macOS

John O'Gorman (john@og.co.nz)

9 October 2017

Contents

1 Intro

This paper tries to give hints to people from a Unix and/or Linux background as to how to do things on the Apple Mac.

This document is a draft and will be finalised only when the problems related to installing and setting up PostgreSQL 9.4 or later and LedgerSMB 1.5 are solved completely.

1.1 Unix Based

The Apple macOS is based on NeXT, a graphical version of Unix and you can execute command line shell programs using the Terminal app or XQuartz xterm. But some of the common commands do not work as in Unix. In particular there are significant differences in the areas of users and groups, passwords, setting of environment variables, and others.

The Apple macOS is built on 2 levels -

1. a POSIX base called Darwin with a Unix kernel and commands and
2. a graphical system called Aqua superimposed on the Darwin base.

Apple supplies an application Terminal.app. Unix/Linux diehards like me are advised to find it using the Launchpad icon on the dock, look for Others and click on it. One of the icons is labelled Terminal. Click and drag it on to the dock. Now whenever you want a command line interface, just click on the Terminal icon.

To use xterm (an X terminal emulator) you have to have installed XQuartz which is a freely available implementation of X for the Apple Mac. Once XQuartz is installed you have Apple's version of X11 available.

With the introduction of Version 10.12.1 Apple have changed the name of the Mac's operating system from OS X (where X is pronounced /ten/) to macOS apparently to bring the name into line with iOS, watchOS, and tvOS for the iphone and iPad, Apple watch, and Apple TV. Apple supply a free application called Xcode which allows developers to create apps for all of the aforementioned devices.

I am currently running macOS 10.13 (High Sierra) on my Mac

1.2 Problems Unresolved

- Setting Environments variable outside of the shells
- We cannot get the –with-postgresql option to work with brew install phpxx. It complains that it is missing pg_fe.h

1.3 Homebrew

In the course of trying to get everything working for the installation of postgresql version 9.4 and then LedgerSMB we discovered the following

- Apple don't supply the Mac with complete applications needed by developers
- In particular the Apache webserver has most modules disabled and some omitted
- Perl has only rudimentary support
- Apple do not supply a packet manager to ease the installation of large applications

The solution for these issues is a 3rd party product called Homebrew (described later in this document).

We used Homebrew to install a full developer version of apache, cpanm, and perl.

1.4 Same as Unix/Linux

- Directories: /bin /usr/bin /sbin /usr/sbin /usr/local
- File commands: cat cp mv ln rm chown chmod cd
- Edit commands: vi emacs sed grep awk
- Programming languages: gcc cc java perl python ruby
- Bourne Again Shell: bash ~/.profile ~/.bash_profile ~/.bashrc
- Utilities: who whoami pwd find netstat df du tar zip gzip unzip gunzip hexdump

1.5 Slightly Different

- Case blindness: ls /Private works showing you the contents of /private (/Private does not exist). Similarly ls /users shows the contents of /Users (/users does not exist).
- locate command works fine but to update the file database: `sudo /usr/libexec/locate.updatedb`
- directory `/etc` is a symbolic link to `/private/etc`

1.6 Very Different

The Apple Mac graphical applications are part of Aqua and are stored in the following directories which have no equivalent in Unix:

- /Applications
- /Library
- /System
- /Users
- /Volumes

Some of the above names also occur within a user's home directory - in particular ~/Applications and ~/Library

- There is a command: `open` which tries to guess which program to invoke to execute or edit the target file.
- Processes are initiated via an application: /Library/LaunchAgents and /Library/LaunchDaemons and similar in ~ and in /System
  - LaunchAgents is used for apps invoked by a user
  - LaunchDaemons are run at startup
  - The command: `brew services` can be used to configure Launch see `https://github.com/Homebrew/homebrew-services`
  - When run with sudo, brew installs its .plist file(s) into /Library/LaunchAgents otherwise it installs into ~/Library/LaunchAgents

2 Setting PATH

The Mac uses the file: `/etc/path` and the directory: `/etc/paths.d` to hold the values used to set the environment variable: PATH.

2.1 /etc/path

Typical contents:

```
/usr/local/bin
/usr/bin
/bin
/usr/sbin
/sbin
```

The contents can be edited. We, for example have created a directory: /local and added /local/bin to the paths file.

2.2 /etc/paths.d

You can create a named file for each Application: e.g. 40-XQuartz, LyX, TeX, aubit, postgres. Put into each of these the path(s) required to run the application. For example for postgres you would put

```
/Library/PostgreSQL/9.3/bin
```

The above was for EnterpriseDB postgres which came with the Mac. After we had replaced it with postgres.app we changed this to

```
/Applications/Postgres.app/Contents/Versions/latest/bin
```

For LyX

```
/Applications/LyX.app/Contents/MacOS
```

For TeX

```
/Library/TeX/texbin
```

For Quartz

```
/opt/X11/bin
```

For Aubit 4GL

```
/local/a4gl/bin
```

Some of these will have created for you by the Mac installation process. But for others like LyX you will need to create the file and edit it with the correct path.

As usual when you next login you can test the path variable in xterm: `echo $PATH`

3 Setting Environment variables

The file `~/.profile` and/or `~/.bash_profile` and/or ~/.bashrc can be used just as with Unix/Linux with export commands to set environment variables. e.g.

```
export PGPORT=5432
```

The .profile file is only read when you invoke the Terminal or xterm applications. If you want the variables to be set for all logins then the official mechanism expected by macOS depends on the version of MacOS your are running. MacOS versions from 10.6 are:

Version	Nickname	Year	Comment
v10.6	Snow Leopard	2009	
v10.7	Lion	2011	Last for ~/.MacOS
v10.8	Mountain Lion	2012	/etc/launchd.conf
v10.9	Mavericks	2013	
v10.10	Yosemite	2014	
v10.11	El Capitan	2015	
v10.12	Sierra	2016	macOS!
v10.13	High Sierra	2017	

3.1 Max OSX 10.6 or earlier

macOS will read a file `~/.MacOSX/environment.plist`
If there is no directory ~/.MacOSX then you need to create it:

```
cd
mkdir .MacOSX
```

Now, if necessary create an empty file: environment.plist

```
touch .MacOSX/environment.plist
```

Now invoke the Mac command `open`:

```
open .MacOSX/environment.plist
```

The open command should invoke a Property List Editor (because of the plist suffix):

- Select Root
- Select Add a child
- Key should be the name of the variable you want to set
- Type should be String
- Value should be the value you want to set the variable to

Next time you log in the variable(s) should be set.

3.2 Mac OSX 10? or newer but not 10.12 or later

Mac will ignore /etc/environment.plist. Instead create or modify a file `/etc/launchd.conf`

```
sudo vi /etc/launchd.conf
setenv var value # where var is the name and value the value
setenv ... ....
```

You will need to logout and re-login for the settings to take effect.

3.3 macOS 10.12 (Sierra)

Unfortunately we do not know how to set environment variables for GUI applications. The above methods have been disabled in Sierra.

There is the possibility that the LauunchAgents and LaunchDaemons applications can be exploited to set environment variables.

We are working on that still. Websites have given an example:

Create a file say `environment.plist` and put it into `~/Library/LaunchAgents/`

```
<?xml version=''1'' encoding ‘‘UTF-8''?>
<!DOCTYPE plist PUBLIC ‘‘-//Apple//DTD PLIST 1.0//EN''
  ‘‘http:www.apple.com/DTDs/PropertyList-1.0.dtd''>
<plist version=''1.0''>
<dict>
  <key>Label</key>
  <string>my.startup</string>
  <key>ProgramArguments</key>
  <array>
    <string>sh</string>
    <string>-c<string>
    <string>
      launchctl setenv PGHOST localhost
      launchctl setenv PGPORT 5432
      launchctl setenv PGDATESTYLE 'SQL, dmy'
    </string>
  </array>
  <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

If environment.plist is already loaded in ~/Library/LaunchAgents then

```
launchctl unload ~/Library/LaunchAgents/environment.plist
```

Modify the plist file then

```
launchctl load ~/Library/LaunchAgents/environment.plist
```

4 Launch

Apple uses the terms:

- Agents to designate normal programs which can be invoked at boot up or login
- Daemons (as in Unix/Linux) to designate programs which run in the background and will respond to client requests to invoke other programs

To learn more use the commands:

```
man launchctl
man launchd.plist
man plist
```

The Apple Mac has 2 programs: launchd and launchctl to control the invocations of daemons and user programs.

1. Launchctl reads xml files called Property Lists which have a suffix: `.plist`
2. The Property List describes the properties which will be set in the application to be invoked by the daemon: launchd
3. The launchctl program controls the invocation of launchd. You can see how many instances of launchd have been invoked with the command: `ps -aef | grep launch`
4. There are 5 potential directories into which these property lists are installed:
   (a) /System/Library/LaunchDaemons (Apple supplied system wide daemon programs)
   (b) /System/Library/LaunchAgents (Apple supplied system wide per user applications)
   (c) /Library/LaunchDaemons (System wide daemons supplied by the system admin)
   (d) /Library/LaunchAgents (System wide user programs supplied by the system admin)
   (e) ~/Library/LaunchAgents (Per user programs provided by the user)

4.1 Using Xcode

Xcode is an apple development system which allows users to generate applications for all Apple device operating systems: iOS, tvOS, watchOS, and macOS.

To edit a property list file the official method is to invoke Xcodle. Click on the Launch icon and find Xcode and double-click its icon.

To create a new Property List file (e.g. env.plist)

- Click options: File -> New -> File... -> macOS
- Scroll down to Resource, click on the Property List icon
- Click on Next
- Save As Property List.plist: Overtype Property List with env
- Click Create
- You now see a panel with triples headed: Key Type Value

4.1.1 Entering Values

The triples you need to edit are rather obscure in their interface.

In each of the components you need to click near the left of the label, then after a pause you see a field into which you can enter the value.

e.g. For Key click on the left and when you see the field then type in EnvironmentVariables,

for Type you click on the up-down symbol and select from the popup types (e.g. dictionary, string, date, data, number, array, etc) Select dictionary. This should give you a new indented

for Value you click on the left and type into the field the value you want to enter.

For type array, Xcode won't give you a key field, it knows that you will enter strings.

For type dictionary, Xcode will expect you to enter key-value pairs.

4.1.2 plist arrays and dictionaries

For keys ProgramArguments and EnvironmentVariables you need to be alert to the indentation which indicates the components within the array or dictionary:

There is a small triangle icon which when clicked toggles between pointing down or to the right. Make sure it points down if you want to enter components within the array or dictionary.

A symbol + and a symbol - will appear when you place the pointer to the right of the key. Click the + to insert an item within the array or dictionary. If you don't see the next line indented then hit the - symbol and move up a level and try the + there.

4.1.3 plist structure

A Porperty List is an XML file with a header, and a dictionary between tags <dict> ... </dict>.
For our purposes we wish to create a plist file with the following components within its dictionary:

- Label - key Label, type string, value nz.co.og.env
- ProgramArguments - key ProgramArguments, type array, value /bin/bash
- EnvironmentVariables - key EnvironmentVariables, type dictionary, value(s)
  - key PGHOST, type string, value localhost
  - key PGPORT, type string, value 5432
  - etc
- RunAtLoad - key RunAtLoad, type boolean, value YES. Xcode will translate this to <true/>

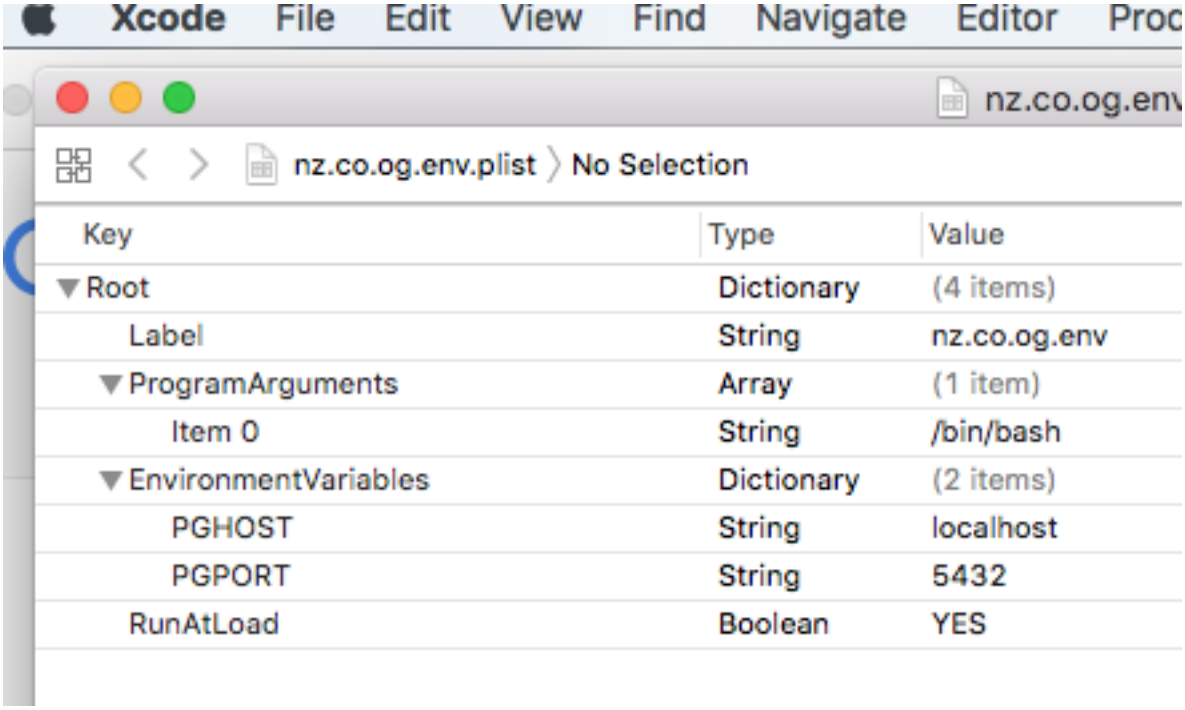
A useful tactic is to:

1. Insert a component - say RunAtLoad
2. Save the file (Command Key + S) or Click on File - Save
3. View in a terminal screen the resulting xml file (say env.plist). You will see that Xcode has created the XML header and all the structural markup: <key>, <array>, <dict>, <string>, <string>, and so on.
4. If alls well continue with previous component. Xcode seems to insert them in reverse order. I don't think the order is important to launchctl.
5. And so on

At the end of this process, you should see something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>nz.co.og.env</string>
<key>ProgramArguments</key>
<array>
<string>/bin/bash</string>
</array>
<key>EnvironmentVariables</key>
<dict>
<key>PGHOST</key>
<string>localhost</string>
<key>PGPORT</key>
<string>5432</string>
</dict>
    <key>RunAtLoad</key>
<true/>
</dict>
</plist>
```

The first 4 lines and the last 2 have been supplied by Xcode. You have told Xcode what to place between the first <dict> and last </dict> tags.
In our example the Label: nz.co.og.key has to be unique on your computer and it is typical for Apple and developers to invert their domain name, followed by an arbitrary key.



The above screen shot was taken using Apple's Shift+CommandKey+4 then clicking and dragging the area wanted.

4.2 Install plist file

- Copy the plist file into ~/Library/LaunchAgents

This did not work!

5 Users and Groups

The Mac does not use the `/home` directory. It puts users into a directory: `/Users`

6 Passwords

The Mac uses LDAP (Lightweight Directory Access Protocol) instead of the Unix/Linux methods.
So `/etc/passwd`, `/etc/shadow` may exist but are ignored, and `passwd` commands don't work.
To set or reset passwords you must use the Apple System Preferences interface:

- Click on the Apple icon on top left of screen
- Click on System Preferences then click on Users and Groups
- Click on the padlock icon to unlock and allow changes
- Choose the user you wish to change the password for
- Click reset password
- Click the padlock icon to lock further changes
- Exit the app by clicking on the red exit button (top left of window)

7 High Sierra macOS 10.13

When we upgraded from Sierra macOS 10.12.6 to High Sierra macOS 10.13 the upgrade worked OK but dismay followed - we no longer had the ftp command.

Our attempted solution to this was to use Homebrew:

```
brew install inetutils
```

The ftp command was now in /usr/local/bin

When we ran it, it seemed to work in that we were able to login to a remote site. The cd html command worked but the command `ls` did not and nor did `mput`.

The error messages are:

```
500 Illegal PORT command
500 Unknown command
425 Use PORT or PASV first
```

This was resolved by running the new ftp with the option: `-p` # which means passive.

The next problem was that PDF files seemed to get contaminated on transfer. The solution here ended up as:

1. Use command: `ftp -p` to login
2. run command: `type binary`
3. `mput filename.pdf`

8 PostgreSQL

In our case Postgres has come installed on the machine but is only accessible to the user: postgres. PostgreSQL is a free and opensource RDBMS (Relational Database Management System)

The installed postgres is version 9.3 sourced from EnterpriseDB and is based on the open source postgreSQL from UCB (University of California Berkeley) with addons which allow management, integration, and migration. Michael Stonebraker has been a leading figure in the develpment of postgres and its predecessor Ingres and for a while he was also employed by Informix to work on their OnLine engine.

There are some problems to resolve before you can use Postgres.

Ledgersmb requires postgres v 9.4 or later.

8.1 Location

EnterpriseDB PostgreSQL was installed in directories:

```
/Library/PostgreSQL/9.3/
```

with executables in subdirectory bin

```
/Library/PostgreSQL/9.3/bin
```

After replacing this with postgres.app the installation was in

```
/Applications/Postgres.app/Contents/Versions/latest
```

where `latest` was a symbolic link to `9.6`

```
/Applications/Postgres.app/Contents/Versions/latest/bin
```

Here you will find commands such as initdb, createdb, createuser, psql, and many others

All the postgres executables have man files. You can therefore type commands like: `man psql` to get manual instructions.

8.2 Postgres password

You need to reset the postgres password unless you know it already. As per an above section, you cannot use the usual Unix/Linux passwd command. It does not work with LDAP used on the Mac. You have to use the supplied GUI app for users. Once this is done you will be able to login is as user postgres using the command:

```
sudo su postgres
```

This will later allow you to create a database cluster, grant access to other users, allow other users to create databases etc.

8.3 Create database cluster

You need to create a new directory for postgres. The directory needs to be owned by the postgres superuser (by default user postgres). Once created you then need to become user postgres and run the initdb command. The example below assumes you will create /local/data as your database directory.

```
sudo su
mkdir /local/data #if it does not already exist
chown postgres /local/data
su postgres
$initdb -D /local/data
```

An alternative to the initdb command is:

```
$pg_ctl -D /local/data initdb
```

While this seems more verbose, you can use pg_ctl to de everything e.g. start, stop, restart the database server etc. So most people choose to use pg_ctl as a maid or all work. To test that the pg_ctl command has worked you can type:

```
pg_ctl -D /local/data status
```

After you have run the initdb command, postgres populates the directory /local/data with several files the most significant of which are as follows but with the comments removed.

8.3.1 pg_hba.conf

```
# IPv4 local connections:
host all all 127.0.0.1/32 trust
# IPv6 local connections:
host all all ::1/128 trust
```

8.3.2 postgresql.conf

```
# These settings are initialized by initdb, but they can be changed.
lc_messages = 'en_NZ.UTF-8' # locale for system error message
# strings
lc_monetary = 'en_NZ.UTF-8' # locale for monetary formatting
lc_numeric = 'en_NZ.UTF-8' # locale for number formatting
lc_time = 'en_NZ.UTF-8' # locale for time formatting
# default configuration for text search
default_text_search_config = 'pg_catalog.english'
```

8.3.3 postmaster.pid

```
80369
/local/data
1503025742
5432
/tmp
localhost
  5432001    196608
```

8.4 Start the database server

```
pg_ctl start -D /local/data -l logfile
```

To check that this has worked

```
pg_ctl -D /local/data status
```

Postgres honours a set of environment variables which, if set, can eliminate the need for options like -D above: e.g.:

```
export PGDATABASE=mydemo
export PGDATA=/local/data
export PGPORT=5432 #the default
export PGDATESTYLE='SQL, dmy'
export PGHOST=localhost
```

Put these statements into either `~/.profile` or `~/.bash_profile`
You can now enter commands like `pg_ctl status`
without the need for the -D argument

8.5 Create user

If you wish to access the database server as a user other than postgres, you need to run the command createuser which comes with postgres. There are options to allow the ability to create databases and drop them.
To create a user (say John) with full privileges:

```
sudo su postgres
createuser -s -d -r John
```

The upper case J in John works OK without having to be quoted.
Once you have used this createuser option, you can login as John and with full privileges create and drop databases, run psql SQL queries and other statements.

8.6 psql

psql is a command line SQL interpreter which can be used to create and drop databases and tables and stored procedures, and to run SQL statements: select, insert, delete, etc. It uses the readline library (similar to the shells) to hold a history of commands so that you can arrow up to previous commands and edit them for re-execution.

8.7 Upgrade to postgres 9.6

Options seem to be

- use Homebrew: brew install postgres
- install postgres.app.dmg

We chose to use postgres.app

8.8 Adding Postgres

We could not find a method of upgrading the EnterpriseDB implementation of PostgreSQL 9.3.
So we decided to

1. Shutdown PostgreSQL 9.3
   (a) `pg_ctl stop`
2. Uninstall PostgreSQL 9.3 by running Finder and then searching for a file: uninstall-xxxxx. double-click and the app uninstalls itself.
3. Download Postgres.app from `httpd://postgresapp.com`
4. Install postgres.app by
   (a) clicking the Download icon on the dock
   (b) double-clicking on the postgres.app.dmg icon
   (c) control-clicking on the Postgress.app icon and dragging it into the Applications directory icon

8.9 Postgres.app

Postgres.app installs by default into the `/Applications` directory
When we installed it it included both versions 9.5 and 9.6 or PostgreSQL within the Versions subdirectory
It places your preferences into ~/Library/Application Support/Postgres/
It sets your default Database location in `~/Library/Application Support/postgres/var-9.6/`
You need to place the path to the postgres executables in a file in directory /etc/paths.d
e.g.

```
/Applications/Postgres.app/Contents/Versions/latest/bin/
```

9 Homebrew

Created by Max Howell, Homebrew is described as the missing package manager for macOS. It is written in Ruby and and is git-based so you can hack it yourself. As its website says: *Homebrew installs the stuff you need that Apple didn't.*
Find it on the web at `https://github.com/Homebrew`
The homebrew site strongly recommends that you install into `/usr/local`
To install Homebrew, open an xterm terminal and enter the command:

```
cd /usr/local
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

The reason for favouring `/usr/local` is that Apple will not interfere with its contents when upgrades occur.
Homebrew sustains the metaphor of brewing with a Cellar for storage of application, bottles to contain binary applications, and taps to hold 3rd party repositories!
Homebrew provides a service similar to Linux programs: zypper, aptitude or apt, yum, rpm, etc.
The following commands have been taken from Andy Miller's website:

```
brew tap homebrew/dupes
brew tap homebrew/homebrew-php
brew tap homebrew/apache
```

9.1 Homebrew Commands

Command		What is does
brew cleanup		Remove old versions
brew commands		Produce this list
brew doctor		Check for problems
brew home		Browse Homebrews homepage
brew link		Symlink from Cellar to /usr/local/bin
brew list		List installed homebrew formulas
brew missing		List missing dependencies
brew options php55		List –options available
brew outdated		List stuff to update
brew prune		Remove broken symlinks
brew reinstall		uninstall then install
brew search		List available formulas
brew uninstall		uninstall arg
brew unlink		Remove links to /usr/local/bin
brew update		Get newest homebrew
brew uses		Show dependencies

9.2 Homebrew services

There is an external command: `brew services` which can be used to configure launchctl.

Use the command: `brew services --help`

to get a list of commands and what they do: list, run, start, stop, restart, and cleanup.

If you use sudo, homebrew operates on /Library/LaunchDaemons instead of its normal ~/Library/LaunchAgents

10 Apache

Apache is a webserver in almost universal use on Unix/Linux and on the Mac. macOS (High) Sierra comes with apache version 2.4 pre-installed. But Andy Miller of `getgrav.org` claims that it is no longer easy for developers to use the installed apache because Apple have removed some important scripts in the Sierra release. The solution is to install the full apache using Homebrew. This way we will not interfere with the Sierra version and future upgrades from Apple will not mess with the Homebrew version.

See: `httpd.apache.org`

10.1 Shutdown Apache

First shutdown the macOS apache:

```
sudo apachectl stop
sudo launchctl unload -w /System/Library/LaunchDaemons/org.apache.http.plist 2>/dev/null
```

10.2 Install Homebrew apache

```
brew install httpd24 --with-privileged-ports --with-http2
```

Upon completion of the above command, you should see a message like:

```
/usr/local/Cellar/httpd24/2.4.27: 212 files, 4.5M, built in 1m32s
```

The path above is important to note as it has to be used in the following commands

10.3 Reconfigure LaunchDaemons

We now must set up the system's LaunchDaemons to auto-start our Homebrew Apache with root privileges:

```
sudo cp -v /usr/local/Cellar/httpd24/2.4.27/homebrew.mxcl.httpd24.plist /Library/LaunchDaemons
sudo chown -v root:wheel /Library/LaunchDaemons/homebrew.mxcl.httpd24.plist
sudo chmod -v 644 /Library/LaunchDaemons/homebrew.mxcl.httpd24.plist
sudo launchctl load /Library/LaunchDaemons/homebrew.mxcl.httpd24.plist
```

There is a homebrew command which seems to do all of the above:

```
brew services start homebrew/php/php55
```

At this point your homebrew Apache should be running. Point your browser (Safari or Chrome) to localhost and you should see a simple header "`It works!`"

This worked fine for us.

If not Andy suggests that you run the command:

```
ps -aef | grep httpd
```

You should see a few httpd processes if Apache is up and running.

Try to restart Apache with:

```
sudo apachectl -k restart
```

While apache is restarting you can watch the Apache error log in a new xterm window with the command:

```
tail -f /usr/local/var/log/apache2/error_log
```

If that does not work check that you have Listen: 80 in your configuration file:

```
/usr/local/etc/apache2/2.4/httpd.conf
```

Apache is controlled via the `apachectl` command. Useful commands are:

```
sudo apachectl start
sudo apachectl stop
sudo apachectl -k restart
```

The -k option tells apachectl to restart immediately.

You can test your apache configuration with the command:

```
apachectl -t
```

11 Configure Apache

The HyperText Transfer Protocol (HTTP) defines 2 components:

1. A front end browser (such as Safari and/or Chrome on the Mac, or Firefox on Linux) which sends requests to a back end
2. A back end (a webserver such as Apache) which sends HTML pages to the front end

Over the years the backends have been improved by adding programming capabilities using CGI (Common Gateway Interface) and then PHP (Private Home Pages) which allow you to shape the html pages before supplying them to the browser.

Similarly browsers have had their specifications improved to support languages embedded in supplied html pages which the browser interprets and executes in the front end. One of these languages is Javascript (also known as ECMAscript) which current browsers support.

Apple do not expect Mac users to be software developers so the standard version of apache they supply is very rudimentary and nearly everything is disabled.

In contrast the apache installed by Homebrew is reasonably complete and can be readily configured to enable more complete implementation.

11.1 Edit httpd.conf

This section explains how to reconfigure the apache HTTP to add a local web server. After changes to any of the relevant files you will need to restart the httpd daemon:

```
sudo apachectl -k restart
```

for the changes to take effect.

The new homebrew version of Apache will be configured by editing the httpd.conf file in the directory

```
/usr/local/etc/apache2/2.4/httpd.conf
```

In contrast the original Apache installed into macOS is in `/etc/apache2` (which is really a symbolic link to `/private/etc/apache2`)

The layout within the 2 apache2 directories is different. In particular the following properties are all configurable from the httpd.conf file:

Shown here are the defaults which are relevant to us:

```
ServerRoot "/usr/local/opt/httpd24"
Listen 80
DocumentRoot "/usr/local/var/www/htdocs"
<Directory "/usr/local/var/www/htdocs">
    Options Indexes FollowSymLinks
    AllowOverRide None
    Require all granted
</Directory>
#LoadModule rewrite_module libexec/mod_rewrite.so
User daemon
Group daemon
```

11.1.1 Comments

Features which have been disabled in httpd.conf begin with a hash:#

To enable the feature, remove the #. (Using vi you position the cursor over the # and type x).

If you are changing an existing statement, it is prudent to copy the current line, add a # at the beginning of the line, then modify the copy to the new value.

This makes it easier to revert to the original by commenting out the new line and uncommenting the old.

11.1.2 User and Group

Andy Miller's site suggests adding your own Directory entry. Make a directory ~/Sites

Change the User and Group statements in httpd.conf to:

```
User John
Group staff
```

where John is your login name (not necessarily John).
You can add into the httpd.conf file a <Directory> entry for yourself (in my case John) as follows:

```
<Directory "/Users/John/Sites/">
    AllowOverride All
Options Indexes MultiViews FollowSymlinks
Require all granted
</Directory>
```

- LoadModule php5_module libexec/libphp5.so
  - The above is one example of dozens most of which are preceded by a # which comments out that module thereby disabling it
  - To enable it you remove the # (remembering to restart the httpd with the command:
  - `sudo apachectl restart`

To enable our webserver for user John we need to search for and uncomment/enable the following:

```
#LoadModule userdir_module libexec/mod_userdir.so
#LoadModule vhost_module libexec/mod_vhost_alias.so
#Include /usr/local/etc/apache2/2.4/extra/http-userdir.conf
#Include /usr/local/etc/apache2/2.4/extra/http-vhosts.conf
```

The command to do this would be:

```
cd /usr/local/etc/apache2/2.4
sudo cp httpd.conf httpd.conf.bak
sudo vi httpd.conf
# use the / command to find the above patterns and remove the leading #
# some of them are already enabled
```

11.1.3 Webserver host sites

To create apache access to your own html files via a local web host e.g. og.localhost, create a directory ~/Sites:

```
mkdir ~/Sites
mkdir ~/Sites/og
```

The Sites directory was expected in earlier versions of Mac OS X but since Lion 10.7 is no longer supplied by default. You will find that the file:

```
/usr/local/etc/apache2/extra/httpd-userdir.conf
```

contains the statement: Userdir public_html
Change this to: Userdir Sites. (The macOS equivalent has this entry)

12 PHP

PHP (originally Private Home Pages but now a recursive acronym called PHP Hypertext Processor) was created by Rasmus Lerdorf and provides a way for you put instructions into your HTML files to create dynamic content. The PHP instructions are read and parsed by the webserver which replaces them with html code and then sends the modified html file to the browser.

PHP statements are embedded in html files within tags <?php ... ?> or possibly <? ... ?>.

To use PHP, you have to install it, and then reconfigure Apache to enable it.

See:`www.php.net`

12.1 Install PHP

There are 4 separate versions of PHP: 5.5, 5.6, 7.0 and 7.1

Update your xcode implementation:

```
xcode-select --install
```

We didn't and our 1st attempt to install php produced a list of errors related to versions of the xcode commands.

Use Homebrew to download, install, and enable PHP support in Apache. You can install all 4 using the following commands:

```
brew install php55 --with-httpd24 --with-postgresql
brew unlink php55
brew install php56 --with-httpd24 --with-postgresql
brew unlink php56
brew install php70 --with-httpd24 --with-postgresql
brew unlink php70
brew install php71--with-httpd24 --with-postgresql
```

We found that the –with-postgresql option does not work complaining about missing libpq_fe.h files.

The missing files are there but in the `/Applications/Postgres.app/Contents/Versions/9.6/include` path rather than the /usr/local/etc path where Homebrew expects it.

We don't know how to tell Homebrew where to look for this. We may end up having to install a Homebrew version of PostgreSQL.

Configuration settings for the above can be tweaked by editing:

```
/usr/local/etc/php/5.5/php.ini
/usr/local/etc/php/5.6/php.ini
/usr/local/etc/php/7.0/php.ini
/usr/local/etc/php/7.1/php.ini
```

12.2 Enable PHP

Edit the httpd configuration file:

```
/usr/local/etc/apache2/2.4/httpd.conf
```

Search for

```
#LoadModule php5_module
```

Modify the LoadModule php5 statements with:

```
LoadModule php5_module /usr/local/opt/php55/libexec/apache2/libphp5.so
LoadModule php5_module /usr/local/opt/php56/libexec/apache2/libphp5.so
LoadModule php7_module /usr/local/opt/php70/libexec/apache2/libphp7.so
LoadModule php7_module /usr/local/opt/php71/libexec/apache2/libphp7.so
```

You read what homebrew advises to complete installation using the brew info command. e.g.

```
brew info php71
```

12.3 PHP Switcher

There is a script: `sphp` which allows you to switch to any of the installed PHP Modules without having to comment and uncomment the LoadModule statements in the httpd.conf file.

Install it into Homebrew's standard location: `/usr/local/bin`

```
curl -L https://gist.github.com/w00fz/142b6b1975ea6979137b963df959d11/raw > /usr/local/bin/sphp
chmod +x /usr/local/bin/sphp
```

Now re-edit the httpd.conf file by

Commenting out all the current LoadModule phpx_module statements with the following 2 statements:

```
# Brew PHP LoadModule for sphp switcher
LoadModule php5_module /usr/local/lib/libphp5.so
#LoadModule php7_module /usr/local/lib/libphp7.so
```

The sphp program will automatically handle the uncommenting and commenting of the appropriate PHP module.

Ignore the fact that the files libphp5.so and libphp8.so are not in `/usr/local/lib`.

After running the commands: sphp 55 and/or php71 the files will magically appear.

13 Virtual Hosts

Apache has provision for supporting virtual hosts. This is installed but, by default, disabled.

Remember to test your configuration after editing files: httpd.conf, httpd-vhosts.conf, and httpd-userdir.conf using the command:

```
apachectl -t
```

If apached says "Syntax OK" then you can proceed with:

```
sudo apachectl -k restart
```

To create support for virtual host for og.localhost:

13.1 Edit httpd-vhosts.conf

edit the file `/etc/apache2/extra/httpd-vhosts.conf`

```
cd /local/etc/apache2/extra
sudo cp httpd-vhosts.conf httpd-vhosts.conf.bak
sudo vi httpd-vhosts.conf
```

13.2 Add VHost entry for og.localhost

Replace the contents of httpd-vhosts.conf with:

```
#virtual Host Entry for og.localhost
<VirtualHost *:80>
 DocumentRoot ‘‘/Users/John/Sites/og’’
 ServerName og.localhost
 ErrorLog ‘‘/usr/local/var/log/apache2/og-error_log’’
 CustomLog ‘‘/usr/local/var/log/apache2/og-access_log’’ common
</VirtualHost>
```

13.3 Add to /etc/hosts

You will have to add the following lines to the file `/etc/hosts`

Add the following lines to the file.

```
#Local sites

    127.0.0.1    og.localhost
```

The command to do this:

```
cd /etc
sudo cp hosts hosts.bak
sudo vi hosts
# use the commands G i to insert the 2 lines then
:wq
```

13.4 Uncomment Include Statements

If necessary uncomment the following lines in file: `httpd.conf`

```
#Include /usr/local/etc/apache2/2.4/extra/http-vhosts.conf
#Include /usr/local/etc/apache2/2.4/extra/http-userdir.conf
```

13.5 Test in browser

Point your browser to: `og.localhost` and you should see an html page listing your full PHP configuration. Alternatively you can use the command:

```
open http://og.localhost
```

The open command will open a new tab in your browser and display the output from <phpinfo()>

14 LedgerSMB

LedgerSMB is a free software double-entry bookkeeping system which uses PostgreSQL as its database server, any standard browser for its user interface (on the Mac: Safari or Chrome), and uses LyX and TeX for high quality typeset printed output. The LedgerSMB project has forked from SQL-Ledger and intends in near future versions to use Javascript in the front end instead of perl in the backend to improve the responsiveness of the system. It is maintained by Dieter Simader.

Its developers intend to re-implement the architecture more to the MVC (model-view-controller) design pattern to make future development easier and less error-prone.

LedgerSMB is available from `ledgersmb.org`

Ledgersmb needs a fairly full and up to date version of perl. There is a website

`https://mikkel.hoegh.org/2013/07/16/installing-ledgersmb-with-homebrew-on-os-x`

which uses homebrew to install a fuller version of perl which will be used by ledgersmb.

We have tried the advice on this site but it is out of date (2013). Since then version 1.5 requires Plack to handle integration with front-end servers

Later versions of ledgersmb will replace the .pl files with javascript so the fuller perl will not be necessary.

After flirting with the install process outlined in the mikkel.hoegh.org website, we abandoned it.

We simply downloaded the tar file and extracted the contents as described below.

14.0.1 Ledgersmb

Ledgersmb supplies a file in `/usr/local/ledgersmb/conf` named `apache-vhost.conf`
Use this as the basis for adding another <VirtualHost> .. </VirtualHost> entry into the virtual host conf file: `/usr/local/etc/apache2/2.4/extra/httpd-vhosts.conf`

```
cd /usr/local/ledgersmb/conf
sudo cp apache-vhost.conf /etc/apache2/extra
```

The file looks like this:

```
# This is a 'vhost' definition file example for use with Starman/LedgerSMB
# reverse proxying.
#
# Please replace the following parameters:
#
#   * WORKING_DIR
#   * YOUR_SERVER_NAME
#   * SSL_KEY_FILE
#   * SSL_CERT_FILE
#   * SSL_CHAIN_FILE
#
#
# this block also requires mod_ssl and mod_rewrite to be enabled
# Comment out the 'Listen' and/or 'NameVirtualHost' when Apache complains
Listen 443
# NameVirtualHost is ignored by Apache 2.4
NameVirtualHost *:443
<VirtualHost *:443>
  ServerName YOUR_SERVER_NAME
  DocumentRoot WORKING_DIR/UI
  # If you own a publicly exposed server, consider submitting it
  # to the SSL security tests available at
  #    https://www.ssllabs.com/ssltest/
  SSLEngine On
  SSLCertificateFile SSL_CERT_FILE
  SSLCertificateKeyFile SSL_KEY_FILE
  SSLCertificateChainFile SSL_CHAIN_FILE
  SSLRequireSSL
  RewriteEngine On
  # Rewrite '/' URL to /login.pl script
  RewriteRule "^/$" "/login.pl" [R=301,L]
  # "hidden" fil<Directory "/usr/local/ledgersmb">     AllowOverride None     Options None     Require
  RewriteRule "^/\." - [R=404,L]
  # configuration files (those ending in '.conf'), don't exist
  RewriteRule "\.conf$" - [R=404,L]
  # Rewrite non-static content to the application backend
  RewriteCond "%{REQUEST_FILENAME}" !-d
  RewriteCond "%{REQUEST_FILENAME}" !-f
  RewriteRule "^/(.*)" "http://localhost:5762/$1" [P]
  ProxyPassReverse "/" "http://localhost:5762/"
</VirtualHost>
```

Make the following changes as suggested by the #comments in the file:

- WORKING_DIR /usr/local/ledgersmb

- YOUR_SERVER_NAME lsmb.localhost
- SSL_KEY_FILE /usr/local/etc/apache2/2.4/server.key
- SSL_CERT_FILE /usr/local/etc/apache2/2.4/server.crt
- SS:_CHAIN_FILE /usr/local/etc/apache2/2.4/server_ca.crt

14.1 Build up the apache vhost entry

We started to progressively build up the appropriate entry for httpd-vhosts.conf

```
<VirtualHost *:443>
  DocumentRoot "/usr/local/ledgersmb/"
  ServerName lsmb.localhost
  ErrorLog "/usr/local/var/log/apache2/lsmb-error_log"
  CustomLog "/usr/local/var/log/apache2/lsmb-access_log" common
</VirtualHost>
```

This failed with an error message saying we didn't have permission to access /
Eventually we fixed this by adding to the httpd.conf file the following statement:

```
<Directory ‘‘/usr/local/ledgersmb’’>
    AllowOverride None
    Options None
    Require all granted
</Directory>
```

14.2 Add lsmb.localhost to /etc/hosts

Add the entry: 127.0.0.1 lsmb.localhost
Your /etc/hosts file should now have its last 4 lines:
Local Sites

```
127.0.0.1 og.localhost
127.0.0.1 lsmb.localhost
```

We skipped the section about SSL encryption.
We copied the files index.html and index.php to /usr/local/ledgersmb and altered the index.html changing og to lsmb.
We could now test either by pointing a browser to lsmb.localhost or from the Terminal command line typing:

```
open http://lsmb.localhost
```

It worked! We saw the pages generated by PHP showing our full configuration.

14.3 Prerequisites

The current version of ledgersmb (1.5) requires the following:

- Perl 5.10 or newer (full core installation)
- Dojo an open source module Javascript library `http://dojotoolkit.org`
- Plack a perl application programming framework `plackperl.org`
  - PSGI Perl web Server Gateway Interface specification implemented by Plack
- Starman a Perl web Server Gateway Interface supporting Plack/PSGI

14.4 Download ledgersmb

14.5 Go to the ledgersmb folder

```
sudo su
cd /usr/local
tar xf /Users/John/Downloads/ledgersmb-1.5.7.tar
```

14.6 brew install cpanm

The latest version (1.5.9) of ledgersmb seems to expect cpanm. So we loaded that:

```
brew install cpanm
```

14.7 Use cpanm to install all perl dependencies

```
sudo su
cd /usr/local/ledgersmb
cpanm --quiet --notest --with-feature=starman \
    --with-feature=latex-pdf-ps \
    --installdeps .
```

Cpanm installs all the perl modules dependencies it finds as well as starman and latex-pdf-ps. On our system it installed 77 modules.

14.8 Create lsmb admin user in PostgreSQL

In our case John was the postgres superuser so the following command created a non superuser user in postgres:

```
createuser --no-superuser --createdb --login --createrole --pwprompt lsmb_dbadmin
```

The command will prompt for the password you need to create and ask you to repeat it. The createuser arguments have the following meanings:

- –no-superuser means the user cannot drop databases
- –createdb means the user can create new databases
- –login means the user must login
- –createrole means the user can assign roles (permissions) to other users
- –pwprompt means the user must supply passwords when logging in

14.9 Alter pg_hba.conf

We had trouble finding where this file was kept.

It was in /Library/Application Support/Postgres/var-9.6

This was despite our having upgraded Postgres.app to version 10.

For now at least we decided to let sleeping dogs lie and chose not to increase the sucurity of the system by changing the contents from:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all                                                     trust
host    all             all             127.0.0.1/32            trust
host    all             all             ::1/128                 trust
```

to

```
local   all                             postgres                        peer
local   all                             all                             peer
host    all                             postgres        127.0.0.1/32    reject
host    all                             postgres        ::1/128         reject
host    postgres,template0,template1    lsmb_dbadmin    127.0.0.1/32    md5
host    postgres,template0,template1    lsmb_dbadmin    ::1/128         md5
host    postgres,template0,template1    all             127.0.0.1/32    reject
host    postgres,template0,template1    all             ::1/128         reject
host    all                             all             127.0.0.1/32    md5
host    all                             all             ::1/128         md5
```

Maybe later on.
When we make the above change we will need to run the command:

```
pg_ctl restart
```

14.10 Create file ledgersmb.conf

```
cd /usr/local/ledgersmb
cp conf/ledgersmb.conf.unbuilt-dojo ledgersmb.conf
```

Edit ledgersmb.conf so that the path statement reads:

```
PATH=/usr/local/bin:/usr/local/sbin:/bin:/usr/bin:\
/Applications/Postgres.app/Contents/Versions/latest/bin
```

The [printers] section needs to be altered in our case to

```
[printers]
laser    = lpr -P HP_LaserJet_CP_1025nw
inkjet   = lpr -P HP_Officejet_Pro_8630
```

14.11 Start starman

```
starman -I lib --kisten localhost:5762 tools/starman/psgi
```

Trouble here immediately!
starman not in our PATH. We found it in

```
/usr/local/Cellar/perl/5.26.0/bin/
```

It is a perl script with header:
So we symlinked it:

```
ln -s /usr/local/Cellar/perl/5.26.0/bin/starman /usr/local/bin
```

Now it ran but produced a never-ending series of error messages.
So we were defeated at this point. We afe still working on resolving this problem!

14.12 perl *.pl

Alter the supplied *.pl files so that their first lines of `#!/usr/bin/perl` all become `#!/usr/local/bin/perl`
The perl command to do this is:

```
perl -p -i -e 's/^#!\/usr\/bin\/perl/#!\/usr\/local\/bin\/perl/' *.pl
```

To explain the above:

- -p means run perl in a loop (one invocation for each of files *.pl)
- -i (alter in place) means take each file, rename it, and put the output into the original file name
- -e means treat the following expression as one to execute
- the perl substitution command is s/original/final/. Because we have slashes in the filenames, we have to escape them using the perl (and shell) mechanism of preceding them with a backslash. The backslash does its job of preventing perl from interpreting the slash as a command delimiter and perl removes the backslash and continues processing.

14.13 run http://localhost:5762/ledgersmb/setup.pl

This should guide you through the creation and privileged management of company databases.

14.14 Run http://localhost:5762/ledgersmb/login.pl

This is the normal login for the ledgersmb application.