Setting up a Mac for Development

Date: 30/12/2014

Version: 1.1

Author: Lucas Challamel

Getting the workstation ready - The Mac path!

As a non Microsoft Developer working on a Mac, you need to get yourself a few productivity tools which unfortunately are not all available out of the box on your Macbook or iMac.

STEP 1 - Starting with a few Mac OS X enhancements

The Finder: Even with the latest OS X 9 Mavericks update, the Finder still has some shortcomings, and in particular still lacks the ability to quickly switch display of hidden system files. My recommended solution for that is to switch to TotalFinder that is US$15 well invested for your productivity. And it's more lightweight and stable than the very ambitious Pathfinder (US$39)

The Terminal: The native Mac terminal is OK, but not always easy to summon in context. To improve usability in that space, I have taken 2 actions: First, I have downloaded the excellent and free TotalTerminal utility from the same BinaryAge company publishing Total Finder. With its Visor feature, you can always summon the Terminal in context using CTRL+~, very handy. And secondly, I am leveraging the new service available in Mavericks which allows to open a "new Terminal tab at Folder", as per instructions found here. Right-click any folder, select Services>Open Terminal, and here you go. A true game changer.

A decent Text Editor: Textedit is definitely outdated, and I wonder when will Apple decide to give it a bit of attention span! In the meantime, the modern developer will invest in a Sublime Text 3 or Textmate 2 license.

STEP 2 - Install CLI developer tools:

Mac OS X native tools

First we need Mac native tools, which are not installed by default but usually come with Xcode.

Install/Update latest Xcode with Command Line Tools: Mac App Store - https://developer.apple.com/xcode/

If you are not planning to develop Mac or iOS apps and do not need Xcode just yet, these tools can simply be installed by using the following command in Terminal: **$ xcode-select** —install or even simply $ gcc

This will trigger a dependency pop-up indicating that Xcode requires the CLI developer tools and offering to install them: Just click install and wait for 30min.

Java: Java is an important language, and unfortunately it is not available natively anymore in the latest releases of Mac OS X.

The current Java version recommended by Apple is 1.6 and it can be downloaded here.

If for some reason you need to work on Java 1.7 or 1.8, visit the official Java website, and download the Java SE 1.7 JDK:
http://docs.oracle.com/javase/7/docs/webnotes/install/mac/mac-jdk.html

Ruby and Python: Ruby and Python, two immensely popular object-oriented scripting languages, have been installed as part of OS X for many years now. But their relevance to software development, and especially application development, has assumed an even greater importance since OS X v10.5. And as a modern developer, in particular in the PHP space, you are likely to use Ruby scripts for automation and deploy on, with Composer, Capistrano and Chef for instance.

Install Homebrew

Homebrew is a package manager for Mac and it helps installing additional services and utilities, that Apple didn't, in an organised manner, managing symlinks into the /usr/local folder. Just like CHEF consumes recipes written in Ruby for server provisioning, Homebrew consumes Ruby-written formulas to install software packages on your Mac.

Install latest XQuartz: https://xquartz.macosforge.org/landing/ (hard dependency for Homebrew)

Then to install Homebrew

```
$ ruby -e "$(curl -fsSL
https://raw.github.com/Homebrew/homebrew/go/install)"
$ brew doctor
```

To install Ruby with Homebrew:

```
$ brew install ruby
```

To manage the different versions of Ruby you may have on your system, use RVM:

```
$ \curl -sSL https://get.rvm.io | bash -s -- --autolibs=read-fail
```

You may also want to install **Bundler** (http://bundler.io) , a ruby gem dependency manager. Similar to the PHP dependency manager Composer, Bundler lets you define all of your

application's gems in a Gemfile located in your project's root directory, then run bundle install. To install Bundler run:

```
$ gem install bundler
```

Install PHP5.5-FPM

What is PHP-FPM? http://php.net/manual/en/install.fpm.php

FPM (FastCGI Process Manager) is an alternative PHP FastCGI implementation with some additional features (mostly) useful for heavy-loaded sites.

Preparation

Warning for Yosemite users:

As a requirement of the PHP compilation process, run:

```
$ sudo ln -s
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/
Developer/SDKs/MacOSX10.10.sdk/usr/include/ /usr/include
```

Search for available PHP formulas (formula's in homebrews are equivalent to packages in aptitude)

```
$ brew search php
```

It will return long list of php 5.2, 5.3, 5.4 packages. We need 5.4. Tap it using:

```
$ brew tap homebrew/php
```

```
# If you do not tap homebrew/dupes you may get Error: No available
formula for zlib
```

```
$ brew tap homebrew/dupes
```

```
# Skip this if using PHP 5.4 or higher
```

```
$ brew tap homebrew/versions
```

Build PHP5.5

Before we build PHP 5.5, you may like to exercise options using:

```
$ brew options php55
```

We have built it using:

```
$ brew install --without-apache --with-fpm --with-mysql php55
```

After quite a long wait, you can verify php & php-fpm version using php -v and php-fpm -v respectively.

```
$ php -v
```

```
$ php-fpm -v
```

```
$ brew list php55
```

Configure PHP-FPM

Check that PHP-FPM is listening on port 9000:

```
$ lsof -Pni4 | grep LISTEN | grep php
```

The output should look something like this:

```
php-fpm   69659  frdmn    6u  IPv4 0x8d8ebe505a1ae01      0t0  TCP 127.0.0.1:9000 (LISTEN)
php-fpm   69660  frdmn    0u  IPv4 0x8d8ebe505a1ae01      0t0  TCP 127.0.0.1:9000 (LISTEN)
php-fpm   69661  frdmn    0u  IPv4 0x8d8ebe505a1ae01      0t0  TCP 127.0.0.1:9000 (LISTEN)
php-fpm   69662  frdmn    0u  IPv4 0x8d8ebe505a1ae01      0t0  TCP 127.0.0.1:9000 (LISTEN)
```

Adding PHP-FPM to startup routine

Please check exact plist filename in /usr/local/Cellar/php55/

```
$ cp /usr/local/Cellar/php55/5.5.19/homebrew.mxcl.php54.plist
~/Library/LaunchAgents/
```

Note: There's a nasty hardcoded path in this file which you want to get rid of, in order to prevent continuity in case of PHP upgrade:

Replace line

```
    <string>/usr/local/Cellar/php55/5.5.19/sbin/php-fpm</string>
```

By

```
    <string>/usr/local/opt/php55/sbin/php-fpm</string>
```

and line

```
  <string>/usr/local/Cellar/php55/5.5.19/var/log/php-fpm.log</string>
```

By

```
  <string>/usr/local/opt/php55/var/log/php-fpm.log</string>
```

To Start PHP-FPM:

```
$ launchctl load -w ~/Library/LaunchAgents/homebrew.mxcl.php55.plist
```

To Stop PHP-FPM

```
$ launchctl unload -w
~/Library/LaunchAgents/homebrew.mxcl.php55.plist
```

To create service shorthands, just edit your .bash_profile and add these lines:

```
alias php-fpm.start="launchctl load -w /usr/local/opt/php55/homebrew.mxcl.php55.plist"
alias php-fpm.stop="launchctl unload -w /usr/local/opt/php55/homebrew.mxcl.php55.plist"
alias php-fpm.restart='php-fpm.stop && php-fpm.start'
```

Then reload the .bash_profile:

```
source ~/.bash profile
```

PHP FPM can run on a Unix socket or IP port, with the latter being the default. However, when you have multiple virtual hosts running on different ports, it's best not to use IP port in FPM configuration. Luckily, we can change that in the /usr/local/etc/php/5.5/php-fpm.conf.

Change the configuration from

```
listen = 127.0.0.1:9000
```

to the following

```
listen = /usr/local/var/run/php/php5-fpm.sock
```

Make sure this folder exists: /usr/local/var/run/php/

We also need to make one small security change in the php configuration.Open up php.ini:

```
$ sudo nano /usr/local/etc/php/5.5/php.ini
```

Find the line, cgi.fix_pathinfo=1, and change the 1 to 0.

```
cgi.fix pathinfo=0
```

If this number is kept as 1, the php interpreter will do its best to process the file that is as near to the requested file as possible. This is a possible security risk. If this number is set to 0, conversely, the interpreter will only process the exact file path—a much safer alternative. Save and Exit.

Then restart PHP-FPM.

Install Pear

http://pear.php.net

PEAR is a framework and distribution system for reusable PHP components. Although nearly deprecated, it is still useful to install and manage globally some PHP utilities like PHPUnit or Code Sniffer.

```
$ curl -O http://pear.php.net/go-pear.phar
$ sudo php -d detect_unicode=0 go-pear.phar
ln -s /usr/local/opt/php55/bin/pear /usr/local/bin/pear
```

Install Composer

What is Composer? https://getcomposer.org

Composer is a tool for dependency management in PHP. It allows you to declare the dependent libraries your project needs and it will install them in your project for you.

```
$ brew install composer
$ composer --version
```

Install Codeception

We use Codeception to script and run functional testing: http://codeception.com

```
$ brew install codeception
$ codecept --version
```

Install Node

What is NodeJS? http://nodejs.org

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

```
$ brew install node
$ node --version
```

Note: If an error is thrown trying to install NPM, you can try

```
$ sudo brew postinstall node
```

Want to try Node? Do the Hello World: http://howtonode.org/hello-node

Create a new program.js file containing:

```
// Load the http module to create an http server.
var http = require('http');
// Configure our HTTP server to respond with Hello World to all
requests.
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});
// Listen on port 8000, IP defaults to 127.0.0.1
server.listen(8000);
// Put a friendly message on the terminal
console.log("Server running at http://127.0.0.1:8000/");
```

Then run the file with:

```
$ node program.js
```

And check in a browser at http://localhost:8000

Install Node Package Manager NPM

```
$ sudo npm install npm -g
$ npm -v
```

Install GruntJS

GruntJS is a Javascript task runner. In one word, it's about automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it through a Gruntfile, a task runner can do most of that mundane work for you—and your team—with basically zero effort. http://gruntjs.com

Update npm then install Grunt CLI:

```
$ npm update -g npm
$ npm install -g grunt
$ npm install -g grunt-cli
$ grunt --version
```

Install Bower

What is Bower? http://bower.io
A front end package manager. Use this to install javascript frameworks and jquery. Install it globally:

```
$ npm install -g bower
$ bower --version
```

Install Git

What is Git? http://git-scm.com

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

```
$ brew install git
$ echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
$ which git
$ git --version
$ git config --global user.name "Your Full Name"
$ git config --global user.email "Your Email Address"
```

Install Capistrano

What is Capistrano? http://capistranorb.com
A remote server automation and deployment tool originally built for Ruby on Rails projects. You can use this tool to deploy any type of web app from the command line. It's fast, simple, and easy to setup. I use it to deploy wordpress projects. To install Capistrano run:

```
$ gem install capistrano
```

Install local SMTP service

Mac OS X Postfix option

OS-X Leopard comes pre-installed with a Postfix version. No need to install it via darwin ports or other such mess (actually uninstall it if you have previously manually installed it via ports or something similar). Postfix just needs to be enabled and following sequence of several easy steps explains how to do it:

```
$ sudo nano /System/Library/LaunchDaemons/org.postfix.master.plist
```

add following line before the closing </dict> tag:

```
<key>RunAtLoad</key> <true/> <key>KeepAlive</key> <true/>
```

Start service with

```
$ sudo launchctl
launchd% start org.postfix.master
```

You can also start simply with "sudo postfix start" but the above command will start via launch sequence and test the configuration you edited in the previous step.

Check that SMPT is running:

```
$ telnet localhost 25
```

Mailcatcher option

MailCatcher runs a super simple SMTP server which catches any message sent to it to display in a web interface. Run mailcatcher, set your favourite app to deliver to smtp://127.0.0.1:1025 instead of your default SMTP server, then check out http://127.0.0.1:1080 to see the mail that's arrived so far. More on http://mailcatcher.me

To install Mailcatcher:

```
$ gem install mailcatcher
$ mailcatcher
```

Go to http://localhost:1080/

Send mail through smtp://localhost:1025

Note: As of 12/12/2014, there is a broken dependency in the GEM install process. This is well documented here: https://github.com/sj26/mailcatcher/issues/155 . Solution is to install a specific version of the i18n gem:

```
$ gem uninstall i18n
$ gem install i18n -v 0.6.11
```

Step 3 - Additional web services

Database server

Installing MySQL on Mac OS X

Run following command:

```
$ brew install mysql --enable-debug
```

In case you need mysql-workbench, please download it from here. It cannot be installed via brew.

Adding MySQL to startup routine

Please check exact plist filename in /usr/local/Cellar/mysql/

```
$ cp /usr/local/Cellar/mysql/5.6.21/homebrew.mxcl.mysql.plist
~/Library/LaunchAgents/
```

To Start

```
$ launchctl load -w ~/Library/LaunchAgents/homebrew.mxcl.mysql.plist
```

To Stop

```
$ launchctl unload -w
~/Library/LaunchAgents/homebrew.mxcl.mysql.plist
```

Manual Start/Stop

Start mysql:

```
$ /usr/local/mysql/support-files/mysql.server start
```

Stop mysql:

```
$ /usr/local/mysql/support-files/mysql.server stop
```

See other options

```
$ /usr/local/mysql/support-files/mysql.server -h
```

Usage: mysql.server {start|stop|restart|reload|force-reload|status} [MySQL server options]

To create service shorthands, just edit your .bash_profile and add these lines:

```
alias mysql.start="launchctl load -w /usr/local/opt/mysql/homebrew.mxcl.mysql.plist"
alias mysql.stop="launchctl unload -w /usr/local/opt/mysql/homebrew.mxcl.mysql.plist"
alias mysql.restart='mysql.stop && mysql.start'
```

Then reload the .bash_profile:

```
source ~/.bash_profile
```

Changes to MySQL Config (optional)

For security

Run following command to improve security of your mysql setup. It will present you wizard to set mysql root password among other things:

```
$ mysql_secure_installation
$ (...)
```

```
$ mysql -u root -p
```

For workbench

Following changes will make it easy to use MySQL WorkBench

```
$ ln -s /usr/local/opt/mysql /usr/local/mysql
$ sudo ln -s /usr/local/opt/mysql/my.cnf /etc/my.cnf
```

Web server: NGINX

Nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server:

http://nginx.org/en/

Installing Nginx on Mac OS X

Run following command:

```
$ brew install nginx
```

Adding Nginx to startup routine

Please check exact plist filename in /usr/local/Cellar/mysql/

```
$ cp /usr/local/Cellar/nginx/1.6.2/homebrew.mxcl.nginx.plist
~/Library/LaunchAgents/
```

To Start

```
$ launchctl load -w ~/Library/LaunchAgents/homebrew.mxcl.nginx.plist
```

To Stop

```
$ launchctl unload -w
~/Library/LaunchAgents/homebrew.mxcl.nginx.plist
```

To create service shorthands, just edit your .bash_profile and add these lines:

```
alias nginx.start='sudo nginx'
alias nginx.stop='sudo nginx -s quit'
alias nginx.reload='sudo nginx -s reload'
alias nginx.restart='nginx.stop && nginx.start'
alias nginx.logs.error='tail -250f /usr/local/etc/nginx/logs/error.log'
alias nginx.logs.access='tail -250f /usr/local/etc/nginx/logs/access.log'
alias nginx.logs.default.access='tail -250f /usr/local/etc/nginx/logs/default.access.log'
alias nginx.logs.default-ssl.access='tail -250f
/usr/local/etc/nginx/logs/default-ssl.access.log'
alias nginx.logs.phpmyadmin.error='tail -250f /usr/local/etc/nginx/logs/phpmyadmin.error.log'
```

```
alias nginx.logs.phpmyadmin.access='tail -250f
/usr/local/etc/nginx/logs/phpmyadmin.access.log'
```

Note: We'll have to make sure these log files exist down the track

Then reload the .bash_profile:

```
source ~/.bash_profile
```

Changes to Nginx Config

By default, Nginx setup expects you to define all virtual hosts in /usr/local/etc/nginx/nginx.conf

```
$ nano /usr/local/etc/nginx/nginx.conf
```

Edit default config file and following line to http{..} block

```
include sites-enabled/*.conf;
```

Next you can create sites-available and sites-enabled folder to manage virtual hosts config.

```
$ mkdir -p /usr/local/etc/nginx/logs
$ mkdir -p /usr/local/etc/nginx/sites-available
$ mkdir -p /usr/local/etc/nginx/sites-enabled
$ mkdir -p /usr/local/etc/nginx/conf.d
$ mkdir -p /usr/local/etc/nginx/ssl
```

The **NGINX Conf** file should look like this:

```
#user  nobody;
worker_processes  1;
pid        /usr/local/etc/nginx/logs/nginx.pid;
error_log  /usr/local/etc/nginx/logs/error.log debug;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    include /usr/local/etc/nginx/sites-enabled/*.conf;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /usr/local/etc/nginx/logs/access.log  main;
    sendfile        on;
    keepalive_timeout  65;
        index index.html index.php;
}
```

Connect NGINX to PHP-FPM

This can be done globally in the nginx.conf file, or on a per case basis in VHOST files, through an include directive of a PHP.CONF file.

Put the file here: /usr/local/etc/nginx/conf.d/php.conf

In any cases, the directives to add in the server section are:

PHP.CONF file

```
location ~ \.php$ {
        root           html;
        try_files   $uri = 404;
        #fastcgi_pass   127.0.0.1:9000;
        fastcgi_pass unix:/usr/local/var/run/php/php5-fpm.sock;
        fastcgi_index  index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include        fastcgi_params;
}
```

Create default sites parameters

We are preparing 3 sites:

- A default site in /usr/local/var/www responding on port HTTP/80 => default.conf
- The same default site responding on HTTPS/463 => default-ssl.conf
- An endpoint for phpMyAdmin on port HTTPS/306 => phpmyadmin.conf

Create these files in folder /usr/local/etc/nginx/sites-available/

DEFAULT.CONF

```
server {
    listen       80;
    server_name  localhost;
    root       /usr/local/var/www/;
    access_log  /usr/local/etc/nginx/logs/default.access.log;
    location / {
        include   /usr/local/etc/nginx/conf.d/php.conf;
    }
    location = /info {
        allow   127.0.0.1;
        deny    all;
        rewrite (.*) /.info.php;
    }
    error_page  404     /404.html;
    error_page  403     /403.html;
}
```

DEFAULT-SSL.CONF

```
server {
    listen       443;
    server_name  localhost;
    root       /usr/local/var/www/;
    access_log  /usr/local/etc/nginx/logs/default-ssl.access.log;
    ssl                  on;
    ssl_certificate      /usr/local/etc/nginx/ssl/localhost.crt;
    ssl_certificate_key  /usr/local/etc/nginx/ssl/localhost.key;
    ssl_session_timeout  5m;
    ssl_protocols  SSLv2 SSLv3 TLSv1;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers   on;
    location / {
        include   /usr/local/etc/nginx/conf.d/php.conf;
    }
    location = /info {
        allow   127.0.0.1;
        deny    all;
        rewrite (.*) /.info.php;
    }
    error_page  404     /404.html;
    error_page  403     /403.html;
}
```

PHPMYADMIN.CONF

```
server {
    listen       306;
    server_name  localhost;
    root    /usr/local/share/phpmyadmin;
    error_log /usr/local/etc/nginx/logs/phpmyadmin.error.log;
    access_log  /usr/local/etc/nginx/logs/phpmyadmin.access.log;
    ssl                  on;
    ssl_certificate      /usr/local/etc/nginx/ssl/phpmyadmin.crt;
    ssl_certificate_key  /usr/local/etc/nginx/ssl/phpmyadmin.key;
    ssl_session_timeout  5m;
    ssl_protocols  SSLv2 SSLv3 TLSv1;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers   on;
    include   /usr/local/etc/nginx/conf.d/php.conf;
}
```

Linking everything together

We need provisional self signed SSL keys:

```
$ cd /usr/local/etc/nginx/ssl
```

```
$ openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj
"/C=US/ST=State/L=Town/O=Office/CN=localhost" -keyout
/usr/local/etc/nginx/ssl/localhost.key -out
/usr/local/etc/nginx/ssl/localhost.crt
$ openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj
"/C=US/ST=State/L=Town/O=Office/CN=phpmyadmin" -keyout
/usr/local/etc/nginx/ssl/phpmyadmin.key -out
/usr/local/etc/nginx/ssl/phpmyadmin.crt
```

Now we need to symlink the virtual hosts we want to enable into the sites-enabled folder:

```
$ ln -sfv /usr/local/etc/nginx/sites-available/default.conf
/usr/local/etc/nginx/sites-enabled/default.conf
$ ln -sfv /usr/local/etc/nginx/sites-available/default-ssl.conf
/usr/local/etc/nginx/sites-enabled/default-ssl.conf
$ ln -sfv /usr/local/etc/nginx/sites-available/phpmyadmin.conf
/usr/local/etc/nginx/sites-enabled/phpmyadmin.conf
```

Restart NGINX with the shortcut: nginx.restart

An check all links works fine:

1. http://localhost → "Nginx works" page
2. http://localhost/info → phpinfo()
3. http://localhost/nope → " Not Found" page
4. https://localhost:443 → "Nginx works" page (SSL)
5. https://localhost:443/info → phpinfo() (SSL)
6. https://localhost:443/nope → "Not Found" page (SSL)
7. https://localhost:306 → phpMyAdmin (SSL)

STEP 4: Get a good IDE

Text editors

Textmate 2

TextMate brings Apple's approach to operating systems into the world of text editors. By bridging UNIX underpinnings and GUI, TextMate cherry-picks the best of both worlds to the benefit of expert scripters and novice users alike.

http://macromates.com

SublimeText 3

Sublime Text is a sophisticated text editor for code, markup and prose. You'll love the slick user interface, extraordinary features and amazing performance.

http://www.sublimetext.com

Enterprise IDEs

There are several options on Mac and I can mention here the well know Eclipse (All languages), Coda 2(PHP), Netbeans (Java, PHP, C++) or Aptana Studio (PHP, …)

But my preference now clearly goes for the great and consistent suite of IDEs offered by JetBrains for the different languages:

- IntelliJ for Java
- PHPStorm for PHP => That's the one I am mostly using (US$99 for a personal license)
- Web Storm for general purpose HTML/CSS development
- PyCharm for Python
- Rubymine for Ruby
- Appcode for iOS development

Setting up PHPStorm

(...)

PHP Utilities

We need to install XDebug, Phing, PHPUnit, XHProf and Codesniffer

XDebug

Xdebug is a free and open source project by Derick Rethans and is probably one of the most useful PHP extensions. It provides more than just basic debugging support, but also stack traces, profiling, code coverage, and so on. http://xdebug.org

```
$ sudo pecl install xdebug
```

And add this section to the PHP.INI file (or create an include INI file named XDEBUG.INI):

```
[xdebug]
zend_extension=/usr/local/opt/php55/lib/php/extensions/no-debug-non-zts-20121212/xdebug.so
xdebug.remote_enable=1
xdebug.remote_host=localhost
xdebug.remote_port=9000
```

Restart PHP-FPM and launch a PHPINFO page to check XDebug is properly loaded.

Phing

http://www.phing.info

Phing is a powerful addition or alternative to Capistrano. it's a PHP project build system or build tool based on Apache Ant. You can do anything with it that you could do with a traditional build system like GNU make, and its use of simple XML build files and extensible PHP "task" classes make it an easy-to-use and highly flexible build framework.

Features include running PHPUnit and SimpleTest unit tests (including test result and coverage reports), file transformations (e.g. token replacement, XSLT transformation, Smarty template transformations), file system operations, interactive build support, SQL execution, CVS/SVN/GIT operations, tools for creating PEAR packages, documentation generation (DocBlox, PhpDocumentor) and much, much more.

We'll use PEAR to install PHING globally. In case PEAR is not installed:

```
$ curl -O http://pear.php.net/go-pear.phar
$ sudo php -d detect_unicode=0 go-pear.phar
```

```
$ sudo pear channel-discover pear.phing.info
$ sudo pear channel-discover pear.pdepend.org
$ sudo pear channel-discover pear.phpmd.org
$ sudo pear channel-discover pear.phpdoc.org
$ sudo pear channel-discover pear.phpunit.de
$ sudo pear channel-discover pear.symfony.com
$ sudo pear channel-discover pear.netpirates.net
```

```
$ sudo pear install --alldeps phing/phing
```

and then

```
$ ln -s /usr/local/opt/php55/bin/phing /usr/local/bin/phing
$ phing -v
```

PHPUnit

PHPUnit is a programmer-oriented testing framework for PHP. It is an instance of the xUnit architecture for unit testing frameworks.

We can simply use Brew to install PHP Unit:

```
$ brew install phpunit
$ phpunit --version
```

XHProf

Again, we can simply use brew to install XHProf, amazingly convenient:

```
$ brew install php55-xhprof
```

Codesniffer

We can install Codesniffer with PEAR or Composer. Note that normally it has been installed as a dependency of PHING.

The composer command looks like:

```
$ composer global require "squizlabs/php_codesniffer=*"
```

Otherwise via PEAR:

```
$ sudo pear install PHP_CodeSniffer
```

```
$ ln -s /usr/local/opt/php55/bin/phpcs /usr/local/bin/phpcs
```

```
$ phpcs --version
```

```
$ ln -s /usr/local/opt/php55/bin/phpcbf /usr/local/bin/phpcbf
```

```
$ phpcbf --version
```

Other essential network utilities

For **FTP/SFTP and S3 access**, I have found nothing better than Transmit 4 from Panic Software (US$34).

If you are after a completely free software though, you'd would fall back to Cyberduck 4.

For **database connectivity**, the "ned plus ultra" solution is Navicat which offers connectivity to MySQL, PostgreSQL, Oracle, MS SQL Server, SQL Lite, MariaDB. Choose the Essential edition for US$16 only, or go pro with Navicat Premium ($239).

If you use exclusively MySQL, then Sequel Pro is the perfect and free solution. Likewise for PostgreSQL, simply use the Postgres.App.

Finally, you need a **web debugging proxy** and the obvious choice here is Charles 3.9 (US$50).

Interesting to note though that Telerik is currently porting the remarkable Fiddler to Linux and Mac (Apha release)

STEP 5: Virtualisation software for DEV environments

xAMP approaches

First and foremost, I would **strongly advise against** setting up localk xAMP suites, such as MAMP and XAMP for Mac. It may look handy at first glance to run your Apache and MySQL servers locally, but it quickly becomes a pain in the back with compatibility issues, when you start using PHP and Apache extensions. Very messy and inaccurate.

According to the idea that there is "nothing like the real stuff", I urge any serious developer to work with locally virtualised environments replicating LIVE destination environments, whatever the operating systemm

The best choice is to use Oracle Virtual Box, free and well supported.

Now you can manually install any of your favourite OS, and set your code folders as shared mount points.

Vagrant

But the latest delightful trick to do so is to use Vagrant to flick up an environment in 1 command line and no time:

Download Vagrant for your system, and then in the terminal simply VargrantUP! You can select an image on Vagrant Cloud: https://vagrantcloud.com/discover/featured

```
$ cd /~/my dev folder // Putting yourself in the folder you intend to store your web app
$ vagrant init hashicorp/precise64  // Installing a standard Ubuntu 12.04 LTS 64bits box
$ vagrant up
$ vagrant ssh // You are logged in!
```

In just a few minutes, you are logged into your VM and can start setting up the services you need fort your app, starting with Apache, MySQL, PHP and more.

More details about provisioning this environment in a chapter to come.

Docker

(...)

- A virtualisation software
-
-
-
- More development friendly browsers and plugins
    - Native Browser's dev tools
    - Firebug Lite
    - Web developer
    - Session Manager
    - JSON View
    - Resizer
    - Cookie Manager
    - Ghostery
    - YSlow
    - Pagespeed
    - BuiltWith
    - Wappalyzer
    - Postman REST client
    - Ruul screen ruler

Step 6 - Setup a build environment

Introduction

You can obviously use your Mac as a build server, especially with all the utilities already installed on it. Otherwise, you will need to provision a dedicated linux stack, with at least the following pre-requisites:

- Ruby
- PHP
- Git
- Phing
- Capistrano (optional)
- PHPUnit
- Composer
- Codeception
- GruntJS

Using Capistrano

Capistrano is probably the simplest and cheapest way to set up deployment automation in a Linux context. More information: http://capistranorb.com

- Hosted services: https://codeship.com/documentation/continuous-deployment/deployment-with-capistrano/
- How to: https://www.digitalocean.com/community/tutorials/how-to-use-capistrano-to-automate-deployments-getting-started

(...)

Using Jetbrains TeamCity

TeamCity is another nice CD solution, free of charge for small organisations and up to 20 build plans. More details: https://www.jetbrains.com/teamcity/

(...)

Using Thoughtworks Go server

Go Server is probably today the best enterprise grade, open source and free solution to set up Continuous Delivery. More details: http://www.go.cd

Set up Go on the local Mac

- Download Go server for Mac and Install: http://www.go.cd/download/
- Download Go Client for Mac and Install: http://www.go.cd/download/
- Run server + agent and hit http://localhost:8153

Alternatively, you can install using Homebrew:

```
$ brew cask install go-server
$ brew cask install go-client
```

Set up Go on a remote Linux server

- Download the right distro
- On CentOS, download RPMs and install Go server:
    - `$ rpm -i go-server-${version}.noarch.rpm`

Configure a proxy for Go with Apache

An example of how to configure Go with Apache is shown below.

Assumptions:

- You have Apache with mod_proxy installed
- The Apache server sits on the same machine as the Go server (localhost)
- You want to enforce SSL connections
- Listen nnn.nnn.nnn.nnn:80
- NameVirtualHost nnn.nnn.nnn.nnn:80

```
<VirtualHost nnn.nnn.nnn.nnn:80>
    ServerName go.local
    DocumentRoot /var/www/html
    SSLProxyEngine on
    SSLEngine on
    ProxyPass / https://localhost:8154/
    ProxyPassReverse / https://localhost:8154/
</VirtualHost>
```

Configure a proxy for Go with NGINX

Create a new config file in /usr/local/etc/nginx/sites-available/go-server.conf

```
server {
    listen       443;
    server_name  go.local;
    root       /usr/local/var/www/;
    access_log  /usr/local/etc/nginx/logs/goserver.access.log;
    ssl                  on;
    ssl_certificate      /usr/local/etc/nginx/ssl/localhost.crt;
    ssl_certificate_key  /usr/local/etc/nginx/ssl/localhost.key;
    ssl_session_timeout  5m;
    ssl_protocols  SSLv2 SSLv3 TLSv1;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers   on;
    location / {
        #include   /usr/local/etc/nginx/conf.d/php.conf;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass https://localhost:8154;
    }
    error_page  404     /404.html;
    error_page  403     /403.html;
}
```

```
server {
    listen       80;
    server_name  go.local;
    root        /usr/local/var/www/;
    access_log  /usr/local/etc/nginx/logs/goserver.access.log;
    location / {
        #include    /usr/local/etc/nginx/conf.d/php.conf;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass http://localhost:8153;
    }
    error_page  404      /404.html;
    error_page  403      /403.html;
}
```

Create an entry in =your local HOSTS file (/etc/hosts)

```
127.0.0.1    go.local
```

And then enable this confifg

```
$ ln -sfv /usr/local/etc/nginx/sites-available/go-server.conf
/usr/local/etc/nginx/sites-enabled/go-server.conf
$ nginx.stop
$ nginx.start
```

And hit https://go.local !

Setup Go users and security

You can connect go to an external LDAp system or simply use local passwords, as explained here: http://www.go.cd/documentation/user/current/configuration/dev_authentication.html

- Prepare a password file:
    - Navigate to /usr/local/etc
    - create a “go” folder
    - `$ htpasswd -c -s gopasswords {user}`
- Login to Go as an admin
- Navigate to the "Admin" section
- Click on the "Server Configuration" tab
- Fill out the "Password File Settings" field under the "User Management" section with /usr/local/etc/gopasswords

Adding build and packaging utilities

GruntJS

We already talked about GruntJS earlier, which is a great utility to have available on the Mac. It is definitely also a must have on a CD server, in order to carry out unit testing and pre-packaging optimisations at Javascript and CSS level. http://gruntjs.com

The best way to install GruntJS is a an NPM package. Update npm then install Grunt CLI:

```
$ npm update -g npm
$ npm install -g grunt
$ npm install -g grunt-cli
$ grunt --version
```

Effing Package Manager

FPM helps you build packages quickly and easily (Packages like RPM and DEB formats). https://github.com/jordansissel/fpm

Install FPM as a gem on your Mac and your build server:

```
$ gem install fpm
$ fpm --version
```

Maintenance

From time to time you need to update your Mac and make sure you have the latest upgrades and security patches applied.

Mac OS X updates

Visit the AppStore to check for OS level updates. Pay particular attention to XCode updates.

Homebrew updates

All brew commands are here:
https://github.com/Homebrew/homebrew/tree/master/share/doc/homebrew#readme

List the installed packages
```
$ brew list
```

Update the formulas and see what needs a refresh
```
$ brew update
```

Now upgrade your packages, individually or as a whole:
```
$ brew upgrade
```

If your paths and launch files are set properly, you should be fine even with an upgrade of PHP, MySQL, Nginx or NodeJS.

Cask updates
```
$ brew cask update
```

Pear updates

Simply run this to get a list of available upgrades:

```
$ sudo pear list-upgrades
```

And then to implement one oft hem

```
$ pear upgrade {Package Name}
```

Gem updates

All Gem commands are here: http://guides.rubygems.org/command-reference/

List the installed packages
```
$ gem list
```

List those needing and update

```
$ gem outdated
```

Then update gems individually or as a whole:

```
$ gem update
```

Node updates

Note itself should be updated with Brew on a Mac.

```
$ brew upgrade node
```

To update Node Package Manager itself, just run

```
$ sudo npm install npm -g
```

To list all packages installed globally

```
$ npm list -g
```

Check for outdated global packages:

```
$ npm outdated -g --depth=0
```

Currently the global update command is bugged, so you can either update packagers individually:

```
$ npm -g install {package}
```

Or run this script

```
#!/bin/sh
set -e
set -x
for package in $(npm -g outdated --parseable --depth=0 | cut -d: -f2)
do
    npm -g install "$package"
done
```

Note that all global modules are stored here: /usr/local/lib/node_modules