WEB DEVELOPMENT

The Ultimate Guide to .htaccess Files

by Joseph Pecoraro 11 May 2009

131 Comments

Apache's .htaccess configuration files have baffled countless developers. This tutorial aims to break through this confusion by focusing on examples and thorough descriptions.

Among the benefits of learning .htaccess configuration is automatic gzipping of your content, providing friendlier URLs, preventing hotlinking, improving caching, and more. First, the basics.

Introduction:

I've read a number of .htaccess articles online. I'll shamelessly admit
I didn't get beyond the front page of Google results. I was shocked when
I actually read the articles and found that none of them explained what Apache was actually
doing. They were merely a collection of popular or useful tricks or
snippets of reusable code. That is all well and good, but the classic
argument is:

> *"Give a man a fish and he will eat for a day. Teach a man to fish and he*
> *will eat for a lifetime."*
> *- Confucius*

In this article I'm going to try to not just show you examples of useful

.htaccess directives, but explain exactly what is
going on. This way, you will understand the core principles and can then extend the examples or create new commands for your own use in whatever creative or useful ways you can come up with.

My focus will be on Apache 2, however much of this will apply
to Apache 1.3 and I'll try to point out any differences that I know of.

Finally, this tutorial will make the most sense if you read it in order.
I try to tie my examples together, and build off of them, in such a
way that you can try them yourself and follow along.

What is .htaccess?:

To quote Apache:

> *.htaccess files (or "distributed configuration files") provide a way to make*
> *configuration changes on a per-directory basis. A file, containing one or more*
> *configuration directives, is placed in a particular document directory, and the*

directives apply to that directory, and all subdirectories thereof.

Directives

“Directives” is the terminology that Apache uses for the commands in Apache’s configuration files. They are normally relatively short commands, typically key value pairs, that modify Apache’s behavior. **An .htaccess file allows developers to execute a bunch of these directives without requiring access to Apache’s core server configuration file, often named httpd.conf.** This file, httpd.conf, is typically referred to as the "global configuration file" and I will refer to it with that name or its short filename equivalent.

This feature is ideal for many hosting companies deploying a shared hosting environment. The hosting company will not allow its customers to access the global configuration file, which ultimately affects all of the customers hosted on that server. Instead, by enabling .htaccess, they give each of their customers the power to specify and execute their own Apache directives in their own directories and subdirectories. Of course it's also useful to the single developer, as you will see.

It's worth mentioning that anything that can be done with a .htaccess file can

be done in the httpd.conf file. However, *NOT* everything that can be done in httpd.conf can be done in a .htaccess file. In fact .htaccess files must be enabled in the httpd.conf file in order to be executed at all. Once enabled, their power can be limited to certain “contexts” so that they may be allowed to override some settings but not others. This gives the system administrators more control over what they let other developers get away with in their .htaccess files.

Enabling .htaccess:

.htaccess files are normally enabled by default. This is actually controlled by the AllowOverride Directive in the httpd.conf file. This directive can only be placed inside of a <Directory> section. Don’t let this confuse you. The typical httpd.conf file defines a DocumentRoot and the majority of the file will contain Directives inside a <Directory> section dealing with that directory. This includes the AllowOverride directive.

The default value is actually “All” and thus .htaccess files are enabled by default. An alternative value would be “None” which would mean that they are completely disabled. There are numerous other values that limit configuration of only certain contexts. Some are:

- AuthConfig - Authorization directives such as those dealing with Basic Authentication.
- FileInfo - Directives that deal with setting Headers, Error Documents, Cookies, URL Rewriting, and more.
- Indexes - Default directory listing customizations.
- Limit - Control access to pages in a number of different ways.
- Options - Similar access to Indexes but includes even more values such as ExecCGI, FollowSymLinks, Includes and more.

Full .htaccess Overriding

I'll show some examples, without their corresponding <Directory> sections.
Here is an example that allows full .htaccess overriding:

```
# Allow .htaccess files their full power
AllowOverride All
```

Limited Overriding

And here is an example that takes a more fine grained approach and only allows overriding of the Authorization and Indexes contexts but nothing else:

```
# Only allow .htaccess files to override Authorization and Indexes
```

```
2 AllowOverride AuthConfig Indexes
```

Comments

The first line in both of these examples are Apache comments. Comments start with the “#” symbol. This is common to many configuration files and scripting languages. I’ll have plenty of comments in my examples to help explain what things do. However, they are not required, and it's really just personal preference on how much you want to comment. Comments are not required.

The second line is the AllowOverride directive itself. This is the usual syntax of an Apache Directive. First there is the Directive name “AllowOverride” followed by a space separated list of values. Although this syntax looks rather loose; *always be careful*.

> *Sometimes even a single error in your httpd.conf or .htaccess file will result in a temporary meltdown of the server, and users will see 500 - Internal Server Error pages.*

For that reason alone, it's good practice to always make a backup of your httpd.conf and .htaccess files before you make a change or addition. This way, if anything goes wrong

with a modification, you will have
nothing to worry about, because you can revert to your previous working version. **I will also encourage you to make small changes at a time and verify that the changes work in
increments as opposed to making a number of changes at once.** This way, if you make
an error, it will be much easier to track down what may have caused it.

If you are ever confused on the syntax of any directive, immediately go to the Apache Directive listing and review the “Syntax” they have listed in the table for each individual directive. I will do my best to try and explain it here (I am trying to teach) but my explanation can never be as good as the formal technical documentation itself. Never be afraid of the documentation, it is your most reliable and trustworthy reference. I’ll try to make things more interesting here (woohoo!), but in the end, I’m just putting a different spin on those docs.

Checking if .htaccess is Enabled:

It's quite possible, in fact it's extremely likely, that your hosting company does not give you access to the httpd.conf file. So how do you know if .htaccess support is enabled or not? Don’t worry, .htaccess is a very common and useful feature that most companies will

have enabled, or will enable if you ask politely.

The best thing to do would be to just check with your hosting company. If it's not explicitly listed anywhere in your hosting plan, then shoot their support an email. This is a relatively common
question so they mostly likely already have a response ready for you. They will likely be willing
to enable the service or at least give a reason why they may not allow it.

In any event, you can always give it a shot and see if a simple .htaccess file works!
Included in this tutorial's sample download are two ways which you can check to see if .htaccess
support is enabled. The two folders are "is_htaccess_enabled" and "is_htaccess_enabled_2".
Give them a shot, I'll explain what each one is doing here.

is_htaccess_enabled

This test case is very simple. It uses a directive to make Apache look
first for the "index*good.html" file before "index.html." If .htaccess support is enabled, when you point your browser to the folder, Apache will load the .htaccess file and know that*

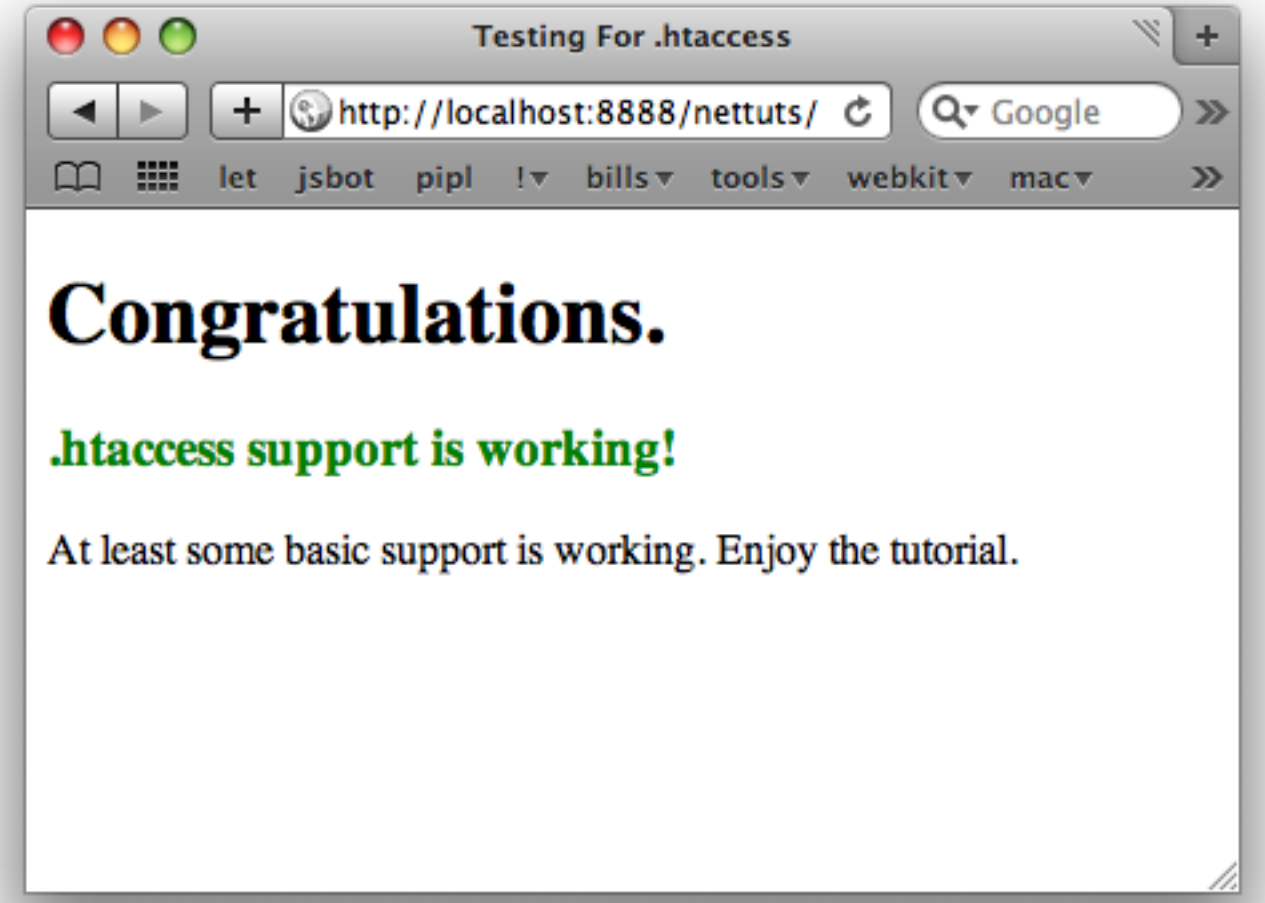
*it should display the "index*good.html" page containing a green message saying Congratulations!
If .htaccess support is not enabled then Apache will, by default, ignore the .htaccess file and immediately look for an index.html file.

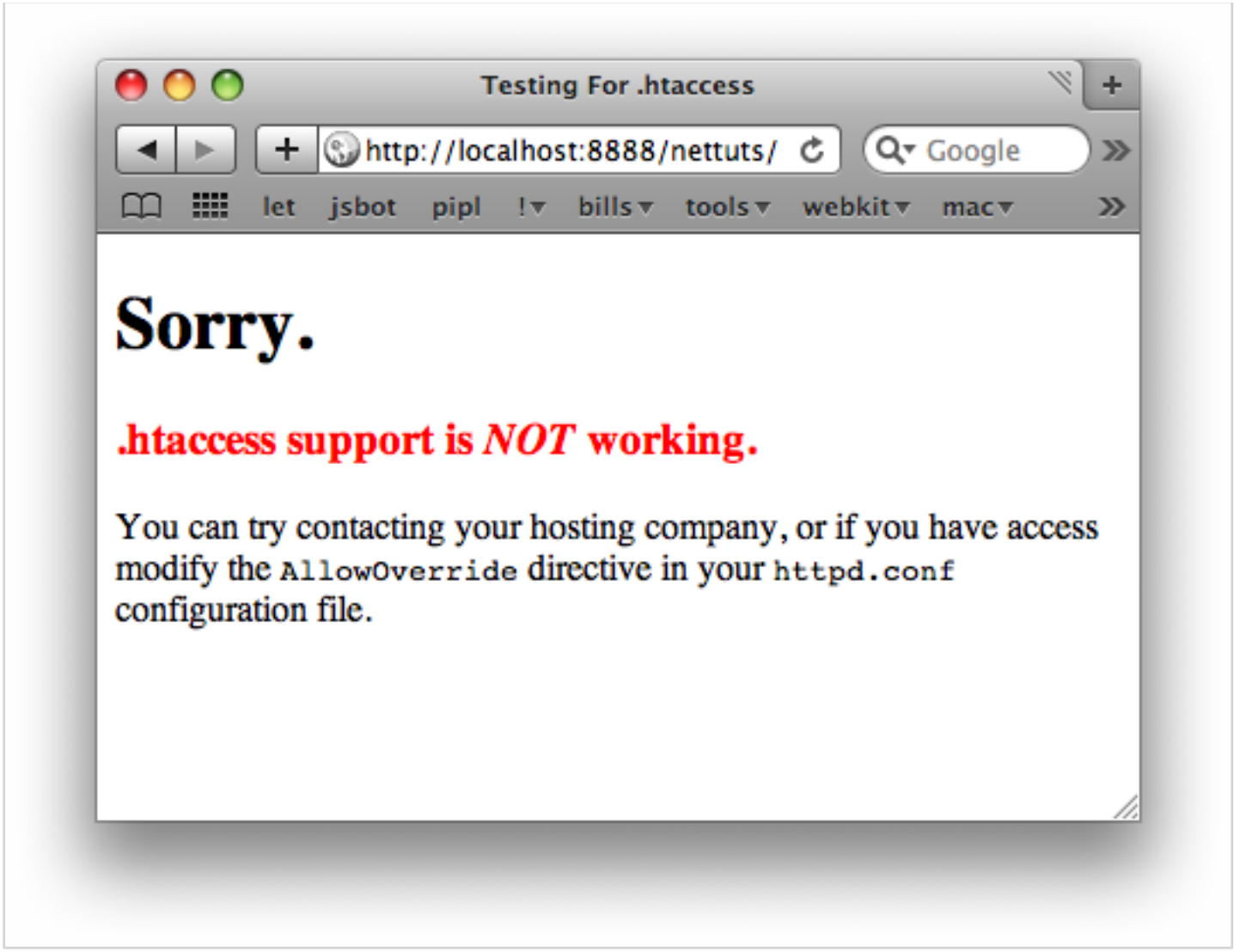
```
# This Directive will make Apache look first
# for "index_good.html" before looking for "index.html"
DirectoryIndex index_good.html index.html
```

DirectoryIndex

The DirectoryIndex directive takes a space separated list of potential filenames. When Apache is given a URL of a directory, and not a direct page (for example http://www.example.com and not http://www.example.com/index.html) Apache will use this list of files to search for the proper page to load. Apache will look for the files using the values in the list from left to right. The first file that Apache sees exists will be the file that it loads and displays to the client.

Using the above .htaccess file, here is an example of the good (enabled) and bad (disabled) cases:



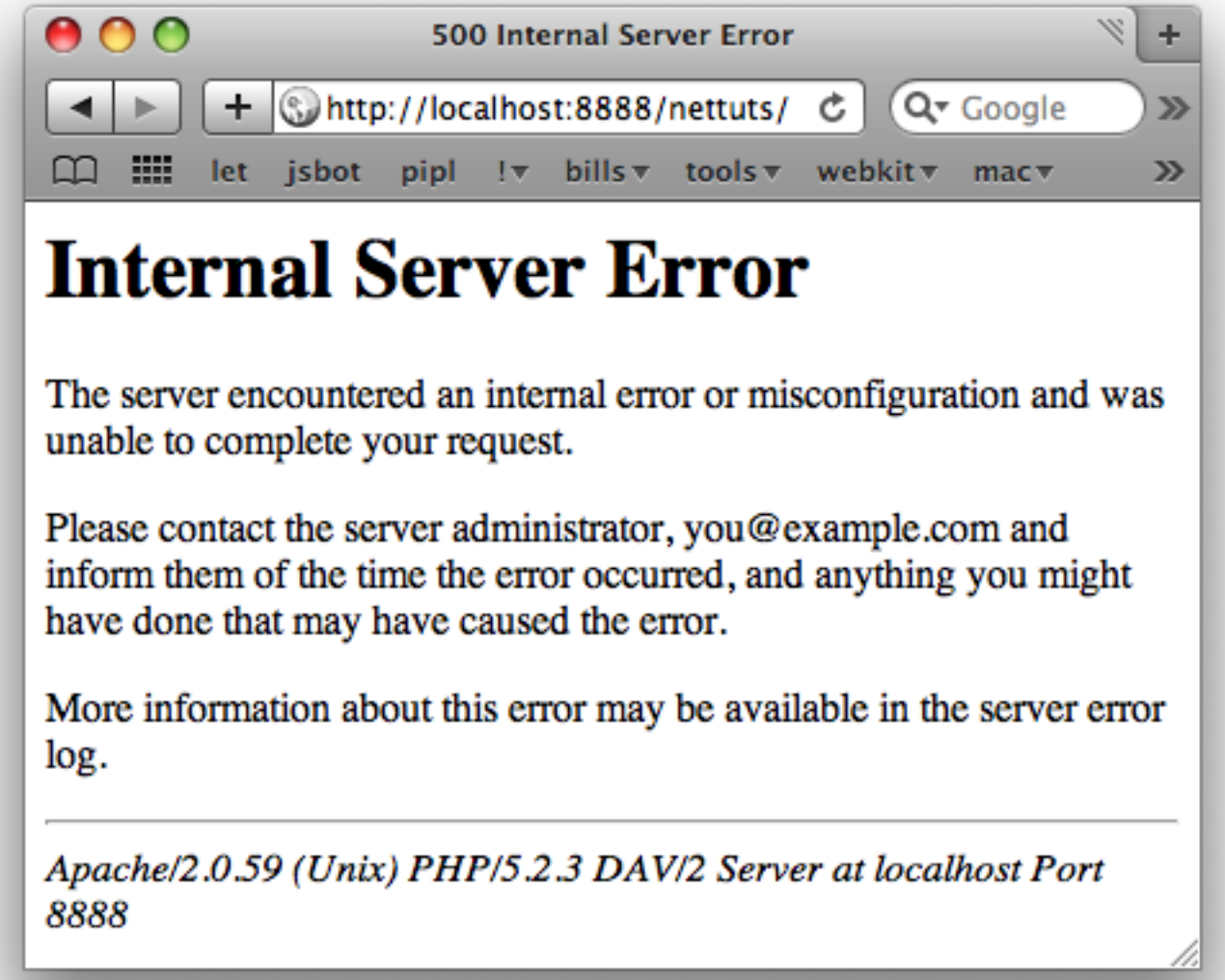


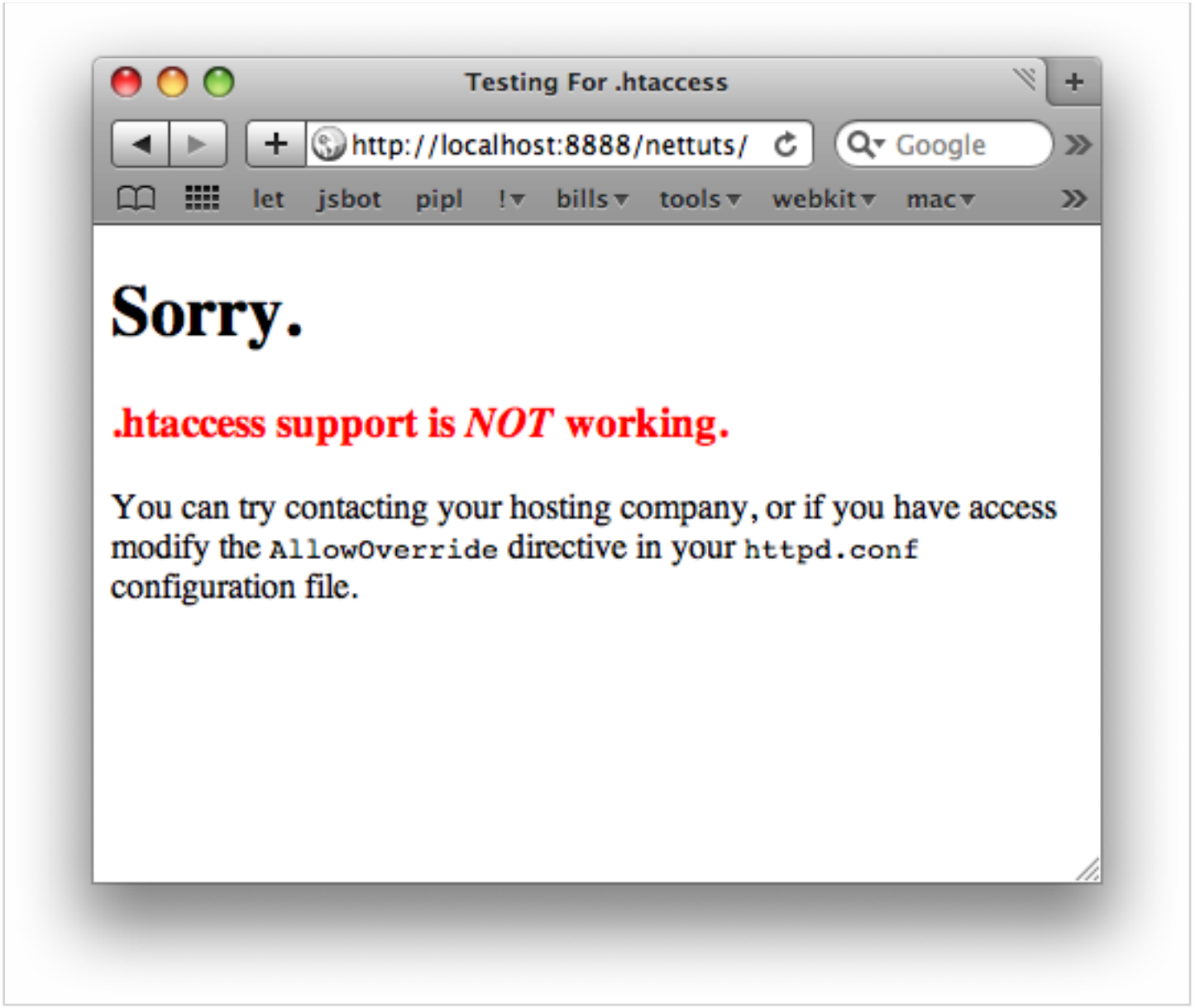
is_htaccess_enabled_2

As I said earlier, a syntax error in your .htaccess file will cause the server to hiccup. You can use this to your advantage to test if your server has .htaccess support enabled! Here is an example of an .htaccess file that is intended to blow up.

```
# This file is intended to make Apache blow up.  This will help
# determine if .htaccess is enabled or not!
AHHHHHHH
```

It's pretty clear that “AHHHHHHH” is not a valid Apache directive. This will cause an error if Apache tries to read the .htaccess file! So, if you get back a page yelling “Internal Server Error” then your server is looking for .htaccess files! If you actually see the contents of the index.html file, then it is likely that they have been disabled. Here again are the good and bad cases:





AccessFileName

Finally, it is still possible that .htaccess support is still enabled, just with

unique settings. Systems administrators can change the name of the .htaccess file just like we changed the name of the default file Apache looks for. This is possible by using the AccessFileName directive in the global configuration file. Again, the best thing to do in that case would be to contact your hosting company for more information.

Consequences of .htaccess files:

Before I get into some of the cool things you can do with .htaccess files, I have to tell you what you're getting into. As I mentioned previously you're allowing overriding server settings for a directory *and* all of its subdirectories. **Always keep in mind that you're affecting all of the subdirectories as well as the current directory.**

> *Also, when enabled the server will take a potential performance hit. The reason is because, every server request, if .htaccess support is enabled, when Apache goes to fetch the requested file for the client, it has to look for a .htaccess file in every single directory leading up to wherever the file is stored.*

This means a number of things. First because Apache always looks for the .htaccess files

on every request, any changes to the file will immediately take effect.
Apache does not cache them, and it will immediately see your changes on the next request.
However, this also means that Apache is going to have to do some extra work for every request.
For example, if a user requests /www/supercool/test/index.html, then your server would check for the following .htaccess files:

```
/www/.htaccess
/www/supercool/.htaccess
/www/supercool/test/.htaccess
```

These potential file accesses (potential because the files may not exist) and their execution (if they did exist) will take time. Again, my experience is that it's unnoticeable and it doesn't outweigh the benefits and flexibility that .htaccess files provide developers.

However, if this does concern you, as long as you have access to the httpd.conf file then you can always put your directives there. By setting AllowOverride to "None" Apache will not look for those .htaccess files. If you really want to, you can put the directives you wanted to put in your /www/supercool/test/.htaccess file directly in httpd.conf like so:

```
<Directory /www/supercool/test>
  # Put Directives Here
</Directory>
```

The disadvantage with this approach is that you will have to restart the Apache server on every change so that it reloads the new configuration.

In the end it comes down to personal preference or whatever your host allows. I prefer using .htaccess files because I have the flexibility to place them where I want, and their effects are live immediately without requiring a server reset.

Starting Simple - Directory Listing - Indexes:

Directory Listings

Before getting into any of the complex features, let's start with something simple, but useful, so that you can gain a feel for working with .htaccess files. Directory Listings are so common that you've probably come across them numerous times browsing the web.

When a user requests a directory, Apache first looks for the default file. Typically, it will be named "index.html" or "index.php" or something similar. When it doesn't find one of these files, it falls back on the mod_autoindex module to

display a listing of the files and folders in that directory. Sometimes this
is enabled, sometimes disabled, and sometimes you want to
make customizations. Well, with .htaccess you can easily manipulate these listings!

By default Directory listings are enabled. Here is an example scenario.
Suppose you have a bunch of media files that you're storing on your web server,
and you want to hide them from the public and search engines so that no one can
steal these files. That's very easy to do! Simply create a .htaccess file
in the directory that you want to hide and add the following directive:

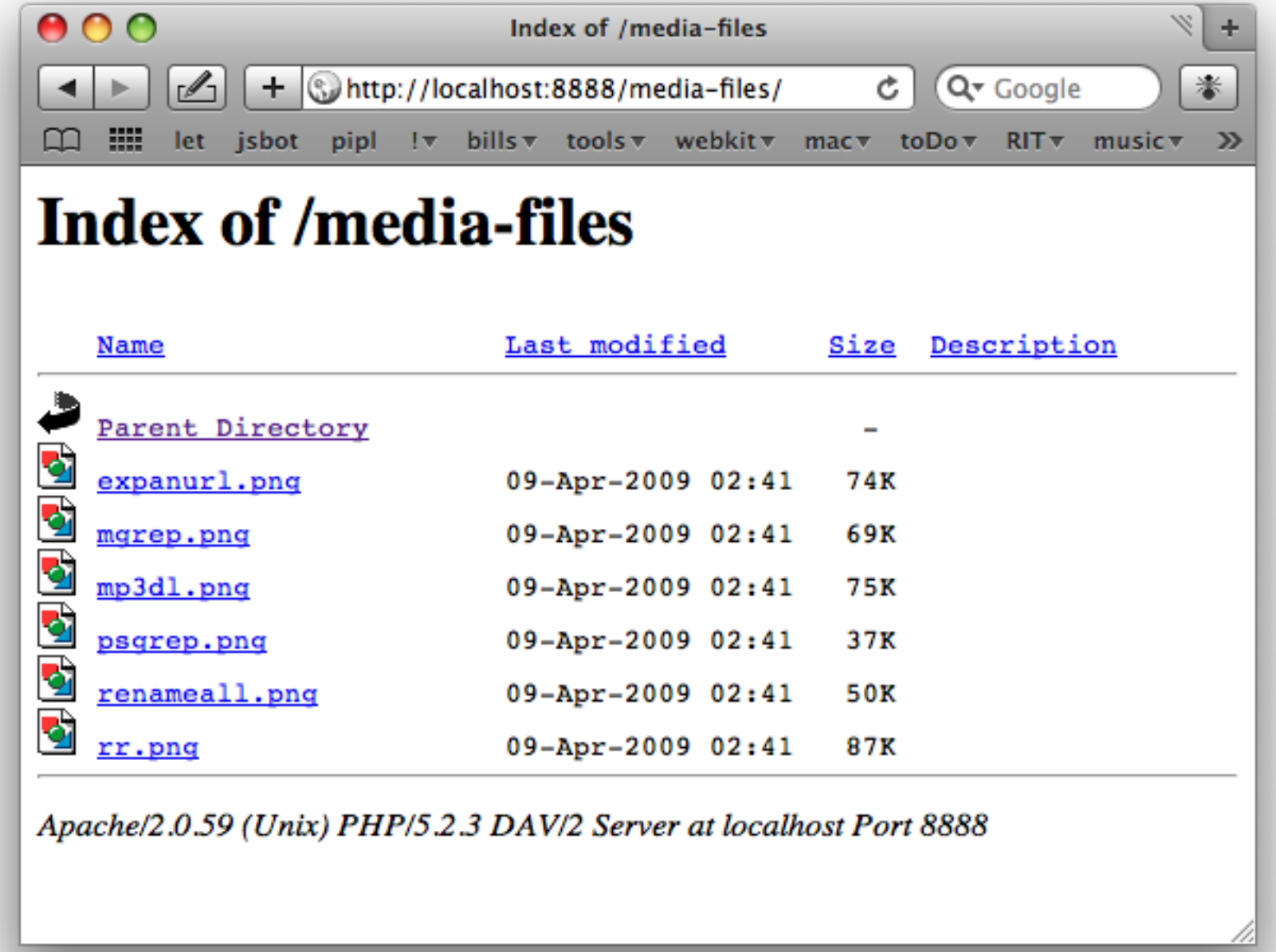
```
# Disable Directory Listings in this Directory and Subdirectories
# This will hide the files from the public unless they know direct URLs
Options -Indexes
```

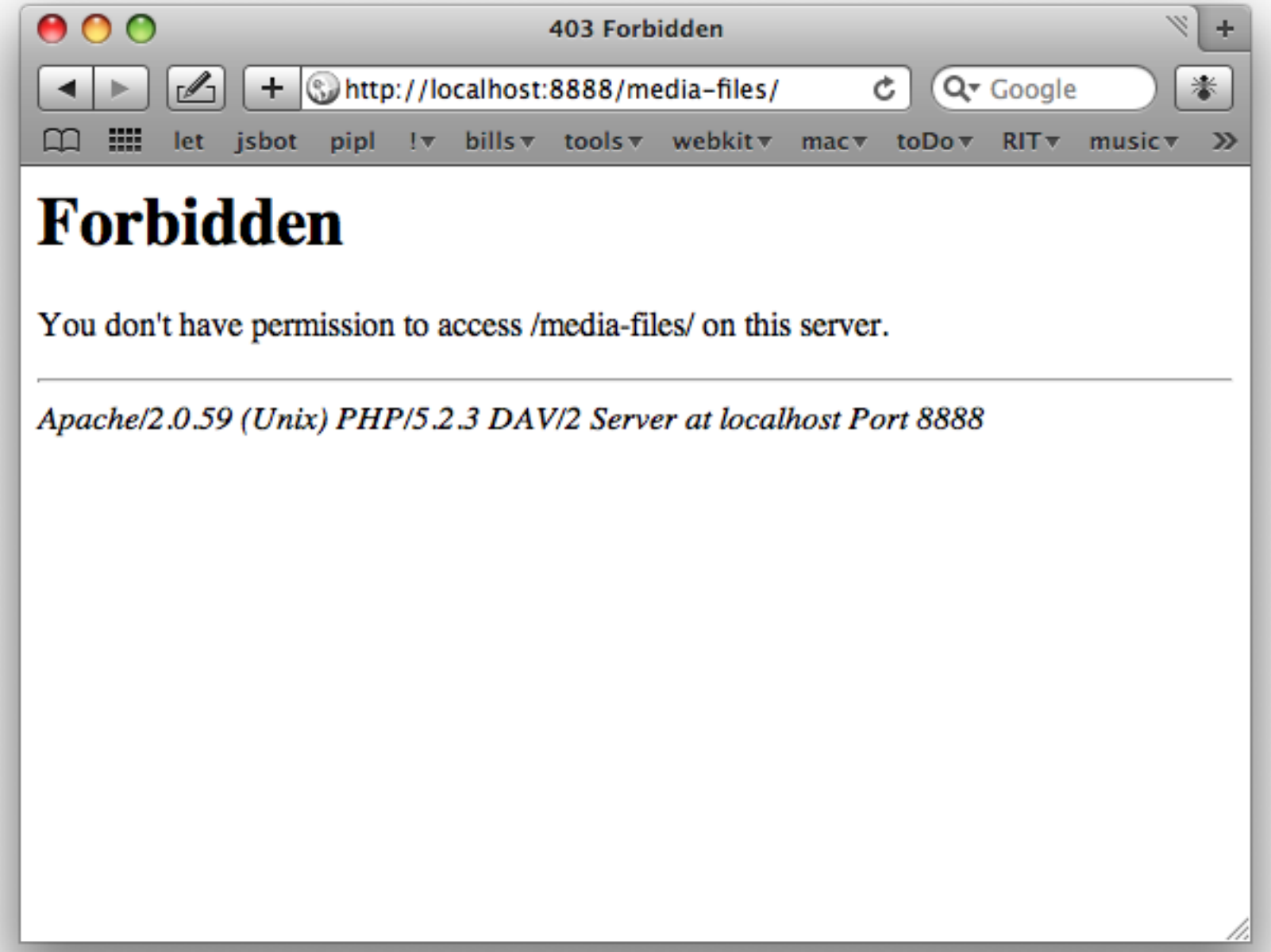
Options Directive

Breaking this down we are using the Options directive.
This directive can take a number of values (mentioned previously). If you provide the values
with a + or - like I did with -Indexes, then this will inherit the
Options that were enabled in higher directories and the global configuration!
If you don’t provide a + or - then the list that you provide will become

the *only* options enabled for that directory and its subdirectories. No other options will be enabled. Because you may not know which Options were enabled previously, you will most likely
use the + or - syntax unless you are absolutely sure you *only* want certain Options.

Now, with that directive in your .htaccess file, when you point your browser to that directory you will no longer be able to see the files. Here is the before and after:





Forge Ahead - Basic Authentication

Okay, maybe totally disabling the Directory Index is not what you want. It's more likely that you want to keep the Indexes but only allow certain people access. That is where Basic Authentication can be very useful. This is the most common type of Authentication on the web. When the user tries to access the page they will see the familiar Username/Password dialog. Only a user with the proper credentials will be able to access the contents.

For basic authentication there are just two steps.

1. Setup a file that stores usernames and password (encrypted).
2. Add a few lines to .htaccess to use that file.

Traditionally web developers have named the file that store the usernames and passwords “.htpasswd”. This is because the command line tool that ships with Apache that generates the proper encrypted username/password pair is actually called htpasswd! If you feel comfortable at the command line you can use the htpasswd tool, however there are plenty of online tools which will generate the output just as easily.

I created a sample .htpasswd file for a user “joe” with password “cool”.
I threw those values into the linked online tool and it produced:

```
joe:$apr1$QneYj/..$0G9cBfG2CdFGwia.AHFtR1
```

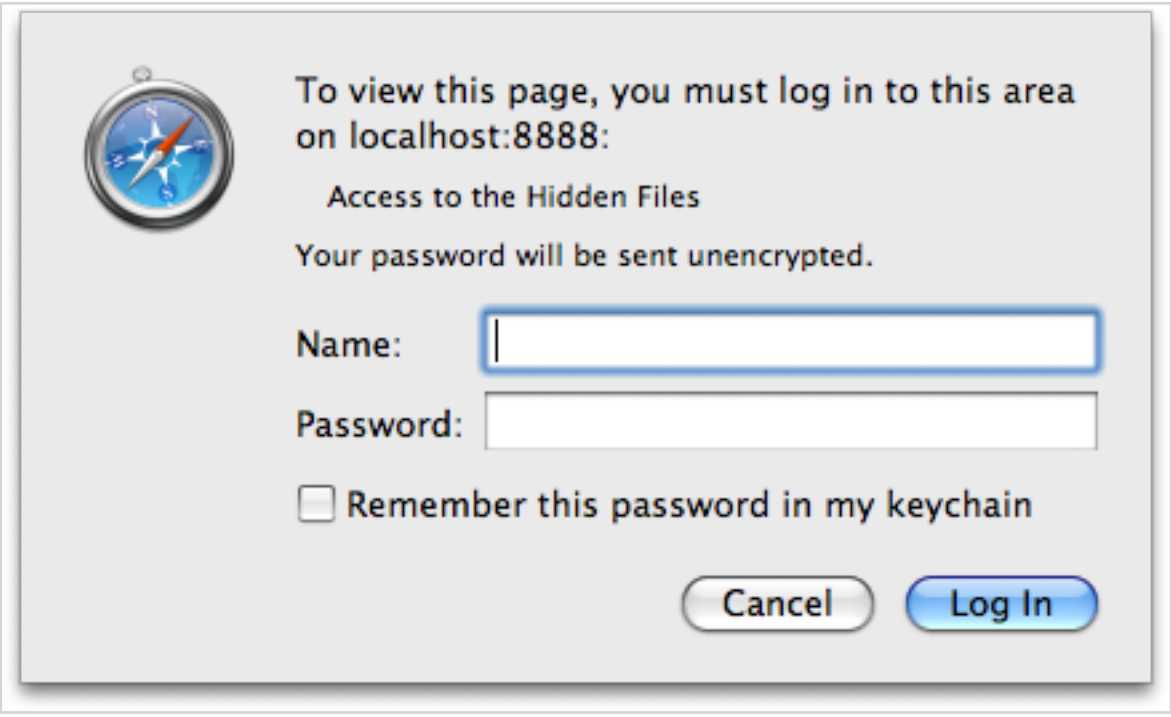
Your output might be different, that is okay. The passwords are hashed with a random salt to make them a bit more unique and secure. Once your username and password combination has been added to the .htpasswd file, then you should add the following lines to your file:

```
# Enable Basic Authentication
AuthType Basic

# This is what will be displayed to the user on the login dialog.
AuthName "Access to the Hidden Files"

# This you must edit.  It is the absolute path to the .htpasswd file.
AuthUserFile /path/to/.htpasswd

# This allows any user from inside the .htpasswd file to access the
# content if they provide the proper username and password.
Require valid-user
```

Those commands are well documented. The only real challenge is that you have to properly set the path to the .htpasswd file that you just generated. This is a full absolute path from the absolute root of the server. Also, because the .htpasswd file path is absolute, it's good practice to put it in a directory outside the directory where Apache

serves webpages to the public. That way malicious users won't be able to easily gain access to the raw listing of users/passwords stored in the .htpasswd.

Once it's all set up, when someone attempts to access the page they will receive the following dialog:



Basic Authentication is nice and easy, but it's not a total solution. Passwords are sent over the wire, Base 64 Encoded, in plain text. If you want more secure authentication you should couple Basic Authentication with https, a more secure protocol. That is a topic for another time.

Headers

The core protocol of the web is the Hypertext Transfer Protocol (HTTP). If you really want to understand what the rest of the Apache directives deal with, you should have some knowledge of the protocol. I'm only going to su pplya very quick summary here. I'll also make an effort to explain what the more complex directives are doing, but it will make more sense if you understand HTTP Headers.

The quick summary is that HTTP is stateless. With every request (from the browser)

and every response (from the Web Server like Apache) there are two sections.
A section of Header information, then an optional section containing the data itself,
if there is any data.

Request header information often specifies the file they are
requesting from the server (index.html), any state information they should provide
(such as cookie data), and the mime types it's willing to accept back from the server
(text/html or even gzip encoded content).

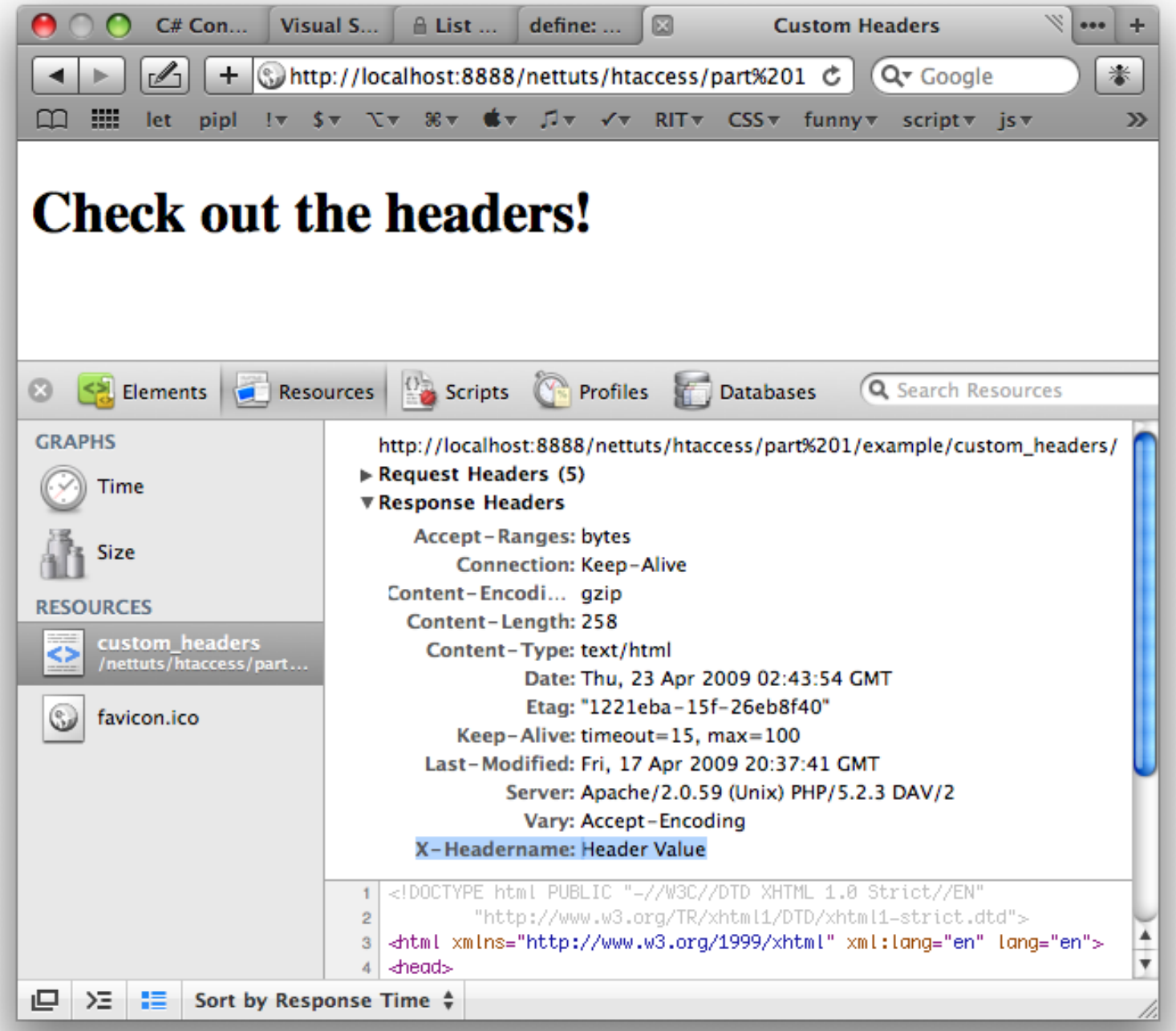
Response header information often specifies generic server information (Apache, PHP,
Perl versions etc.), the content encoding, length, mime/type, and more. There
are a plethora of HTTP headers to specify even more details like Cache Control,
Redirects, and Status Codes. Ever get a 404? That was a result of requesting a
file the server couldn't find, and thus it sent back a 404
Status Code in its
Response.

What does this have to do with .htaccess? Well, you can use Apache directives to
overwrite (set) or add new headers (add) which are sent back to the client
in the Response's Header section. Also, as you will see in later tutorials,
more advanced functionality such as URL Rewriting deals with the incoming headers.

Let's start simple, we'll add a header to the Response and see what happens:

```
# Add the following header to every response
Header add X-HeaderName"Header Value"
```

Requesting a file in the same directory as this .htaccess file shows the extra header:



You probably thought it was peculiar that I prefixed the custom header with "X-". This is actually a common convention that developers use to denote that the header is a non-standard header. This makes it really easy to realize that this header is custom. This convention is briefly mentioned here.

On a more comical note, some people have had a bit of fun with headers. This site points out some rather unusual headers found all over the web.

However, I really want to show you how to create Headers so that you can use them as a debugging technique. Just the other day, I ran a test to check to see if certain modules were enabled on a web-server. I wrote the following check:

```
<IfModule mod_gzip.c>
  Header add X-Enabled mod_gzip
</IfModule>
<IfModule mod_deflate.c>
  Header add X-Enabled mod_deflate
</IfModule>
```

When I made my next request with my browser and checked the Response Headers, it

showed that neither of the modules were turned on! I contacted my hosting company, and they agreed to enable gzip compression!

There is a difference between Header set and
Header add. With add, the Header will *always* get added
to the Response. Even if it happens to show up multiple times in the response. This is most often
what you would want for custom headers. You would use set when you want to override the value of one of the default headers that Apache returns. An example would be overriding the mime/type specified by the Content-Type header for a certain file. Apache would set the internal value and then use that when it prints out the default header. There will be no duplicates, and thus no possibility for interpreting an error or confusion by the client. (In case you were wondering, the HTTP specification states that, in the case of duplicates, the client should always use the last value specified for that duplicate header.)

Conclusion:

I've gone over some basic Apache directives in quite a bit of detail. I wanted to get the fundamental details out of the way so that the next tutorial may discuss cooler things. My next article will

focus on some of the more useful features you can enable with .htaccess. These topics will include:

- GZip encoding of content for both Apache 1.3 and Apache 2
- A through description of mod_rewrite and plenty of examples that are dissected and explained in detail.

Follow us on Twitter, or subscribe to the NETTUTS RSS Feed for more daily web development tuts and articles.

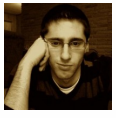
Difficulty:

Intermediate

Tagged with:

Web Development **.htaccess**

About Joseph Pecoraro



My name is Joseph Pecoraro. I'm a web developer and designer from western New York. I am presently attending the great Rochester Institute of Technology to earn my MS in Computer Science by the end of 2009.

Related Posts

Code

Code

Code

Code

Rapid Website Deployment With Django, Heroku & New Relic

9 days ago

Optimizing WordPress with Varnish and W3 Total Cache

18 days ago

Installing WordPress in the Amazon Cloud

25 days ago

How to Use New Relic With PHP & WordPress

14 Apr 2014

131 Comments Nettuts+

Sort by Best

Share

Join the discussion…

Barry McGee • 5 years ago

I look forward to the next tutorial to learn more about mod_rewrite, thanks!

2 • Reply • Share ›

ahmed • 10 months ago

great job

1 • Reply • Share ›

Wayne Shears • 5 years ago

Awsome tut. This is the best insight I have had in to .htaccess files, I now reliase some mistakes in my

Best Tut on here for a while.

1 ^ | v • Reply • Share ›

Reyaz • 19 days ago

Really awesome way for the to get through the htaccess. You Really Teach man to Fish.

Keep it Up. i like it.

^ | v • Reply • Share ›

r0mel • 4 months ago

your tutorial is really great.. looking forward for another tutorial

^ | v • Reply • Share ›

mani • 7 months ago

awesome tutorial

^ | v • Reply • Share ›

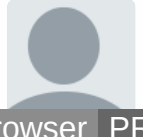
JuicePixel • a year ago

Hello,

Easy and explanatory as always.

Thanks

^ | v • Reply • Share ›



Dean Lynn • a year ago

Awesome tutorial! Really helpful.

Thanks!

Reply • Share ›

OJ • a year ago

Very good tutorial... Thank you

Reply • Share ›

Julien • 2 years ago

Hi

thanks for this good tutorial

anyway I can't find an answer to my problem

It seems directory listing is denied by my host (OVH). I can't access to the httpd.conf too...

Can I authorize a specific folder's directory listing with an htaccess ?

I put in this folder a "Options +Indexes" but I get an Error 500

Thank you !
have a nice day

Reply • Share ›

Nitin GUpta • 2 years ago

Htaccess, is a way you can redirect pages internally and user will not know from where u r getting the p another way this is a mechanism to redner a script.

Reply • Share ›

Chuck • 2 years ago

"Catch a man a fish, and you can sell it to him. Teach a man to fish, and you ruin a wonderful business
Karl Marx

• Reply • Share ›

Anooj • 2 years ago

Thank You Very Much, its realy helpful.

• Reply • Share ›

Joe • 2 years ago

Really helpful and comprehensive tutorial. But i was wondering if there is a way to prevent direct acces
the browser. For instance, i am making an alcohol site and i have a age gateway called default.php. On
users get to verify that they are of the legal age and then upon submitting the form, it calls index.php.

I want to be able to prevent direct access via the browser of index.php e.g. mysite.com/index.php but al
have on default.php to call it. Is this possible?

Your help will be much appreciated, thanx.

• Reply • Share ›

Akhenaton_2 • 3 years ago

Thanks toyou, Apache and htaccess files are somewhat less misty for me now. Great work !

• Reply • Share ›

josemaria • 3 years ago

Thanks for your effort. Here is another interesting and clear tut: http://corz.org/serv/tricks/ht...

• Reply • Share ›

Hemant • 3 years ago

Very Nice Tute Thanks a lot

• Reply • Share ›

Mariagrafs • 4 years ago

I had a big confusion before which was cleared by this fantastic article. I have saved the entire content f

• Reply • Share ›

Musta • 4 years ago

Hi, I have followed dozens of tutorials to enable the .htaccess file in my Macbook Pro i tried all of them b have really spent more than five days trying to make it work but I do not know what I am doing wrong. I h following problem. Well knowing that mac comes with a built in Apache server I changed the AllowOver AllowOverride All in both the private/etc/httpd/httpd.conf and in MAMP/conf/apache/httpd.conf . In the sa have uncomented the following line: AccessFileName .htaccess.

By the way my files reside in the web root MAMP/htdocs/testsfolder/ .I did all these changes but I still co it work even trying with your demo files (is_htaccess_enabled) wich always gets an error message.

I would be so thankful if you could give me a solution to this problem!

• Reply • Share ›

joke ➦ Musta • 4 years ago

you talking very PollofShit

• Reply • Share ›

Zac • 4 years ago

This is great and all, but why am i the only one getting an internal error msg when I add the header? All t

comments and Im the only one with the problem?

The checks worked out so Im assuming I have everything enabled. When adding the header I get the sa
get with the "AHHHHHHHHHHHHHHHHHH" check.

Reply • Share ›

Zac Zac • 4 years ago

Ah i figured it out. Im running wamp and had to check header_modules in the modules flyout.

Reply • Share ›

argehaber • 4 years ago

some sentences, but in general did not exactly turn a nice and helpful article. Thank you and good work

Reply • Share ›

Dave • 4 years ago

This is hardly an 'ultimate' guide. More of an introduction to..

Reply • Share ›

Alastair Hodgson • 4 years ago

Every .htaccess guide I've found is pretty rubbish, you've managed to explain everything that I need to k
understandable way, so cheers!

Reply • Share ›

Krimo • 5 years ago

Thanks for the article, awesomely helpful! One thing though, everytime I try to set up gzip to compress
on my page, either nothing happens (AddOutputFilterByType DEFLATE text/html text/plain text/xml) or I
error if I use FilesMatch....Anyone can help? Thanks!

mrak911 • 5 years ago

Great article.God job

Reply • Share ›

ilus • 5 years ago

THANKS MAN! I finally made it to protect a folder with htaccess.

Thanks!

Reply • Share ›

Gareth James • 5 years ago

Great tutorial - but its gonna take a few reads to understand it as a non -programmer

Reply • Share ›

เพชร • 5 years ago

well done, dude.

Reply • Share ›

Andy • 5 years ago

I am pleased this has helped.

Reply • Share ›

rodel • 5 years ago

this is a great tutorial

Reply • Share ›

Søge Maskineoptimering • 5 years ago

Great article. Htaccess is great for solving those annoying problems we meet every day

• Reply • Share ›

Lester • 5 years ago

Hi. I have a hidden directory which I've password protected. This .htaccess file works fine:
AuthName "Restricted Area"
AuthType Basic
AuthUserFile /XXX/YYY
AuthGroupFile /dev/null
require valid-user

However, some of these sub-directories and files are "hidden" -- start with "," and I cannot see them? W
to the .htaccess file above to permit this once I can login?

Thanks.

Lester

• Reply • Share ›

Lester ➚ Lester • 5 years ago

P.S. I have a typo in this posting: "," -> "."

Lester

• Reply • Share ›

url directory • 5 years ago

Wowwww....this is cool dude i never knew such things thanks buddy keep up ur gud work

Reply • Share ›

hedi • 5 years ago

Thankssss Youuu Soooooo muchhhhhhhhhhhhh

It Workkkkkkkkkkkkkkkkkk

Reply • Share ›

David • 5 years ago

htaccess files are as powerful as to tackle with. thanks for that very nice tut (images, coddes, everythin here !)

Reply • Share ›

B A B U • 5 years ago

Eagerly waiting for ur “friendly URLs” using PHP
Tanx

Reply • Share ›

flashfs • 5 years ago

Very cool. When you write about mod_rewrite, don't forget these plenty examples!

Reply • Share ›

Johan Steen • 5 years ago

Awesome! Great tutorial, it's really appreciated. I'm really looking forward to part 2. Keep up the great w

Reply • Share ›

John • 5 years ago

ideas?

Reply • Share ›

WallpaperDude • 5 years ago

Wonder why people who use CMS frameworks are so worried about using their own files, scripts or co making URLs friendly. Most CMS frameworks have lots of built in stuff just to do clean, friendly URLs. H one of the things Drupal is all about.

Reply • Share ›

Sathish • 5 years ago

wow... whenever I want to know of something when I falling asleep... I get it right away :D

thanks for this detailed tut

Reply • Share ›

NetHawk • 5 years ago

One question regarding the .htaccess in directories/subdirectories: does Apache stop looking for .htacc once it found one in a subdirectory? Or will it go up the entire folder path anyway?

My guess is, that it has to check on every directory level, because the final directives could consist of pa files on all the levels, if they complement each other.

Reply • Share ›

Joseph Pecoraro ↗ NetHawk • 5 years ago

Apache will check .htaccess in every subdirectory, starting from the root until it finds the file bein (then it won't go any deeper). Each level deeper inherits the properties from the previous levels a directives defined at this level will override any previous directives.

Roger Pilon • 5 years ago

I am using wordpress but would like to override it just for the error messages.

#guardian# Added by Guardian auto-config
#guardian# These lines should invoke a custom error handler
#guardian# added 2009-05-17 09:07:14
ErrorDocument 401 /guardian/ag.pl
ErrorDocument 403 /guardian/ag.pl
ErrorDocument 404 /guardian/ag.pl
ErrorDocument 500 /guardian/ag.pl
#guardian# / end Guardian stuff

AddType application/x-httpd-php5 .html .php
Use PHP5 as default
AddHandler application/x-httpd-php5 .html .php

BEGIN WordPress

RewriteEngine On
RewriteBase /

see more

Markus • 5 years ago

Good tutorial.. But I know the most things because I read all the stuff I could find about .htaccess...

Only one thing I stuck really, is how I could get my mod_vhost_alias configuration - something like:

VirtualDocumentRoot /home/%-2.0.%-1.0/web/%-3+
VirtualScriptAlias /home/%-2.0.%-1.0/cgi-bin

...to work with it. I tried all stuff but it do not work!

Somebody got a working example of a pure mod_vhost_alias configuration with .htaccess?

Would be great and help myself a lot!

  • Reply • Share ›

odel • 5 years ago

great tut!

looking forward to the part two of this tutorial on mod_rewrite.

thanks!

  • Reply • Share ›

rgt • 5 years ago

@Joseph Pecoraro - what browser are you using? I don't recognize it in the screenshots.....

  • Reply • Share ›

Load more comments

tuts+

Teaching skills to millions worldwide.

- Tutorials
- Courses
- eBooks
- Jobs
- Blog

Follow Us

- Subscribe to Blog
- Follow us on Twitter
- Be a fan on Facebook
- Circle us on Google+
- Tutorials
- Courses
- eBooks

Add more features to your website such as user profiles, payment gateways, image galleries and more.

Browse WordPress Plugins

Microlancer is now Envato Studio! Custom digital services likelogo design, WordPress installaton, video production and more.

Check out Envato Studio

- About
- FAQ
- Advertise
- Blog
- Support
- Privacy Policy
- Pricing
- Write For Us
- Terms of Use

envato