit the Apache config file

The next step is to add virtual hosts to your Apache configuration file. Open the file, "`/Applications/MAMP/conf/apache/httpd.conf`", in a text editor and scroll down to the line that says "`#NameVirtualHost *`". Replace that line with the following code:

```
NameVirtualHost *
<virtualHost *>
      ServerName example.com
      ServerAlias example.com *.example.com
      DocumentRoot "/Applications/MAMP/htdocs/"
      <directory "/Applications/MAMP/htdocs/">
            Options Indexes FollowSymLinks Includes
            AllowOverride All
            Order allow,deny
            Allow from all
      </directory>
</virtualHost>
```


Apache Docs: Working with Configuration Files
http://htaccessbook.com/e

Change each instance of `example.com` to match your domain, save the file, and then Restart Apache by clicking "Stop" and then "Start" in the MAMP control panel.

Step 3. Intall & configure WordPress

Now to install WordPress by placing the WordPress installation files in your "`/htdocs/`" directory, create the database via phpMyAdmin (@ `http://localhost/MAMP/`), and edit `wp-config.php` with your database credentials. Then complete the installation process by accessing "`http://example.com/wp-admin/install.php`" in your browser.

Next, enable MultiSite by adding the following line to your `wp-config.php` file, just above the line that says, "`That's all, stop editing! Happy blogging`":

```
define('WP_ALLOW_MULTISITE', true);
```

With that in place, return to the WP Admin and click on **Tools > Network**. On this page you will now see an option to use sub-domains for your site addresses. Make sure that's selected, check the other details, and then click the "Install" button to make it happen. Note that you'll see a warning message that says, "Wildcard DNS may not be configured correctly!" — we can ignore this warning because we know our DNS is correct.

Finally, complete the steps outlined there on the "Enabling the Network" page (i.e., create a "`blogs.dir`" folder and add the required code snippets). After that, re-login to the Admin area and go to **Network Admin > Sites > Add New** to begin adding your sub-domain network sites. And that's it — you're now rolling tuff with WordPress subdomains on a local MAMP development server. No go forth and develop your network.

Official WP-guide to installing WordPress:
http://htaccessbook.com/67

phpMyAdmin (makes life easier)
http://htaccessbook.com/7w

9.3 Redirecting WordPress feeds

In this section you'll learn a variety of ways to redirect your WordPress feeds to third-party delivery-services like FeedBurner or similar.

Redirecting feeds to FeedBurner

WordPress generates a variety of feeds for your site's content, but there's no default way to track them for statistical purposes. Although it has its downsides, Google's FeedBurner provides a feed-delivery service that provides useful statistical data about who's using your site's feed. All that's needed is a Google account, configuration of your FeedBurner feed, and the following .htaccess rules to do the redirect:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_URI} ^/feed/ [NC]
      RewriteCond %{HTTP_USER_AGENT} !(FeedBurner|FeedValidator) [NC]
      RewriteRule .* http://feeds.feedburner.com/exampleFeedURL [L,R=302]
</IfModule>
```

What is My WordPress Feed URL?
http://htaccessbook.com/68

Redirect WordPress Feeds to Feedburner via htaccess
http://htaccessbook.com/6a

FeedBurner Subscriber Count Problems
http://htaccessbook.com/69

Serve Yourself: Why Feedburner Needs a Feed Fix
http://htaccessbook.com/6b

There are two things to edit in this code. In the first rewrite-condition, edit "`/feed/`" to match that of the WordPress feed you would like to use. Then in the rewrite-rule, edit the "`exampleFeedURL`" to match the chosen name for your FeedBurner feed.

If you're also delivering your comment-feed via FeedBurner, use this code instead:

```
<IfModule mod_rewrite.c>
 RewriteCond %{REQUEST_URI} ^(/comments)?/feed/?$ [NC]
 RewriteCond %{HTTP_USER_AGENT} !(FeedBurner|FeedValidator) [NC]
 RewriteRule ^feed/?$ http://feeds.feedburner.com/exampleFeed [L,NC,R=302]
 RewriteRule ^comments/feed/?$ http://feeds.feedburner.com/commentsFeed [L,NC,R=302]
</IfModule>
```

Here, the two rewrite-conditions are basically saying, "if the request is for either the comment-feed or the main-feed, AND the request is not being made from FeedBurner, then execute the rewrite-rules." If both conditions are met, the two rewrite-rules will redirect each feed to their respective FeedBurner feeds.

Redirecting category-feeds to FeedBurner

This section is about redirecting individual-category feeds to their respective FeedBurner URLs. We will also look at the complete code required to redirect all of the above: the main feed, comments feed, and of course any number of individual category feeds.

If unsure about which format to use, RSS2 is a good choice. So for your main articles, the feed URL would be: `http://example.com/feed/`

Note that we're using a 302-redirect, which tells search-engines that the redirect is temporary.

Without the second `RewriteCond`, requests for your feed would result in an infinite loop.

Here is the generalized .htaccess code that we will be using to redirect three hypothetical categories, "business", "pleasure", and "nonsense":

```
<IfModule mod_rewrite.c>
 RewriteCond %{HTTP_USER_AGENT} !(FeedBurner|FeedValidator) [NC]
 RewriteRule /business/feed/?$ http://feeds.feedburner.com/businessFeed [L,NC,R=302]
 RewriteRule /pleasure/feed/?$ http://feeds.feedburner.com/pleasureFeed [L,NC,R=302]
 RewriteRule /nonsense/feed/?$ http://feeds.feedburner.com/nonsenseFeed [L,NC,R=302]
</IfModule>
```

To use, edit the category names and the FeedBurner feed-name to match your own. This technique is easily combined with the previous technique for redirecting the main-feed and comment-feed to FeedBurner.

```
<IfModule mod_rewrite.c>
 RewriteCond %{HTTP_USER_AGENT} !(FeedBurner|FeedValidator) [NC]
 RewriteRule ^/feed/?$ http://feeds.feedburner.com/exampleFeedURL [L,NC,R=302]
 RewriteRule /comments/feed/?$ http://feeds.feedburner.com/commentsFeed [L,NC,R=302]
 RewriteRule /business/feed/?$ http://feeds.feedburner.com/businessFeed [L,NC,R=302]
 RewriteRule /pleasure/feed/?$ http://feeds.feedburner.com/pleasureFeed [L,NC,R=302]
 RewriteRule /nonsense/feed/?$ http://feeds.feedburner.com/nonsenseFeed [L,NC,R=302]
</IfModule>
```

With that code in place, your server will redirect all requests for your blog's main, comments, and specified category feeds to their respective Feedburner feeds. All other

Note that FeedBurner is popular, but these days there are some great alternatives, and these .htaccess redirect-techniques work for any of them. Just change the "`http://feeds.feedburner.com/nonsenseFeed`" with the full URL of your target feed-URL.

7 Best Alternatives to FeedBurner and Why you should be using them: http://htaccessbook.com/6c

types of feeds (tags, post comments, author feeds, etc.) will not be affected and will appear as normal when requested.

Redirecting default query-string feed-formats

One last quick technique for the case where permalinks aren't enabled on a WP-powered site. As discussed in the original article, we want to redirect WP's default, query-string feed-formats, for example:

```
http://example.com/blog/?feed=rss
http://example.com/blog/?feed=rss2
http://example.com/blog/?feed=atom
http://example.com/blog/?feed=comments-rss2
```

So, similar to the previous method, here we are redirecting all of the differently formatted main-content feeds. Edit the following to match your own, then add to .htaccess:

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_USER_AGENT} !(feedburner|feedvalidator) [NC]
      RewriteCond %{QUERY_STRING} feed=(rss|rss2|atom) [NC]
      RewriteRule .* http://feeds.feedburner.com/exampleFeed? [L,NC,R=302]
</IfModule>
```

And likewise for the comments-feed:

WordPress Feedburner HTAccess Redirect for Default (Non-Permalink) Feed URLs: http://htaccessbook.com/6f

Custom Shortlinks for WordPress http://htaccessbook.com/6h


```
<IfModule mod_rewrite.c>
 RewriteCond %{HTTP_USER_AGENT} !(feedburner|feedvalidator) [NC]
 RewriteCond %{QUERY_STRING} feed=(comments-rss|comments-rss2|comments-atom) [NC]
 RewriteRule .* http://feeds.feedburner.com/commentsFeed? [L,NC,R=302]
</IfModule>
```

In either of these techniques, if the request isn't from FeedBurner, we're checking the `QUERY_STRING` parameter and redirecting matching requests to your FB feed-URL. Note also that we're sending a 302-status header to indicate that this is a temporary redirect.

9.4 WordPress security techniques

Rounding out the WordPress-tricks chapter, here are some excellent techniques for improving the security of your site. We've already seen an entire chapter on securing your site with .htaccess, but there are a few additional tricks that are specific to WordPress that are worth covering here.

Block spam by denying access to no-referrer requests

What we have here is an excellent method for preventing a great deal of blog spam. With a few strategic lines placed in your htaccess file, you can prevent spambots from dropping spam-bombs by denying access to all requests that do not originate from your domain.

How does it work? Well, when a legitimate user decides to leave a comment on your blog,

Note: to send a 301 "Permanent" status instead, replace the "301" in the RewriteRule, like so:

```
http://feeds.feedburner.com/commentsFeed? [L,NC,R=301]
```

Official WP-guide to Hardening WordPress http://htaccessbook.com/6i

Also check out my Lynda.com video-tutorial on developing secure WP sites: http://htaccessbook.com/3k

they have (hopefully) read the article for which they wish to leave a comment, and have subsequently loaded your blog's comment template (e.g., `comments.php`). After filling out the comment form, the user clicks the "submit" button, which then initiates the PHP file/script that actually processes the comment for the world to see.

This is why the HTTP-referrer for all legitimate comments will be your own domain. Automated spam-robots typically target the comment-processing script directly, bypassing your `comments.php` form altogether. Such activity results in HTTP-referrers that are not from your domain, and usually empty as well.

Thus, by blocking all requests for the comments-processing script that are not sent directly from your domain, or are empty, you eliminate a large percentage of blog-spam. Sound good? Here is the code to add to your site's root .htaccess file:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} POST
      RewriteCond %{REQUEST_URI} wp-comments-post.php
      RewriteCond %{HTTP_REFERER} !^http://example.com [NC,OR]
      RewriteCond %{HTTP_USER_AGENT} ^$
      RewriteRule .* http://example.com/landing-page-for-spammers/ [R=301,L]
</IfModule>
```


For more info, see "Block Spam by Denying Access to No-Referrer Requests": http://htaccessbook.com/6j

This name of the script is `wp-comments-post.php`, and is located in the root directory of the WP installation.

To use this technique, edit the "`example.com`" with your own domain, and also set the "landing-page" URL to someplace super-special for your spamming visitors.

Secure posting for visitors

Normally, when visitors post a comment to your site, specific types of client data are associated with the request. Commonly, a client will provide a user-agent, a referrer, and a host-header. When any of these variables is absent, there is reason to suspect foul play.

For example, virtually all browsers identify themselves with a particular user-agent header, even if the information is false. Conversely, malicious scripts directly posting spam and other payloads to your site frequently operate without specifying a user-agent.

Likewise, empty referrer-headers should be taken as suspect. Unless special privacy software is being used, the referrer-information is generally present. Further, when a visitor posts a comment at your site, the referrer-string for that post request will be the URL of that particular page. Thus, as with blank user-agent requests, no-referrer requests are frequently indicative of spam and other malicious behavior.

Another important piece of information provided by all legitimate clients is the host request-header. The host-header specifies the Internet-host and port-number of the requested resource. This information is required for all clients making HTTP/1.1 requests.

See section 7.8 for some great ways to deal with bad requests.

Note: PayPal — for whatever reason — employs an empty/blank user-agent for certain transactions.

For more information, see my article, "Secure Visitor Posting for WordPress": http://htaccessbook.com/6k

Thus, requiring the host request-header field for all posts to your site safely eliminates illicit requests from hitting your server.

By targeting these three circumstances — blank user-agents, empty referrers, and missing host-headers — we can greatly improve the overall security of our WordPress-powered sites. Our weapon of choice to forge this server-side strategy is a custom slice of .htaccess:

```
<IfModule mod_rewrite.c>
      RewriteCond %{REQUEST_METHOD} POST
      RewriteCond %{HTTP_REFERER} !^http://example\.com [NC]
      RewriteCond %{REQUEST_URI} !wp-(login|admin|content|includes|trackback) [NC]
      RewriteCond %{HTTP_USER_AGENT} ^-?$ [OR]
      RewriteCond %{HTTP_REFERER} ^-?$ [OR]
      RewriteCond %{HTTP_HOST} ^-?$
      RewriteRule .* - [F,L]
</IfModule>
```

To implement this .htaccess strategy, replace the "`example.com`" domain with your own. To allow multiple domains, replicate the second rewrite-condition to accommodate, like so:

```
RewriteCond %{HTTP_REFERER} !^http://example\.com [NC]
RewriteCond %{HTTP_REFERER} !^http://example\.net [NC]
RewriteCond %{HTTP_REFERER} !^http://example\.org [NC]
```


See section 7.8 for more creative ways to deal with redirected requests. For example, instead of sending the default 403 "Forbidden" status, you can redirect blocked requests back to the site from whence they came:

```
RewriteRule ^(.*)$ ^http://%{REMOTE_ADDR}/$ [R=301,L]
```


Blocking spam on contact and other forms

Protect your insecure WordPress contact forms against online unrighteousness by verifying the domain from whence the form is called. Remember to replace the "`example.com`" and "`contact.php`" with your domain and file names, respectively.

```
<IfModule mod_rewrite.c>
      RewriteCond %{HTTP_REFERER} !^http://example.com [NC]
      RewriteCond %{REQUEST_POST} /path/to/contact.php
      RewriteRule .* - [F]
</IfModule>
```

In this chapter, we've some great ways to improve the usability, security, and SEO of WordPress-powered sites. But we're not done yet — one more awesome chapter for a few miscellaneous tricks, logging techniques, and troubleshooting guide.

Shameless plug: check out my clean and simple contact-form for WordPress: http://htaccessbook.com/6l

chapter 10

even more techniques

So far, we've seen some great ways .htaccess may be used to improve SEO, enhance usability, tighten security, redirect stuff, and do some cool WordPress tricks. But there's some further techniques that didn't fall neatly into any of these categories, so here's the obligatory "Even more techniques" chapter, which includes a few miscellaneous .htaccess tricks, plus logging techniques, and even a troubleshooting guide.

If I haven't yet said it enough, remember to backup your original .htaccess file and test locally whenever possible. It's virtually impossible to permanently break anything if you can restore the previous working copy of your file. That said, let's wrap things up with even more .htaccess techniques.

The inspiration for this book: my 2006 article, "Stupid .htaccess Tricks" at Perishable Press: http://htaccessbook.com/6m

10.1 Miscellaneous tricks

Just when you think there's no more, BAM! — more awesome .htaccess tricks to enhance and improve your website.

Change the default index page

By default, Apache serves "`index.php`" or "`index.html`" for directory requests, but you can change this with a simple line of .htaccess. The `DirectoryIndex` directive tells the server to serve "`new.html`" as the default directory index. The following rule works sitewide when placed in the root .htaccess file, or locally if placed in a subdirectory.

```
<IfModule mod_dir.c>
      DirectoryIndex new.html new.txt /cgi-bin/new.pl
</IfModule>
```

With this in place, Apache will look for `new.html` and serve it if found. If not, Apache will look for `new.txt` and then finally "`/cgi-bin/new.pl`" as a last resort. This example also shows how the new index file doesn't need to be located in the same directory.

Activate SSI for HTML/SHTML file types

Using Server-Side Includes (SSI) is a great way to dynamically add snippets of HTML to your otherwise static web-pages. Just add the following directives to the root .htaccess file:

See section 8.3 for methods of customizing directory-views when index-files are disabled.

Apache Module mod_dir
http://htaccessbook.com/6o

Apache Tutorial: Introduction to Server Side Includes
http://htaccessbook.com/6n

Apache Tutorial: Introduction to Server Side Includes
http://htaccessbook.com/6n

```
<IfModule mod_mime.c>
      Options +Includes
      AddType text/html .shtml
      AddOutputFilter INCLUDES .shtml
</IfModule>
```

The downside to using this method is that all extensions and links will need changed to ".shtml", which isn't convenient or even possible in some cases. An alternative method is to use the XBitHack directive, which controls the parsing of ordinary HTML documents.

```
XBitHack on
```

As discussed in the Apache docs, "XBitHack tells Apache to parse files for SSI directives if they have the execute bit set. So, to add SSI directives to an existing page, rather than having to change the file name, you would just need to make the file executable using chmod." If you're familiar with using the SSH, the chmod-command looks like this:

```
chmod +x pagename.html
```

Retain rules defined in httpd.conf

Save yourself time and effort by defining replicate rules for multiple virtual hosts once and only once via your httpd.conf file. Then, simply instruct your target .htaccess file(s) to inherit the httpd.conf rules by including this directive:

```
<IfModule mod_rewrite.c>
      RewriteOptions Inherit
</IfModule>
```

10.2 Logging stuff..

One of the most useful ways to understand what's happening on your server is to examine its various access, server, and error logs. As explained in the Apache documentation:

> *The Apache HTTP Server provides a variety of different mechanisms for logging everything that happens on your server, from the initial request, through the URL mapping process, to the final resolution of the connection, including any errors that may have occurred in the process. In addition to this, third-party modules may provide logging capabilities, or inject entries into the existing log files, and applications such as CGI programs, or PHP scripts, or other handlers, may send messages to the server error log.*

Let's look at some specific ways to monitor your site's activity using some different logging techniques. Keep in mind the subject of data-logging with Apache is vast, especially as you get into configuring the main configuration file, httpd.conf. When more info is needed, refer to the Apache docs. Even without access to httpd.conf, much logging is possible.

See the Apache Docs for more info on mod_mime:
http://htaccessbook.com/3r

More information about the XBitHack Directive:
http://htaccessbook.com/6p

Note: Using either -IncludesNOEXEC or -Includes options disables server-side includes (SSI) completely.

Excellent chmod Tutorial
http://htaccessbook.com/7x

Apache Docs: Log Files
http://htaccessbook.com/6q

Apache Tutorial: Dynamic Content with CGI
http://htaccessbook.com/6r

Logging errors

The best way to monitor your site's errors is to examine the default error-log, which is generally located in one of the following directories:

- **RHEL / Red Hat / CentOS / Fedora Linux**: `/var/log/httpd/error_log`
- **Debian / Ubuntu Linux**: `/var/log/apache2/error.log`
- **FreeBSD**: `/var/log/httpd-error.log`

The "`error_log`" records diagnostic information along with any errors that it encounters in processing requests. Unless you customize the `error_log` with the `LogFormat` or `ErrorLogFormat` directive, the default output for each request will look something like this:

```
[Fri Sep 09 10:42:29.902022 2011] [core:error] [pid 35708:tid 4328636416] [client
72.15.99.187] File does not exist: /usr/local/apache2/htdocs/favicon.ico
```

Much customization is possible using Apache's **mod_log_config**, but you need access to the httpd.conf file to make it happen. Generally speaking, the default log-configuration provides plenty of information, but keep in mind that more is possible with httpd.conf.

Logging access

Going beyond error-logging, Apache also logs the details of all requests made to the server.

Ask your host if it's not there or if you can't access it, there's usually a way to do it via the server control-panel.

Typically named "`error_log`" on Unix systems and "`error.log`" on Windows and OS/2.

Apache Module mod_log_config
http://htaccessbook.com/6s

By default, the access-log is located in one of the following directories:

- **RHEL / Red Hat / CentOS / Fedora Linux**: `/var/log/httpd/access_log`
- **Debian / Ubuntu Linux**: `/var/log/apache2/access.log`
- **FreeBSD**: `/var/log/httpd-access.log`

The location and content of the access log are controlled by the `CustomLog` directive. The `LogFormat` directive can be used to further customize and simplify how the log records data. Using these directives, much advanced configuration is possible, but most of the time one of Apache's pre-configured formats will be more than sufficient. These logging formats are defined in the httpd.conf file as such:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

These directives provide formatting for logging of user-agents, referrers, URL-requests, and all of the above. To enable one of these formats, say the one for "combined" logging, simply locate and uncomment the following line in the httpd.conf file:

```
CustomLog logs/access_log combined
```

For more information, check out the documentation for Apache's mod_log_config: http://htaccessbook.com/6s

Once enabled, Apache will log detailed information about the request in a file named "access_log" that's located in the "logs" directory. Given that we're able to specify a custom-location and filename for our access-logs, we can segregate access-logging into multiple logs. In httpd.conf, uncomment and edit the following directives with the names and locations of your separate log-files:

```
CustomLog logs/access_log  combined
CustomLog logs/referer_log referer
CustomLog logs/agent_log   agent
```

Using a combination of the LogFormat and CustomLog directives, it's possible to completely customize the manner in which Apache logs your site's activity. For the sake of completeness, here is what the "combined" logging method generates per request:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200
2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

And here is an example showing the "common" type of logging:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

It's amazing how much control Apache provides over the logging process. You can even do conditional logs if you're into that sort of thing. See the Apache Docs for more information.

Apache Log Viewer
http://htaccessbook.com/6t

Apache Web Log Analyzer
http://htaccessbook.com/6u

logQL is an simple and extremely flexible Log Parser and analysis tool: http://www.logql.com/

How to log mod_rewrite activity

Apache's **mod_rewrite** module provides its own mechanism for logging rewrites on the server. As with previous logging methods, rewrite-logging is configured in the main-configuration file. Once you have access, add the following directives to httpd.conf:

```
<IfModule mod_rewrite.c>
      # Roll your own Rewrite log
      # Log details via scale of 1 to 9
      # 1 = few details, 5 = enough details, 9 = too much detail
      RewriteEngine On
      RewriteLog "/path/to/your/rewrite.log"
      RewriteLogLevel 5
</IfModule>
```

Two things to edit here: **1)** the path to your rewrite-log, and **2)** the level of detail that you want to record. In my experience, "5" is just the right amount, but "9" is there if needed.

Customizing logs via .htaccess

Even when you don't have access to the httpd.conf file, you can do some light customizing via the .htaccess file. Once logging is enabled on your server, we can use Apache's SetEnvIf directive to filter requests based on specific criteria. For example, let's say that we don't want to log requests for images in the access-log. Add this code to the root .htaccess file:

Even more "How to log mod_rewrite activity":
http://htaccessbook.com/6v

Tip: make sure the file is writable by the server.

```
<IfModule mod_setenvif.c>
      SetEnvIf Request_URI \.gif image-request
      SetEnvIf Request_URI \.jpg image-request
      SetEnvIf Request_URI \.jpe image-request
      SetEnvIf Request_URI \.png image-request
      SetEnvIf Request_URI \.ico image-request
      CustomLog logs/access_log common env=!image-request
</IfModule>
```

This technique sets an environmental variable of "image-request" for all requested images. Then the `CustomLog` directive instructs Apache to exclude any such image requests from the specified access-log. Using regular-expressions, similar directives may be constructed to target just about any type of request. Further, in addition to Host, User-Agent, Referrer, and Accept-Language, the following attributes of the request may be filtered:

- **Remote_Host** — the hostname (if available) of the client making the request
- **Remote_Addr** — the IP address of the client making the request
- **Server_Addr** — the IP address of the server on which the request was received
- **Request_Method** — the name of the request method (`GET`, `POST`, `HEAD`, `POST`, etc.)
- **Request_Protocol** — the name and version of the request-protocol
- **Request_URI** — the resource requested on the HTTP request-line

For `Server_Addr`: only with versions later than 2.0.43.

For `Request_Protocol`: e.g., "`HTTP/0.9`", "`HTTP/1.1`", etc.

For `Request_URI`: i.e., the portion of the URL following the scheme and host-portion without the query-string.

Let's look at one more example, whereby we exclude visits from Googlebot:

```
<IfModule mod_setenvif.c>
      SetEnvIfNoCase User-Agent googlebot
      CustomLog logs/access_log common env=!googlebot
</IfModule>
```

And with that, you're all set to log just about anything. For more information, see the Apache Documentation.

How to enable PHP error-logging

Tracking your site's PHP-errors is an excellent way to manage and troubleshoot unexpected issues related to plugins and themes. Even better, monitoring PHP-errors behind the scenes via private log is far better than trying to catch them as they appear at random visits. Thanks to the magical powers of .htaccess, there is an easy way to implement this effective strategy.

Hide PHP errors from visitors

This technique suppresses PHP-errors via htaccess when using PHP as an Apache module. You will need "`AllowOverride Options`" or "`AllowOverride All`" privileges to do so. Hiding PHP-errors from the public is a good security measure. To do so, add the following directives to your domain's httpd.conf or to any .htaccess file:

Apache Module mod_setenvif
http://htaccessbook.com/2g

Media Temple's guide to logging PHP errors
http://htaccessbook.com/7g

Advanced PHP Error Handling via PHP
http://htaccessbook.com/6x

Advanced PHP Error Handling via htaccess
http://htaccessbook.com/6w

```
<IfModule mod_php5.c>
      php_flag display_startup_errors off
      php_flag display_errors off
      php_flag html_errors off
</IfModule>
```

With that in place, PHP-errors will no longer be displayed publicly on your site. This eliminates a potential security risk, and keeps those ugly, unintelligible PHP-errors from breaking your site-layout and disorienting your visitors. No editing required for this code.

Enable private PHP error logging

Now that we have hidden PHP-errors from public view, let's enable the logging of PHP-errors so that we can privately keep track of them. This is done by adding the following directives to httpd.conf or to any .htaccess file:

```
<IfModule mod_php5.c>
      php_flag log_errors on
      php_value error_log /path/to/php_errors.log
</IfModule>
```

For this to work, you will need to edit the path in the last line to reflect the actual location of your "`php_errors.log`" file. Of course, you will need to create this file and subsequently set the file permissions to `755` or, if necessary, `777`. Finally, you need to secure the log file

3 Ways to Monitor PHP Errors
http://htaccessbook.com/6y

Understanding permissions on an Apache Unix server
http://htaccessbook.com/6z

itself by adding this final line of code to your htaccess file:

```
<Files php_errors.log>
      Order Allow,Deny
      Deny from all
      Satisfy All
</Files>
```

Then, after everything is in place, check that everything is working by triggering a few PHP-errors. You may also want to verify protection of your error-log by trying to access it via a browser.

10.3 Troubleshooting guide

Once you get the hang of how Apache and .htaccess files work, the errors you encounter should be easy to identify, interpret, and resolve using data available in your access, error, and other logs. Even so, things tend to go wrong even for experts, and fortunately there is an effective way to troubleshoot problems when they arise.

This section provides steps to guide you through the process of identifying issues and resolving them as expediently as possible. For more advanced help, refer to the links listed at the end of this section.

See previous section for more information on seting up various types of server logs.

Make sure Apache is running

It may sound obvious, but if Apache isn't running, things just aren't going to work. The simplest way to verify that Apache is working is to request a page from your site. If it's returned, then Apache is running.

Check AllowOverride in httpd.conf

If Apache is running but .htaccess doesn't seem to be working, the most common reason is that the `AllowOverride` directive is set to "`None`" in the main configuration file. If it is disabled, you can enable it by changing its value to "`All`" or by specifying specific directives based on your needs.

Also note that `AllowOverride` may be set to "`None`" by default, and then enabled only in specific directories using `Directory` containers, like so:

```
<Directory "/path/to/htdocs/directory">
      AllowOverride FileInfo AuthConfig Limit Indexes
      Options All
</Directory>
```

If you don't have access to httpd.conf, ask your web-host if .htaccess is enabled on the server. And if it's not, perhaps they would be so generous as to enable it for you.


Apache Module mod_access_compat
http://htaccessbook.com/71


Troubleshoot Apache problems with these tips
http://htaccessbook.com/70


Why cheap web hosting is expensive
http://htaccessbook.com/72

Verify that a specific module is running

If .htaccess seems to be working, but not some specific module, say **mod_rewrite**, check to see if it's being loaded from the httpd.conf file. Access the httpd.conf file and look for the module-loading section, which begins with something similar to this:

```
# LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule auth_anon_module modules/mod_auth_anon.so
 :
 :
```

Scan the list for the module that isn't working and make sure that it's not commented-out with a hash-symbol "`#`". Depending on how your web-host has configured the server, you may find that **mod_rewrite** is disabled. In our example, we see that the first module, **mod_access**, is disabled, so to enable it we remove the hash-symbol (or pound-sign), like so:

```
LoadModule access_module modules/mod_access.so
```

If your host doesn't allow access to httpd.conf and doesn't support the required module, try asking and if that doesn't work try finding a better host. You'll thank yourself for it.


To see which modules are currently compiled into the server, you can use the `-l` command-line option. You can also see what modules are loaded dynamically using the `-M` command-line option.


How to check and enable mod_rewrite module in apache
http://htaccessbook.com/7r

Check the server logs

As discussed previously, Apache can log a tremendous amount of data, providing many clues to the origin and nature of many common errors. Especially useful for errors is the error-log, which should contain the information required to get things up and running.

Check HTTP status-codes

With each request logged by Apache, you'll see its HTTP status-code, which can be a huge help in diagnosing errors and other issues. For example, scanning your access logs, you may notice a lot of 404-errors, which means visitors aren't finding the resources they're looking for on your server. Likewise, 403 "Forbidden" errors are commonly seen when a request is blocked due to server-configuration, blacklist-rules, and so on.

Check your code for errors

Another good way to diagnose and prevent errors is to run a configuration-test on your httpd.conf file. Included as part of the **apachectl** program, Apache's error-checking tool is executed from the command-line to scan your configuration file for any errors. To use it, enter the following command:

```
# ./apachectl configtest
```

If any errors are found, Apache will tell you about it. For example, if there is an error in the

Including a check for proper `AllowOverride` directives. See the "httpd.conf" sidebars in section 3.2 and 4.5 for more information.

Apache Docs: Configuration Files
http://htaccessbook.com/e

Apache Docs: AllowOverride Directive
http://htaccessbook.com/73

AllowOverride directive, Apache will return something similar to the following:

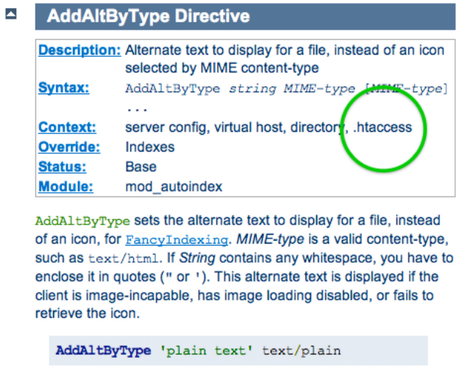
```
Syntax error on line 50 of /usr/local/apache/conf/httpd.conf:
Invalid command 'Allowoveride', perhaps misspelled or defined by a module not
included in the server configuration
```

If, on the other hand, no errors are found, Apache sends the "all-clear" message:

```
Syntax OK
```

Is the directive allowed in .htaccess

All of the code in this book — unless specifically labeled otherwise — is intended for use in .htaccess files. But this is not the case for all Apache directives. If, in your online travels, you encounter some awesome Apache directives, you can verify their allowed context in the Apache Docs. See the screenshot for an example.



AddAltByType Directive

Description:	Alternate text to display for a file, instead of an icon selected by MIME content-type
Syntax:	`AddAltByType string MIME-type [MIME-type] ...`
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

`AddAltByType` sets the alternate text to display for a file, instead of an icon, for `FancyIndexing`. *MIME-type* is a valid content-type, such as `text/html`. If *String* contains any whitespace, you have to enclose it in quotes (" or '). This alternate text is displayed if the client is image-incapable, has image loading disabled, or fails to retrieve the icon.

```
AddAltByType 'plain text' text/plain
```

Isolating problems in .htaccess files

I call this technique the "halving-method", whereby you locate problematic code by removing half of the code, then testing, then repeating until you've isolated the issue.

When looking up Apache directives, check its "Context" for ".htaccess" as shown in the screenshot.

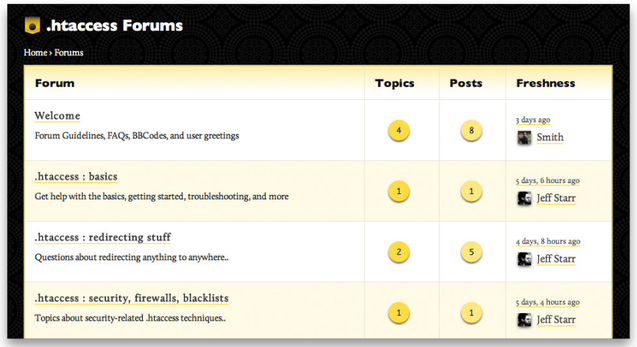
The Halving Method of Identifying Problematic Code
http://htaccessbook.com/74

For example, if you add 100 blacklist-directives to your .htaccess file and suddenly the server returns a 500-error, you know immediately that the newly added code is probably the issue. So you remove the first half of the directives and try again. This time the page loads fine, so you know the problem code is contained in that first half of rules. So you replace half of the rules you removed and try again, repeating the process until you've isolated the issue. I use this method frequently and it always gets me there in about two or three iterations. See the footer link on the previous page for more information.

10.4 Where to get help with Apache

If all else fails, or if you have questions, suggestions, or concerns, there are some good places to go for help.

- The .htaccess forums for this book: http://htaccessbook.com/forums/
- The official Apache Documentation: http://httpd.apache.org/docs/
- The Web, Google, forums, etc.
- IRC channel #Apache



Visit the .htaccess Forums for help:
`http://htaccessbook.com/forums/`

Apache: "The Number One HTTP Server On The Internet": http://httpd.apache.org/

Apache HTTP Server Tutorial: .htaccess files http://htaccessbook.com/b

Apache HTTP Server Documentation http://httpd.apache.org/docs/

Apache HTTP Server Support http://httpd.apache.org/support.html

Thank you

Thank you for buying this book. For over six years, I've freely shared my .htaccess and web-design tips at Perishable Press and elsewhere. I enjoy helping people succeed on the Web, and am able to do so thanks to awesome people like you. As a way of showing my appreciation, purchase of the book includes free updates for as long as I can crank them out. Fortunately, .htaccess doesn't change as fast as, say, WordPress, but when it does and something needs changed in the book, I'm on it, and will release an updated version, which will be 100% free for you, because you bought the book. So again, thank you.

About the author

Jeff Starr has been designing and developing sites on Apache servers for over 10 years, sharing hundreds of .htaccess tips and tricks at Perishable Press and elsewhere. He enjoys writing about all aspects of web design and development, with WordPress and Web Security among his favorite pursuits. Together with Chris Coyier, Jeff co-authors the book and site, Digging into WordPress. And just to be sure that he stays busy, Jeff also runs his own web-design business, Monzilla Media, and works as Editor for Smashing Magazine's exclusive WordPress section.

Perishable Press http://perishablepress.com/

Digging into WordPress http://digwp.com/

Monzilla Media http://mon.co/m

Jeff Starr on Twitter https://twitter.com/perishable

references

Here is a complete list of web pages referenced in the book via shortened URLs.

3 Ways to Monitor PHP Errors
http://digwp.com/2009/07/monitor-php-errors-wordpress/

4G Series: The Ultimate Referrer Blacklist, Featuring Over 8000 Banned Referrers
http://perishablepress.com/4g-ultimate-referrer-blacklist/

5 Easy Ways to Display Syntax Highlighted PHP Code
http://perishablepress.com/5-easy-ways-to-display-syntax-highlighted-php-code/

5G Blacklist 2012
http://perishablepress.com/5g-blacklist-2012/

6G Beta
http://perishablepress.com/6g-beta/

7 Best Free Alternatives to Feedburner and Why you should be using them
http://knolzone.com/7-best-free-alternatives-to-feedburner-and-why-you-should-be-using-them/

8 Canonicalization Best Practices In Plain English
http://searchengineland.com/8-canonicalization-best-practices-in-plain-english-44475

10 Ways To Beef Up Your Website's Security
http://www.smashingmagazine.com/2010/06/15/10-ways-to-beef-up-your-websites-security/

15KB Of Fame: HTTP Cache Poisoning via Host Header Injection
http://carlos.bueno.org/2008/06/host-header-injection.html

404 Error Pages: Reloaded
http://www.smashingmagazine.com/2007/08/17/404-error-pages-reloaded/

2010 User-Agent Blacklist
http://perishablepress.com/2010-user-agent-blacklist/

Advanced PHP Error Handling via htaccess
http://perishablepress.com/advanced-php-error-handling-via-htaccess/

Advanced PHP Error Handling via PHP
http://perishablepress.com/advanced-php-error-handling-via-php/

Ajax-Powered Error Logs
http://perishablepress.com/ajax-error-log/

Allow Feedburner Access to Hotlink-Protected Images
http://perishablepress.com/allow-feedburner-access-to-hotlink-protected-images/

Allow Google Reader Access to Hotlink-Protected Images
http://perishablepress.com/allow-google-reader-to-access-hotlink-protected-images/

Alphabetical HTTP_USER_AGENT IndexNo more CSS hacks:
http://www.siteware.ch/webresources/useragents/db.html

Amazon Books - mod_rewrite
http://www.amazon.com/exec/obidos/asin/1590595610/drbacchus/

Apache - Command List
http://www.apache.com/tag/command-list/

Apache .htaccess mod_rewrite and mod_alias Rewriting and Redirecting Examples
http://www.tedpavlic.com/post_apache_rewriting_examples.php

Apache .htaccess query string redirects
http://www.simonecarletti.com/blog/2009/01/apache-query-string-redirects/

Apache Commands
http://www.lagmonster.org/docs/apachecommands.html

Apache HTTP Server - Access Control
http://httpd.apache.org/docs/current/howto/access.html

Apache HTTP Server - Advanced Techniques with mod_rewrite
http://httpd.apache.org/docs/current/rewrite/advanced.html#time-dependent

Apache HTTP Server - Apache Performance Tuning
http://httpd.apache.org/docs/current/misc/perf-tuning.html

Apache HTTP Server - Apache Tutorial: Dynamic Content with CGI
http://httpd.apache.org/docs/current/howto/cgi.html

Apache HTTP Server - Apache Tutorial: Introduction to Server Side Includes
http://httpd.apache.org/docs/current/howto/ssi.html

Apache HTTP Server - Authentication and Authorization
http://httpd.apache.org/docs/current/howto/auth.html

Apache HTTP Server - Caching Guide
http://httpd.apache.org/docs/current/caching.html

Apache HTTP Server - Configuration Files
http://httpd.apache.org/docs/current/configuring.html

Apache HTTP Server - Configuration Sections
http://httpd.apache.org/docs/current/sections.html

Apache HTTP Server - core
http://httpd.apache.org/docs/current/mod/core.html

Apache HTTP Server - Environment Variables in Apache
https://httpd.apache.org/docs/current/env.html

Apache HTTP Server - htpasswd - Manage user files for basic authentication
http://httpd.apache.org/docs/current/programs/htpasswd.html

Apache HTTP Server - Log Files
http://httpd.apache.org/docs/current/logs.html

Apache HTTP Server - Mapping URLs to Filesystem Locations
http://httpd.apache.org/docs/current/urlmapping.html#outside

Apache HTTP Server - mod_access_compat
http://httpd.apache.org/docs/current/mod/mod_access_compat.html

Apache HTTP Server - mod_alias
http://httpd.apache.org/docs/current/mod/mod_alias.html

Apache HTTP Server - mod_auth
http://httpd.apache.org/docs/2.0/mod/mod_auth.html

Apache HTTP Server - mod_authn_file
http://httpd.apache.org/docs/current/mod/mod_authn_file.html

Apache HTTP Server - mod_autoindex
http://httpd.apache.org/docs/current/mod/mod_autoindex.html

Apache HTTP Server - mod_deflate
http://httpd.apache.org/docs/current/mod/mod_deflate.html

Apache HTTP Server - mod_dir
http://httpd.apache.org/docs/current/mod/mod_dir.html

Apache HTTP Server - mod_env
https://httpd.apache.org/docs/current/mod/mod_env.html

Apache HTTP Server - mod_expires
http://httpd.apache.org/docs/current/mod/mod_expires.html

Apache HTTP Server - mod_filter
http://httpd.apache.org/docs/current/mod/mod_filter.html

Apache HTTP Server - mod_headers
http://httpd.apache.org/docs/current/mod/mod_headers.html#header

Apache HTTP Server - mod_include
http://httpd.apache.org/docs/current/mod/mod_include.html#xbithack

Apache HTTP Server - mod_log_config
http://httpd.apache.org/docs/current/mod/mod_log_config.html

Apache HTTP Server - mod_mime
http://httpd.apache.org/docs/current/mod/mod_mime.html

Apache HTTP Server - mod_rewrite
http://httpd.apache.org/docs/current/mod/mod_rewrite.html

Apache HTTP Server - mod_setenvif
http://httpd.apache.org/docs/current/mod/mod_setenvif.html

Apache HTTP Server - mod_speling
http://httpd.apache.org/docs/current/mod/mod_speling.html

Apache HTTP Server - Module Index
https://httpd.apache.org/docs/current/mod/

Apache HTTP Server Tutorial: .htaccess files
http://httpd.apache.org/docs/current/howto/htaccess.html

Apache HTTP ServerCache - mod_headers
http://httpd.apache.org/docs/current/mod/mod_headers.html

Apache Logs Viewer
http://www.apacheviewer.com/logfiles.php

Apache RewriteRule and query string
http://www.simonecarletti.com/blog/2009/01/apache-rewriterule-and-query-string/

Apache SSL in htaccess examples
http://www.askapache.com/htaccess/apache-ssl-in-htaccess-examples.html

Apache Web Log Analyzer
http://www.ghacks.net/2009/09/07/apache-web-log-analyzer/

Apache Week. Publishing Pages with PUT
http://www.apacheweek.com/features/put

AskApache Password Protection, For WordPress
http://www.askapache.com/wordpress/htaccess-password-protect.html

Automatically Version Your CSS and JavaScript Files
http://particletree.com/notebook/automatically-version-your-css-and-javascript-files/

Basic HTML Elements
http://www.thelivingcanvas.com/guither/webtricks/basiccontainer.html

Beginner's Guide to the .htaccess File
http://www.johnfdoherty.com/beginners-guide-to-the-htaccess-file/

Behind the Scenes with Apache's .htaccess
http://brainstormsandraves.com/archives/2005/10/09/htaccess/

Best Free Web Browser
http://www.techsupportalert.com/best-free-web-browser.htm

Best Practices for Speeding Up Your Web Site
http://developer.yahoo.com/performance/rules.html

Better Default Directory Views with HTAccess
http://perishablepress.com/better-default-directory-views-with-htaccess/

Archive of articles on blacklisting bad bots, evil scripts, and other scumbags
http://perishablepress.com/tag/blacklist/

Blacklist Candidate Number 2008-04-27
http://perishablepress.com/blacklist-candidate-number-2008-04-27/

Block Spam by Denying Access to No-Referrer Requests
http://perishablepress.com/block-spam-by-denying-access-to-no-referrer-requests/

Block Tough Proxies
http://perishablepress.com/block-tough-proxies/

Bot-trap, a bad web robot blocker
http://danielwebb.us/software/bot-trap/

Brian Huisman - Apache mod_rewrite & mod_alias tricks you should know
http://my.opera.com/GreyWyvern/blog/2007/09/12/apache-mod-rewrite

Browser sniffing with .htaccess
http://v4.thewatchmakerproject.com/blog/no-more-css-hacks-browser-sniffing-with-htaccess/

Browser Statistics
http://www.w3schools.com/browsers/browsers_stats.asp

Browser Wars 2011
http://www.unleashed-technologies.com/blog/2011/01/24/browser-wars-2011

Building the 5G Blacklist
http://perishablepress.com/building-the-5g-blacklist/

Building the Perishable Press 4G Blacklist
http://perishablepress.com/building-the-perishable-press-4g-blacklist/

Cache Poisoning
https://www.owasp.org/index.php/Cache_Poisoning

Caching Tutorial for Web Authors and Webmasters
http://www.mnot.net/cache_docs/

Canonical URLs and Subdomains with Plesk
http://perishablepress.com/canonical-urls-subdomains-plesk/

Canonicalization - SEO Best Practices
http://www.seomoz.org/learn-seo/canonicalization

Carbonara the right way
http://elliottrichmond.wordpress.com/2012/05/05/carbonara-the-right-way/

Cascading Style Sheets
http://www.w3.org/Style/CSS/Overview.en.html

CERT/CC Denial of Service
http://www.cert.org/tech_tips/denial_of_service.html

Changing The Site URL
http://codex.wordpress.org/Changing_The_Site_URL

check and enable mod_rewrite module in apache of xampp or wamp
http://roshanbh.com.np/2008/04/check-enable-mod_rewrite-apache.html

Classless InterDomain Routing: CIDR Explained
http://www.orbit-computer-solutions.com/CIDR.php

Clean Up Malicious Links with HTAccess
http://perishablepress.com/clean-up-links-htaccess/

Close your website temporarily with Apache htaccess
xhttp://25yearsofprogramming.com/blog/20070704.htm

Comparison of WAMPs
http://en.wikipedia.org/wiki/Comparison_of_WAMPs

Complete MIME Types List
http://www.freeformatter.com/mime-types-list.html

Comprehensive guide to .htaccess
http://www.javascriptkit.com/howto/htaccess.shtml

Compressing files with mod_deflate
http://kb.mediatemple.net/questions/1567/Compressing+web+pages+with+mod_deflate#dv

Conditional GET Request -- GET / HTTP/1.1
http://ruturajv.wordpress.com/2005/12/27/conditional-get-request/

Controlling Proxy Access with HTAccess
http://perishablepress.com/controlling-proxy-access-with-htaccess/

Cookie Protected Directories
http://www.willmaster.com/library/cookies/cookie-protected-directories.php

Cookie Stuffing with .htaccess
http://www.chewie.co.uk/blackhat/cookie-stuffing-with-htaccess/

Crazy Advanced mod_rewrite Tutorial
http://www.askapache.com/htaccess/crazy-advanced-mod_rewrite-tutorial.html

Create A Network
http://codex.wordpress.org/Create_A_Network

Creating the Ultimate htaccess Anti-Hotlinking Strategy
http://perishablepress.com/creating-the-ultimate-htaccess-anti-hotlinking-strategy/

Cross-site scripting
http://en.wikipedia.org/wiki/Cross-site_scripting

Custom HTTP Errors via htaccess
http://perishablepress.com/custom-http-errors-via-htaccess/

Custom Shortlinks for WordPress
http://old.deanjrobinson.com/article/custom-shortlinks-for-wordpress/

Developing WordPress Locally With MAMP
http://wp.smashingmagazine.com/2011/09/28/developing-wordpress-locally-with-mamp/

Download Houdini for Mac - Create, access, or delete hidden folders
http://www.macupdate.com/app/mac/26729/houdini

(dv) 4.0:PHP error logging
http://wiki.mediatemple.net/w/(dv)_4.0:PHP_error_logging

ebSphere Notes: Cache-control vs. Expires
http://wpcertification.blogspot.com/2010/07/cache-control-vs-expires.html

Eight Ways to Blacklist with Apache's mod_rewrite
http://jungels.net/articles/mod-rewrite-recipes.html

Ensure basic Web site security with this checklist
http://www.techrepublic.com/blog/security/ensure-basic-web-site-security-with-this-checklist/424

Everything You Ever Wanted to Know about Favicons
http://perishablepress.com/everything-you-ever-wanted-to-know-about-favicons/

Evil Incarnate, but Easily Blocked
http://perishablepress.com/evil-incarnate-but-easily-blocked/

Expires HTTP header: the magic number of YSlow
http://weblogs.java.net/blog/felipegaucho/archive/2007/08/expires_http_he.html

Expires vs. max-age
http://www.mnot.net/blog/2007/05/15/expires_max-age

FeedBurner Subscriber Count Problems
http://perishablepress.com/feedburner-subscriber-count-problems/

Free Online SEO Analysis Tool, Free SEO Tools
http://www.onlineseoanalyzer.com/

GNU Wget
http://www.gnu.org/software/wget/

Hardening WordPress
http://codex.wordpress.org/Hardening_WordPress

How to Write Valid URL Query String Parameters
http://perishablepress.com/how-to-write-valid-url-query-string-parameters/

How to Verify the Four Major Search Engines
http://perishablepress.com/how-to-verify-the-four-major-search-engines/

How To Use HTML Meta Tags
http://searchenginewatch.com/article/2067564/How-To-Use-HTML-Meta-Tags

How to see hidden files in Windows
http://www.bleepingcomputer.com/tutorials/how-to-see-hidden-files-in-windows/

How to Password Protect a Directory on Your Website
http://www.thesitewizard.com/apache/password-protect-directory.shtml

How To Optimize Your Site With GZIP Compression
http://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression/

How to install configure Apache, MySQL and PHP on Mac OS X 10.7 Lion | OS X 10.6 Snow Leopard
http://www.coolestguyplanettech.com/how-to-install-php-mysql-apache-setup-on-os-x

How to install Apache (and PHP+MySQL) on Windows: test your web pages at home
http://www.ampsoft.net/webdesign-l/how-to-install-apache-php-mysql.html

How to edit the hosts file in Mac OS X - Leopard
http://decoding.wordpress.com/2009/04/06/how-to-edit-the-hosts-file-in-mac-os-x-leopard/

How to disable Apache's server signature
http://blog.tersmitten.nl/archives/29

How to create symlink?
http://linux.byexamples.com/archives/19/how-to-create-symlink/

How To Create & Edit The .htaccess File For Your Site
http://www.makeuseof.com/tag/how-to-easily-create-and-edit-htaccess-file-for-your-site/

How to Block Proxy Servers via htaccess
http://perishablepress.com/how-to-block-proxy-servers-via-htaccess/

How (and why) to disable Apache server signature on your web pages
http://nixtechnica.blogspot.com/2007/05/how-and-why-to-disable-apache-server.html

HTTP/1.1: Status Code Definitions
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP/1.1: Request
http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html

HTTP/1.1: Header Field Definitions
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

HTTP/1.1: Caching in HTTP
http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

HTTP Status Codes Checker ~ Server Header Response Code Checking Tool
http://tools.seobook.com/server-header-checker/

HTTP cookies explained
http://www.nczonline.net/blog/2009/05/05/http-cookies-explained/

HTTP Compression Test
http://www.whatsmyip.org/http-compression-test/

HTML5 Boilerplate - .htaccess file
http://html5boilerplate.com/template/htaccess.txt

HTAccess Privacy for Specific IPs
http://perishablepress.com/htaccess-privacy-for-specific-ips/

HTAccess Password-Protection Tricks
http://perishablepress.com/htaccess-password-protection-tricks/

htaccess Password Generator
http://tools.dynamicdrive.com/password/

htaccess Generator
http://cooletips.de/htaccess/

htaccess Combo Pack: WordPress Permalinks and non-www Redirect
http://perishablepress.com/htaccess-combo-pack-wordpress-permalinks-and-non-www-redirect/

htaccess Code for WordPress Multisite
http://perishablepress.com/htaccess-code-for-wordpress-multisite/

htaccess tips
http://virendrachandak.wordpress.com/2011/10/02/htaccess-tips/

htaccess rules for site speed optimization
http://zemalf.com/1343/htaccess-rules-for-site-speed/

htaccess made easy
http://htaccessbook.com/

htaccess Examples: Cookies, Variables, Custom Headers
http://www.askapache.com/htaccess/htaccess-fresh.html

IE image-flicker
http://dean.edwards.name/my/flicker.html

Important note for your custom error pages
http://perishablepress.com/important-note-for-your-custom-error-pages/

Improve Your Website Search Optimization Using Chrome's SEO Site Tools
http://blog.kissmetrics.com/seo-site-tools/

Insights and inspiration for the user experience community
http://www.uxmatters.com/index.php

Installing WordPress
http://codex.wordpress.org/Installing_WordPress

Instruct Search Engines to come back to site after you finish working on it
http://www.askapache.com/seo/503-service-temporarily-unavailable.html

Introduction to HTTP Response Splitting
http://www.securiteam.com/securityreviews/5WP0E2KFGK.html

Invite Only: Visitor Exclusivity via the Opt-In Method
http://perishablepress.com/invite-only-visitor-exclusivity-via-the-opt-in-method/

IP address
http://en.wikipedia.org/wiki/IP_address

Is Your Web Site Cache Friendly?
http://www.netlingo.com/more/cache.php

ISO 8859-1 character set overview
http://htmlhelp.com/reference/charset/

JavaScript Cookie Handling Library and jQuery plugin
http://code.google.com/p/cookies/

Learn how to configure Apache
http://www.techrepublic.com/article/learn-how-to-configure-apache/5076696

Linux Tutorial - Apache Web Login Authentication
http://www.yolinux.com/TUTORIALS/LinuxTutorialApacheAddingLoginSiteProtection.html

List of Apache, MySQL, and PHP packages
http://en.wikipedia.org/wiki/List_of_AMP_packages

Local area network
http://en.wikipedia.org/wiki/Local_area_network

Major IP Addresses Blocks By Country
http://www.nirsoft.net/countryip/

Making "clean" URLs with Apache and PHP
http://evolt.org/node/22880/

Mandelbox World
http://www.youtube.com/watch?v=jwOBEZOfWdE

Media types - vnd.microsoft.icon
http://www.iana.org/assignments/media-types/image/vnd.microsoft.icon

MicroTut: Getting And Setting Cookies With jQuery & PHP
http://tutorialzine.com/2010/03/microtut-getting-and-setting-cookies-with-jquery-php/

mod_gzip - serving compressed content by the Apache webserver
http://schroepl.net/projekte/mod_gzip/

Mod_Rewrite Variables Cheatsheet
http://www.askapache.com/htaccess/mod_rewrite-variables-cheatsheet.html

more .htaccess tips
http://techtalk.virendrachandak.com/more-htaccess-tips/

Move Your WordPress Files Out of the Root Directory
http://digwp.com/2009/07/move-your-wordpress-files-out-of-the-root-directory/

Opt-in or Blacklist?
http://www.spam-whackers.com/blog/2007/08/28/opt-in-or-blacklist/

Optimizing WordPress Permalinks
http://digwp.com/2010/07/optimizing-wordpress-permalinks/

Optimizing WordPress Permalinks with htaccess
http://perishablepress.com/wordpress-permalinks-htaccess/

Order, Allow, and Deny (Apache: The Definitive Guide)
http://docstore.mik.ua/orelly/linux/apache/ch05_06.htm

Permalink Evolution: Customize and Optimize Your Dated WordPress Permalinks
http://perishablepress.com/permalink-evolution-customize-and-optimize-your-dated-wordpress-permalinks/

PHP: List of Supported Timezones
http://php.net/manual/en/timezones.php

phpMyAdmin
http://www.phpmyadmin.net/

Poll: www. or no www.?
http://digwp.com/2012/05/poll-www-no-www/

Prevent hotlinking of images
http://www.htaccesstools.com/hotlink-protection/

Protect Your Site with a Blackhole for Bad Bots
http://perishablepress.com/blackhole-bad-bots/

Proxy server
http://en.wikipedia.org/wiki/Proxy_server

Pushing Beyond Gzipping
http://developer.yahoo.com/blogs/ydn/posts/2010/12/pushing-beyond-gzipping/

Redirect All (Broken) Links from any Domain via HTAccess
http://perishablepress.com/redirect-all-broken-links-from-any-domain-via-htaccess/

Redirect Checker
http://www.internetofficer.com/seo-tool/redirect-check/

Redirect WordPress Feeds to Feedburner via htaccess (Redux)
http://perishablepress.com/redirect-wordpress-feeds-to-feedburner-via-htaccess-redux/

Regex Tutorial, Examples and Reference - Regexp Patterns
http://www.regular-expressions.info/

Regular Expressions Cheat Sheet (V2)
http://www.addedbytes.com/cheat-sheets/regular-expressions-cheat-sheet/

Revving Filenames: don't use querystring
http://www.stevesouders.com/blog/2008/08/23/revving-filenames-dont-use-querystring/

Same Origin Policy
http://www.w3.org/Security/wiki/Same_Origin_Policy

Secure Visitor Posting for WordPress
http://perishablepress.com/secure-visitor-posting-for-wordpress/

Security tips for web developers
http://www.squarefree.com/securitytips/web-developers.html

SEO advice: url canonicalization
http://www.mattcutts.com/blog/seo-advice-url-canonicalization/

SEO Checker: Check SEO Optimization and Find Website Problems
http://www.powermapper.com/products/sortsite/checks/seo-checks.htm

SEO Redirects without mod_rewrite
http://www.askapache.com/htaccess/seo-search-engine-friendly-redirects-without-mod_rewrite.html

SEO: The Free Beginner's Guide From SEOmoz
http://www.seomoz.org/beginners-guide-to-seo

Series Summary: Building the 3G Blacklist
http://perishablepress.com/series-summary-building-the-3g-blacklist/

Serve Alternate Content based on Time
http://www.askapache.com/htaccess/time_hour-rewritecond-time.html

Serve Yourself: Why Feedburner Needs a Feed Fix
http://perishablepress.com/serve-yourself-why-feedburner-needs-a-feed-fix/

Server Headers Check tool
http://www.seocentro.com/tools/online/server-headers-check.html

Setting charset in htaccess
http://www.askapache.com/htaccess/setting-charset-in-htaccess.html

Setting Cookies In WordPress - Trap For Beginners
http://scratch99.com/wordpress/development/setting-cookies-in-wordpress-trap-for-beginners/

Simulate any User Agent or Bot
http://www.botsvsbrowsers.com/SimulateUserAgent.asp

Smart Filtering for Apache
http://www.apachetutor.org/dev/smart-filter

Smarter Slugs ~!@$%^&*()={}[]?
http://digwp.com/2012/01/smarter-slugs/

Smashing UX Design
http://uxdesign.smashingmagazine.com/

Speed up your site with Caching and cache-control
http://www.askapache.com/hacking/speed-site-caching-cache-control.html

Stop 404 Requests for Mobile Versions of Your Site
http://perishablepress.com/stop-404-requests-for-mobile-versions-of-your-site/

Stupid htaccess Tricks
http://perishablepress.com/stupid-htaccess-tricks/

TCP/IP basics: IP address, Netmask, Network
http://faqintosh.com/risorse/en/guides/net/tcp/basic/

The Apache HTTP Server Project: Documentation
http://httpd.apache.org/docs/

The Apache HTTP Server Project - Download
http://httpd.apache.org/download.cgi

The Halving Method of Identifying Problematic Code
http://perishablepress.com/the-halving-method-of-identifying-problematic-code/

The htaccess Rules for all WordPress Permalinks
http://perishablepress.com/the-htaccess-rules-for-all-wordpress-permalinks/

The Indie Publisher's mod_rewrite Recipe Book
http://jungels.net/articles/mod-rewrite-recipes.html

The Intricacies of Mechanoid Eyeballs HD
http://www.youtube.com/watch?v=Yb5MRbgNKSk

The Ultimate Guide to .htaccess Files
http://net.tutsplus.com/tutorials/other/the-ultimate-guide-to-htaccess-files/

Three Ways to Allow Hotlinking in Specific Directories
http://perishablepress.com/three-ways-to-allow-hotlinking-in-specific-directories/

Top 15 Most Popular Search Engines
http://www.ebizmba.com/articles/search-engines

Top Ten Pink Floyd Songs for Audiophiles
http://perishablepress.com/top-ten-pink-floyd-songs-for-audiophiles/

Trap bad bots in a bot trap
http://www.kloth.net/internet/bottrap.php

Troubleshoot Apache problems with these tips
http://www.techrepublic.com/article/troubleshoot-apache-problems-with-these-tips/6049506

Turning off the Internet Explorer image toolbar
http://www.killersites.com/articles/newsletterArchive/Newsletter_October27_2003.htm

Tutorial for chmod
http://catcode.com/teachmod/

Ultimate htaccess Blacklist
http://perishablepress.com/ultimate-htaccess-blacklist/

Understanding permissions on an Apache server
http://ag.arizona.edu/ecat/web/permissions.html

Universal www-Canonicalization via htaccess
http://perishablepress.com/universal-www-canonicalization-via-htaccess/

Use PHP to Create Symbolic Links without Shell Access
http://perishablepress.com/use-php-to-create-symbolic-links-without-shell-access/

Using Apache mod_expires to control browser caching
http://www.electrictoolbox.com/apache-mod-expires-browser-caching/

Using Cookies in PHP > Introduction to Cookies
http://www.phpnerds.com/article/using-cookies-in-php

Using cookies with jQuery
http://www.ilovecolors.com.ar/using-cookies-jquery/

Using .htaccess to redirect obsolete browsers
http://evansims.com/6032/using-htaccess-to-redirect-obsolete-browsers/

Using Permalinks
http://codex.wordpress.org/Using_Permalinks

Video Tutorial - Building Secure WP sites
http://www.lynda.com/tutorial/78547

Viewing hidden files on a Mac
http://guides.macrumors.com/Viewing_hidden_files_on_a_Mac

Web Performance Best Practices - Make the Web Faster
https://developers.google.com/speed/docs/best-practices/rules_intro

What characters are allowed unencoded in query strings?
http://www.456bereastreet.com/archive/201008/what_characters_are_allowed_unencoded_in_query_strings/

What is My WordPress Feed URL?
http://perishablepress.com/what-is-my-wordpress-feed-url/

What Is SEO / Search Engine Optimization?
http://searchengineland.com/guide/what-is-seo

What is SQL?
http://kb.iu.edu/data/ahux.html

Whitelist in .htaccess
http://www.it4y.info/apache/whitelist-in-htaccess.html

WHOIS Search for Domain Registration Information
http://www.networksolutions.com/whois/index.jsp

Why cheap web hosting is expensive
http://www.freelock.com/blog/john-locke/2010-03/why-cheap-web-hosting-expensive

WordPress Admin
http://htaccessbook.com/wp/wp-admin/

WordPress Boilerplate Theme
http://wordpress.org/extend/themes/boilerplate

WordPress Feedburner HTAccess Redirect for Default (Non-Permalink) Feed URLs
http://perishablepress.com/wordpress-feedburner-htaccess-redirect-default-feeds/

WordPress Plugin: Contact Coldform
http://perishablepress.com/contact-coldform/

Working with cookies using jQuery and JavaScript
http://web.enavu.com/tutorials/working-with-cookies-using-jquery-and-javascript/

WPO: Web Performance Optimization
http://www.stevesouders.com/blog/2010/05/07/wpo-web-performance-optimization/

For a complete list of URLs that haven't been shortened, visit the book's companion site:
http://htaccessbook.com/site-references/

> Jeff has long been the go-to guy for all things .htaccess. I always think of .htaccess as pretty much programming voodoo, where saying a chant as you save the file has about the same chance of working as any actual characters I type. But of course that isn't true, and I'm glad Jeff is here to help us through it with this book.
>
> — **Chris Coyier**
> **css-tricks.com**

> I'll probably never master regex or the mysterious ways of the .htaccess file but thanks to Jeff I am a few steps closer to being a little more in control of both! It's all in here, from redirects, to optimisation through to a dedicated section on using .htaccess to super charge your WordPress site. I can't recommend it enough — grab a copy now.
>
> — **Keir Whitaker**
> **keirwhitaker.com**

> If you ever wanted to finally fully understand .htaccess and all its peculiarities, this comprehensive, well-written book will be just what you are looking for.
>
> — **Vitaly Friedman**
> **smashingmagazine.com**

> In .htaccess made easy, Jeff Starr has carried out the difficult task of creating a guide that provides instruction for beginners while acting as a useful reference that advanced users can dip in and out of. This beautiful guide to .htaccess is delivered in Jeff's good-humoured and engaging style, making it readable, accessible, and a must-have for your web development library.
>
> — **Siobhan McKeown**
> **siobhanmckeown.com**

.htaccess made easy
htaccessbook.com
jeff starr | perishable press