OS X 10.10 Yosemite Local Development Environment: Apache, PHP, and MySQL with Homebrew

Dec 22, 2014

OS X 10.10 Yosemite comes with Apache and PHP pre-installed, but it's not in a great configuration, requires root to make lots of changes, and can introduce issues with file ownership and permissions. We prefer to use Homebrew to avoid these problems and because it's easier to keep up to date with newer versions of each component and extend customization. We can also set things up to be fully automatic so you can create new websites locally without needing to edit any configuration files.

With the arrival of Yosemite, some of the changes previously used in 10.9 for setting up Apache, PHP, and MySQL with Homebrew[1] don't work quite the same. This guide will walk you through using Homebrew[2] to install Apache, PHP, and MySQL for a "MAMP" development environment. We'll also use DNSMasq and Apache's VirtualDocumentRoot to set up "auto-VirtualHosts" so you don't need to edit configuration files when starting new projects. Finally, we'll add a firewall rule to allow the default http port 80 to be used without running Apache as root.

The following steps are intended for use on a Yosemite system without any previous attempts to use Homebrew for Apache, PHP, or MySQL. If you have attempted to install a similar stack and run into conflicts, or you've upgraded your operating system from 10.9 and things broke, the final section has some troubleshooting pointers. If that fails, leave a comment and I'll try my best to help you out.

At the conclusion of this guide, you'll be able to create a directory like *~/Sites /project* and access it immediately at *http://project.dev* without editing your */etc/hosts* file or editing any Apache configuration. We'll configure PHP and MySQL to allow for enough flexibility for development.

Because some of the commands span several lines, each command will be in a separate code block. This means you should copy and paste each code block in its entirety as a single command.

Before diving in, yes, this is a lot of steps. You can do it faster and pay money for something like MAMP Pro, but this is more fun, and you may learn something along the way! And, while you can simplify things with Vagrant[3] or other virtual machine format, some people prefer to run things on "bare metal" and not have the overhead of a virtual machine.

Homebrew Setup

If you've not already installed Homebrew, you can follow the instructions at http://brew.sh[2]. I used to include the command in previous walkthrough blogs, but it could change after posting, so definitely check their website to install it properly.

If you do not have git available on your system, either from Homebrew, Xcode, or another source, you can install it with Homebrew now (if you already have it installed, feel free to skip this step to keep the version of git you already have):

```
brew install -v git
```

PATH Variable

In previous guides on 10.9 and earlier, I added a change to $PATH in *~/.bash_profile* to ensure that Homebrew-installed applications would run by default over similar ones that were already installed on OS X. Thankfully, Yosemite's $PATH order is different than earlier OS versions and now includes the default Homebrew location of */usr/local/bin* in front. If you installed Homebrew to a custom location, or are not seeing */usr/local/bin* at the beginning of your shell's $PATH, check out the file */etc/paths* or the directory */etc/paths.d/*.

MySQL

Install MySQL with Homebrew:

```
brew install -v mysql
```

Copy the default *my-default.cnf* file to the MySQL Homebrew Cellar directory where it will be loaded on application start:

```
cp -v $(brew --prefix mysql)/support-files/my-default.cnf $(brew --
```

This will configure MySQL to allow for the maximum packet size, only appropriate for a local or development server. Also, we'll keep each InnoDB table in separate files to keep *ibdataN*-type file sizes low and make file-based backups, like Time Machine, easier to manage multiple small files instead of a few large InnoDB data files. This is the first of many multi-line single commands. The following is a single, multi-line command; copy and paste the entire block at once:

```
cat >> $(brew --prefix)/etc/my.cnf <<'EOF'

# Echo & Co. changes
max_allowed_packet = 1073741824
innodb_file_per_table = 1
EOF
```

Uncomment the sample option for innodb_buffer_pool_size to improve performance:

```
sed -i '' 's/^#[[:space:]]*\(innodb_buffer_pool_size\)/\1/' $(brew
```

Now we need to start MySQL using OS X's launchd. This used to be an involved process with launchctl commands, but now we can leverage the excellent brew services[4] command:

```
brew tap homebrew/services
```

```
brew services start mysql
```

By default, MySQL's root user has an empty password from any connection. You are advised to run mysql_secure_installation and at least set a password for the root user:

```
$(brew --prefix mysql)/bin/mysql_secure_installation
```

Apache

Start by stopping the built-in Apache, if it's running, and prevent it from starting on boot. This is one of very few times you'll need to use sudo:

```
sudo launchctl unload /System/Library/LaunchDaemons/org.apache.http
```

The formula for building Apache is not in the default Homebrew repository that you get by installing Homebrew. While we can use the format of brew install external-repo/formula, if an external formula relies on another external formula, you have to use the brew tap command first. I know, it's weird. So, we need to tap homebrew-dupes because "homebrew-apache/httpd22" relies on "homebrew-dupes/zlib". Whew:

```
brew tap homebrew/dupes
```

A slight deviation from my prior walkthroughs: we'll install Apache 2.2 with the event MPM and set up PHP-FPM instead of mod_php. If those terms mean anything to you and you're curious as to why I decided to go this route; it's because: 1) switching PHP versions is far easier with PHP-FPM and the default 9000 port instead of also editing the Apache configuration to switch the mod_php module location, and 2) if we're therefore not using mod_php, we don't have to use the prefork MPM and can get better performance with event or worker. As to why I'm using 2.2 instead of 2.4, popular FOSS projects like Drupal and WordPress still ship with 2.2-style .htaccess files. Using 2.4 sometimes means you have to set up "compat" modules, and that's above the requirement for a local environment, in my opinion.

Onward! Let's install Apache 2.2 with the event MPM, and we'll use Homebrew's OpenSSL library since it's more up-to-date than OS X's:

```
brew install -v homebrew/apache/httpd22 --with-brewed-openssl --wit
```

In order to get Apache and PHP to communicate via PHP-FPM, we'll install the mod_fastcgi module:

```
brew install -v homebrew/apache/mod_fastcgi --with-brewed-httpd22
```

To prevent any potential problems with previous mod_fastcgi setups, let's remove all references to the mod_fastcgi module (we'll re-add the new version later):

```
sed -i '' '/fastcgi_module/d' $(brew --prefix)/etc/apache2/2.2/http
```

Add the logic for Apache to send PHP to PHP-FPM with mod_fastcgi, and reference that we'll want to use the file *~/Sites/httpd-vhosts.conf* to configure our VirtualHosts. The parenthesis are used to run the command in a subprocess[5], so that the exported variables don't persist in your terminal session afterwards. Also, you'll see export USERHOME a few times in this guide; I look up the full path for your user home directory from the operating system wherever a full path is needed in a configuration file and "~" or a literal "*$HOME*" would not work.

This is all one command, so copy and paste the entire code block at once:

```
(export USERHOME=$(dscl . -read /Users/`whoami` NFSHomeDirectory |

# Echo & Co. changes

# Load PHP-FPM via mod_fastcgi
LoadModule fastcgi_module    ${MODFASTCGIPREFIX}/libexec/mod_fastcg

<IfModule fastcgi_module>
  FastCgiConfig -maxClassProcesses 1 -idle-timeout 1500

  # Prevent accessing FastCGI alias paths directly
  <LocationMatch "^/fastcgi">
    <IfModule mod_authz_core.c>
      Require env REDIRECT_STATUS
    </IfModule>
    <IfModule !mod_authz_core.c>
      Order Deny,Allow
      Deny from All
      Allow from env=REDIRECT_STATUS
    </IfModule>
  </LocationMatch>

  FastCgiExternalServer /php-fpm -host 127.0.0.1:9000 -pass-header
  ScriptAlias /fastcgiphp /php-fpm
  Action php-fastcgi /fastcgiphp

  # Send PHP extensions to PHP-FPM
  AddHandler php-fastcgi .php

  # PHP options
  AddType text/html .php
  AddType application/x-httpd-php .php
  DirectoryIndex index.php index.html
</IfModule>

# Include our VirtualHosts
Include ${USERHOME}/Sites/httpd-vhosts.conf
EOF
)
```

We'll be using the file *~/Sites/httpd-vhosts.conf* to configure our VirtualHosts, but the *~/Sites* folder doesn't exist by default in newer versions of OS X. We'll also create folders for logs and SSL files:

```
mkdir -pv ~/Sites/{logs,ssl}
```

Let's populate the *~/Sites/httpd-vhosts.conf* file. The biggest difference from my previous guides are that you'll see the port numbers are 8080/8443 instead of 80/443. OS X 10.9 and earlier had the ipfw firewall which allowed for port redirecting, so we would send port 80 traffic "directly" to our Apache. But ipfw is now removed and replaced by pf which "forwards" traffic to another port. We'll get to that later, but know that "8080" and "8443" are not typos but are acceptable because of later port forwarding. Also, I've now added a basic SSL configuration (though you'll need to acknowledge warnings in your browser about self-signed certificates):

```
touch ~/Sites/httpd-vhosts.conf
```

```
(export USERHOME=$(dscl . -read /Users/`whoami` NFSHomeDirectory |
#
# Listening ports.
#
#Listen 8080  # defined in main httpd.conf
Listen 8443

#
# Use name-based virtual hosting.
#
NameVirtualHost *:8080
NameVirtualHost *:8443

#
# Set up permissions for VirtualHosts in ~/Sites
#
<Directory "${USERHOME}/Sites">
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    <IfModule mod_authz_core.c>
        Require all granted
    </IfModule>
    <IfModule !mod_authz_core.c>
        Order allow,deny
        Allow from all
    </IfModule>
</Directory>

# For http://localhost in the users' Sites folder
<VirtualHost _default_:8080>
    ServerName localhost
    DocumentRoot "${USERHOME}/Sites"
</VirtualHost>
<VirtualHost _default_:8443>
    ServerName localhost
    Include "${USERHOME}/Sites/ssl/ssl-shared-cert.inc"
    DocumentRoot "${USERHOME}/Sites"
</VirtualHost>

#
# VirtualHosts
#

## Manual VirtualHost template for HTTP and HTTPS
#<VirtualHost *:8080>
#  ServerName project.dev
#  CustomLog "${USERHOME}/Sites/logs/project.dev-access_log" combin
#  ErrorLog "${USERHOME}/Sites/logs/project.dev-error_log"
#  DocumentRoot "${USERHOME}/Sites/project.dev"
#</VirtualHost>
#<VirtualHost *:8443>
#  ServerName project.dev
#  Include "${USERHOME}/Sites/ssl/ssl-shared-cert.inc"
#  CustomLog "${USERHOME}/Sites/logs/project.dev-access_log" combin
#  ErrorLog "${USERHOME}/Sites/logs/project.dev-error_log"
#  DocumentRoot "${USERHOME}/Sites/project.dev"
#</VirtualHost>

#
# Automatic VirtualHosts
#
# A directory at ${USERHOME}/Sites/webroot can be accessed at http:
# In Drupal, uncomment the line with: RewriteBase /
#

# This log format will display the per-virtual-host as the first fi
LogFormat "%V %h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Ag

# Auto-VirtualHosts with .dev
<VirtualHost *:8080>
  ServerName dev
  ServerAlias *.dev

  CustomLog "${USERHOME}/Sites/logs/dev-access_log" combinedmassvho
  ErrorLog "${USERHOME}/Sites/logs/dev-error_log"

  VirtualDocumentRoot ${USERHOME}/Sites/%-2+
</VirtualHost>
<VirtualHost *:8443>
  ServerName dev
  ServerAlias *.dev
  Include "${USERHOME}/Sites/ssl/ssl-shared-cert.inc"

  CustomLog "${USERHOME}/Sites/logs/dev-access_log" combinedmassvho
  ErrorLog "${USERHOME}/Sites/logs/dev-error_log"

  VirtualDocumentRoot ${USERHOME}/Sites/%-2+
</VirtualHost>
EOF
)
```

You may have noticed that *~/Sites/ssl/ssl-shared-cert.inc* is included multiple times; create that file and the SSL files it needs:

```
(export USERHOME=$(dscl . -read /Users/`whoami` NFSHomeDirectory |
SSLEngine On
SSLProtocol all -SSLv2 -SSLv3
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW
SSLCertificateFile "${USERHOME}/Sites/ssl/selfsigned.crt"
SSLCertificateKeyFile "${USERHOME}/Sites/ssl/private.key"
EOF
)
```

```
openssl req \
  -new \
  -newkey rsa:2048 \
  -days 3650 \
  -nodes \
  -x509 \
  -subj "/C=US/ST=State/L=City/O=Organization/OU=$(whoami)/CN=*.dev
  -keyout ~/Sites/ssl/private.key \
  -out ~/Sites/ssl/selfsigned.crt
```

START APACHE

Start Homebrew's Apache and set to start on login:

```
brew services start httpd22
```

RUN WITH PORT 80

You may notice that *httpd.conf* is running Apache on ports 8080 and 8443. Manually adding ":8080" each time you're referencing your dev sites is no fun, but running Apache on port 80 requires root. The next two commands will create and load a firewall rule to forward port 80 requests to 8080, and port 443 requests to 8443. The end result is that we don't need to add the port number when visiting a project dev site, like "http://projectname.dev/" instead of "http://projectname.dev:8080/".

The following command will create the file */Library/LaunchDaemons*

/co.echo.httpdfwd.plist as root, and owned by root, since it needs elevated privileges:

```
sudo bash -c 'export TAB=$'"'"'\t'"'"'
cat > /Library/LaunchDaemons/co.echo.httpdfwd.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.ap
<plist version="1.0">
<dict>
${TAB}<key>Label</key>
${TAB}<string>co.echo.httpdfwd</string>
${TAB}<key>ProgramArguments</key>
${TAB}<array>
${TAB}${TAB}<string>sh</string>
${TAB}${TAB}<string>-c</string>
${TAB}${TAB}<string>echo "rdr pass proto tcp from any to any port {
${TAB}</array>
${TAB}<key>RunAtLoad</key>
${TAB}<true/>
${TAB}<key>UserName</key>
${TAB}<string>root</string>
</dict>
</plist>
EOF'
```

This file will be loaded on login and set up the 80->8080 and 443->8443 port forwards, but we can load it manually now so we don't need to log out and back in:

```
sudo launchctl load -Fw /Library/LaunchDaemons/co.echo.httpdfwd.pli
```

PHP

The following is for the latest release of PHP, version 5.6. If you'd like to use 5.3, 5.4 or 5.5, simply change the "5.6" and "php56" values below appropriately.

```
brew install -v homebrew/php/php56
```

Set timezone and change other PHP settings (sudo is needed here to get the current timezone on OS X) to be more developer-friendly, and add a PHP error log (without this, you may get Internal Server Errors if PHP has errors to write and no logs to write to):

```
(export USERHOME=$(dscl . -read /Users/`whoami` NFSHomeDirectory |
```

Fix a pear and pecl permissions problem[6]:

```
chmod -R ug+w $(brew --prefix php56)/lib/php
```

The optional Opcache extension will speed up your PHP environment dramatically, so let's install it. Then, we'll bump up the opcache memory limit:

```
brew install -v php56-opcache
```

```
/usr/bin/sed -i '' "s|^\(\;\)\{0,1\}[[:space:]]*\(opcache\.enable[[
```

Finally, let's start PHP-FPM:

```
brew services start php56
```

Optional: At this point, if you want to switch between PHP versions, you'd want to: brew services stop php56 && brew unlink php56 && brew link php54 && brew services start php54. No need to touch the Apache configuration at all!

DNSMasq

A difference now between what I've shown before[7], is that we don't have to run on port 53 or run dnsmasq as root. The end result here is that any DNS request ending in *.dev* reply with the IP address *127.0.0.1*:

```
brew install -v dnsmasq
```

```
echo 'address=/.dev/127.0.0.1' > $(brew --prefix)/etc/dnsmasq.conf
```

```
echo 'listen-address=127.0.0.1' >> $(brew --prefix)/etc/dnsmasq.con
```

```
echo 'port=35353' >> $(brew --prefix)/etc/dnsmasq.conf
```

Similar to how we run Apache and PHP-FPM, we'll start DNSMasq:

```
brew services start dnsmasq
```

With DNSMasq running, configure OS X to use your local host for DNS queries ending in *.dev*:

```
sudo mkdir -v /etc/resolver
```

```
sudo bash -c 'echo "nameserver 127.0.0.1" > /etc/resolver/dev'
```

```
sudo bash -c 'echo "port 35353" >> /etc/resolver/dev'
```

To test, the command ping -c 3 fakedomainthatisntreal.dev should return results from 127.0.0.1. If it doesn't work right away, try turning WiFi off and on (or unplug/plug your ethernet cable), or reboot your system.

Great! So, what did I do?

We set up Apache to run on boot on ports 8080 and 8443 with auto-VirtualHosts for directories in the *~/Sites* folder and PHP-FPM via mod_fastcgi. The OS X firewall will forward all port 80 traffic to port 8080 and port 443 to port 8443, so we don't have specify the port number when visiting web pages in local web browsers or run Apache as root. MySQL is installed and set to run on boot as well. DNSMasq and some OS X configuration is used to direct any hostname ending in *.dev* to the local system to work in conjunction with Apache's auto-VirtualHosts.

WHAT DO I DO NOW?

You shouldn't need to edit the Apache configuration or edit /etc/hosts for new local development sites. Simply create a directory in *~/Sites* and then reference "http://" + that foldername + ".dev/" in your browser to access it.

For example, download Drupal 7 to the directory *~/Sites/firstproject*, and it can then be accessed at *http://firstproject.dev/* without any additional configuration. A caveat - you will need to uncomment the line in Drupal's *.htaccess* containing "RewriteBase /" to work with the auto-VirtualHosts configuration.

WHAT IF THIS "AUTO-VIRTUALHOST" DOESN'T WORK FOR *[OTHER PROJECT]*?

If you need to create a manual VirtualHost in Apache because the auto-VirtualHost does not work for your configuration, you will need to declare it before the auto-VirtualHosts if you're also going to use .dev as the TLD. Otherwise the auto-VirtualHost block will be accessed first. I have a commented-out sample for a manual VirtualHost entry in "*~/Sites/httpd-vhosts.conf*" you may use.

Troubleshooting

The commands above can be run on a fresh Yosemite system without issue as I've tested with fresh installs in VMware. Nearly every problem I've heard about has been related to upgrading from 10.9 or earlier, or switching to this style of setup from another or similar setup. The easiest thing to do would be to brew uninstall each component referenced above, delete all related config files in *$(brew --prefix)/etc* and any Homebrew files in *~/Library/LaunchAgents*, and start over. If that's a bit heavy-handed, check each of the configuration files edited in this guide and look for duplicate entries or typos.

You can also check log files for error output:

> Apache: *$(brew --prefix)/var/log/apache2/error_log*, or run httpd -DFOREGROUND and look for output
>
> PHP-FPM: *$(brew --prefix)/var/log/php-fpm.log*
>
> MySQL: *$(brew --prefix)/var/mysql/$(hostname).err*
>
> DNSMasq: no log file, run dnsmasq --keep-in-foreground and look for output

Leave a comment if you have any questions or problems!

LINKS

1. https://echo.co/blog/os-x-109-local-development-environment-apache-php-and-mysql-homebrew
2. http://brew.sh/
3. http://www.vagrantup.com/
4. https://github.com/Homebrew/homebrew-services
5. http://stackoverflow.com/a/10856211/534275

6 https://github.com/Homebrew/homebrew-php/issues/1039#issuecomment-41307694

7 https://echo.co/blog/never-touch-your-local-etchosts-file-os-x-again

464 Comments **Echo & Co.** 1 **Login**

Recommend 47 **Share** **Sort by Best**

Join the discussion…

Paul • 2 years ago
Just wanted to say, this has been the least painful and most well-configured of all my many, many MAMP-esq development stacks - a setup process that has, in the past, significantly reduced the joy of a new machine. This just worked, first try, love the auto-vhosts: THANK YOU!
15 • Reply • Share ›

Alan Ivey Paul • 2 years ago
Glad to hear, and thank you for the kind of words!
• Reply • Share ›

Bertrand Kintanar • a year ago
Can we request for a refresh for this post for El Capitan? Please?
4 • Reply • Share ›

Michael J Kormendy Bertrand Kintanar • a year ago
What isn't working for you when you go through these steps?
1 • Reply • Share ›

Hadi Michael J Kormendy • a year ago
In El Capitan steps of installing php and dnsmasq see this error

```
Error: The `brew link` step did not complete successfully
The formula built, but is not symlinked into /usr/local
Could not symlink sbin/php-fpm
/usr/local/sbin is not writable.
```

and then we cannot start php56 and dnsmasq

```
$ brew services start php56
Error: Could not read the plist for `php56`!
```

1 • Reply • Share ›

reptar Hadi • a year ago

```
sudo chown $(whoami):admin /usr/local/sbin \
```

I see you like to read printed material. You should check out Nicco's book *The End of Big*: http://endofbig.com