Homebrew for OS X

Written by Matthew Ibarra

v1.0

May 16th, 2013

https://github.com/pengii23/Homebrew-for-OS-X-doc

Contents

Homebrew for OS X

`Homebrew` for OS X

Helpful information regarding the community supported package manager for Mac OS X

You will be introduced to (one of) the community supported package managers for Mac OS X. You will be accessing said package manager via your terminal of which there are a few choices; the default, the OS X terminal, is perfectly fine for use with this document. For those curious, I post my personal terminal recommendation along with other various shell configurations at the end of the document but again, just to be clear, my terminal of choice and shell configuration is definitely NOT a requirement for using this document. Command examples will, at the least, be written for the bash terminal since that is the default. Let's see, I can't think of anything else I should mention so enough with the formalities; let's just dive right in, shall we?

First, we have to do a few things before we get to play with `Homebrew`. `Homebrew` requires that you, at the least, have the Xcode command line tools installed on your computer. Personally, I say just download and `install` Xcode as a whole since you will more than likely use it for testing your code for another class, project, or what have you, sometime in the future if you haven't already done so. Yeah, I know, it's a big download, but so it goes.

So, once Xcode is downloaded, open it, follow whatever instructions it gives you, and get past the main menu. Once past the main menu, go to the Xcode main menu on the top left > Preferences... (Command + ,). When the Preferences... window opens, go to the Downloads tab > Components, and specifically, make sure that the *Command Line Tools* are installed. Personally, I say you might as well `install` everything else while you're here, but the *Command Line Tools* are all you will need for `Homebrew`. Got it? Alright, let's get down to business.

Setup

Well, a good place to start would be getting `Homebrew` installed. So, to do so, you will want to navigate to Homebrew's website and scroll down towards the bottom where there will be a section titled **`Install Homebrew`**. The `install` command will look something like:

```
ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"
```

Simply run that command in your terminal and let `Homebrew install`.

A Little Vocabulary

Quick! While `Homebrew` is installing, let's go over some basic vocabulary to `help` you understand `Homebrew` and what it's all about:

- `Homebrew`

  The missing package manager for OS X.

- **Cellar**

  Where all Kegs are installed by `Homebrew`. Resides under your `/usr/local` directory. Also, a yummy place to eat in downtown Blacksburg. **Go Hokies!** (end-shameless-collegiate-pride-plug)

  ```
  /usr/local/Cellar
  ```

- **Keg**

  An installation prefix of a Formula.

  ```
  /usr/local/Cellar/foo/0.1
  ```

- **Formula**

  The description of a package that you would `install` with `Homebrew`.

  ```
  /usr/local/Library/Formula/foo.rb
  ```

- **Tap**

  An optional repository (git) of Formulae. We will not be discussing `taps` until the very end of the document. For beginners, I would suggest saving `taps` for a later date when you are more comfortable with `Homebrew` and the terminal in general.

  ```
  /usr/local/Library/Taps
  ```

- **Bottle**

  A binary package for `Homebrew`. They are simple gzipped tarballs of compiled binaries. Bottles can be downloaded from `Homebrew` if available or, if Bottles are available locally, those will be used as well. For those that may be curious, this can be disabled if desired but that will not be covered in this document; feel free to check out the Homebrew Wiki's *Bottles* section for more information.

  ```
  /Library/Caches/Homebrew/qt-4.8.4.mountain_lion.bottle.1.tar.gz
  ```

- **Pouring**

  The act of extracting the contents of a downloaded Bottle and moving those contents to their appropriate location(s).

  ```
  ...
  ==> Installing haskell-platform dependency: ghc
  ==> Downloading https://downloads.sf.net/project/
      machomebrew/Bottles/
      ghc-7.4.2.mountain_lion.bottle.1.tar.gz
  ######################################################## 100.0%
  ==> Pouring ghc-7.4.2.mountain_lion.bottle.1.tar.gz
  ...
  ```

Two (Relatively) Simple Yet Important Points

When using `Homebrew`, it is important to note that all Formulae are written in the language of Ruby; Regardless of whether you have written in the language of Ruby before or not, this is important to note because you can now get a general understanding of the language during your usage of `Homebrew` but perhaps more importantly, since you now know that Ruby is the language for all Formulae, that means everything `Homebrew` related is going to follow the same similar rules. Over time, this will make things a lot easier for you so don't worry about `Homebrew` using a language you may or may not have ever used before. Next `Homebrew` is also hosted solely by `git` as the source code control manager. This is nice as well because you will, if not now, but soon, have some insight as to how `Homebrew` manages its Formulae updates and where your programs are pulled down from when you use `Homebrew` to `install` programs on your OS X device. Like I said, those points may not seem all that simple right now, but try and keep them in the back of your mind as you continue to use `Homebrew` and you'll see what I mean soon enough!

Quick Post-Install Step

We need to tell the system to use programs installed by `Homebrew` (in `/usr/local/bin`) rather than the OS X default, if it exists. To do this, we need to add `/usr/local/bin` to our `$PATH` environment variable like so:

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Or you can achieve this same result while also ensuring it works for any terminal shell you use on your OS X device, now or in the future, with the following:

```
launchctl setenv PATH /usr/local/bin:$PATH
```

Basic Usage

Alright! `Homebrew` is installed! Awesome. Now what, you ask? Well, for starters, let's go over some of the basic commands, especially commands that `help` when you have to troubleshoot because they will save you from many potential headaches later on. And the `Homebrew` devs have so kindly implemented some spectacular troubleshooting commands into `Homebrew` so that you spend less time troubleshooting and more time getting your work done.

For starters, there is the useful standard `help` command

```
[Blogs@ECE2524 ~] $ brew help
```

Simple, I know, and that's how it should be; it's an excellent place for us to start since we are new to `Homebrew`. As you may have noticed, it lists many of the commands I will be covering here in the following sections. So, let's keep moving!

When you need all the details, the `man` has your back

```
[Blogs@ECE2524 ~] $ man brew
```

Like all legitimate terminal commands, `man` pages are provided for your late night reading pleasures. Actually, no, they are provided so that you can easily find that one input flag or what not for any terminal program to get whatever result you want. And, as you may have guessed, these `man` pages are provided for `Homebrew` as well. Feel free to have a look at the `man` page for `Homebrew` now if you'd like; don't worry, we'll wait. And don't worry if a lot of things look intimidating at first. `Homebrew` is exceptionally well documented and the more you use it, the less you will want to (or have to) leave your terminal.

Next, we meet the `doctor` (1/2)

```
[Blogs@ECE2524 ~] $ brew doctor
```

This command is one of the best time saving commands `Homebrew` has so kindly provided to us. When executed, the `doctor`, (`doctor` who, perhaps), checks your system for potential problems. And not only that, but if it finds a potential problem, you are provided a fair amount of information as to why the `doctor` is seeing it as a potential problem, a potential fix for the problem, and often if not always, you are able to do a "*dry run*" of the provided potential fix. Since this "*dry run*" ability is kind of super awesome and unheard of in most programs, let's give it it's own mini section inside the `doctor`'s section.

"Almost" only counts in horseshoes and hand grenades, and `Homebrew` too, with the "*dry run*" flag

When the `doctor` provides you with potential fixes to execute, you are also usually provided with information about adding some "*dry run*" flag to the command to see the results of the command without having any actual changes made to your system. This allows you to see what the `doctor`'s provided potential fix will do to your system beforehand instead of while it's happening in case the potential fix isn't appropriate for your system. This "*dry run*" ability, as the `doctor` explains clearly when he returns potential fix information back to you in your terminal, is accessed by adding the "`--dry-run`" flag to the command the `doctor` suggests, like so:

```
[Blogs@ECE2524 ~] $ brew link --overwrite --dry-run lame
Would link:
/usr/local/bin/lame
/usr/local/include/lame
...
/usr/local/lib/libmp3lame.a
/usr/local/lib/libmp3lame.0.dylib
[Blogs@ECE2524 ~] $
```

Back to the `doctor` we go (2/2)

And, best case scenario, the `doctor` tells you:

```
[Blogs@ECE2524 ~] $ Your system is ready to brew.
```

When's the last time your standard compiler did that for you? Yeah, that's what I thought. And to hopefully make things a little more clear on what you can expect from the `doctor`, here is an example of some output I received when running `brew doctor` on my system while writing up this document:

```
[Blogs@ECE2524 ~] $ brew doctor
Warning: You have unlinked kegs in your Cellar
Leaving kegs unlinked can lead to build-trouble and cause brews that
depend on those kegs to fail to run properly once built. Run
`brew link` on these:

    faac
    lame
[Blogs@ECE2524 ~] $
```

So that's the `doctor` in a nutshell. We will continue on but keep the `doctor` in mind for the next four `Homebrew` commands because there are the four commands that are commonly suggested you to run whenever the `doctor` shows you having potential errors or when you run into issues when installing new Formulae / packages on your system.

Am I running the latest and greatest!? `brew update`

```
[Blogs@ECE2524 ~] $ brew update
```

So, as the command suggests, this updates your `Homebrew` Formulae. Pretty self explanatory, yeah? Well, just to note, if the `doctor` tells you that you have some potential errors, this command, the `brew update` command, is the first command you should run. Having everything up-to-date is like blowing into the bottom of an N64 game cartridge: it just works. I may have just dated myself with that statement. Here are a couple examples of potential terminal output after running the `brew update` command, the first being the best case scenario where everything checks out:

```
[Blogs@ECE2524 ~] $ Already up-to-date.
```

And here is the output where `Homebrew` updates some of its Formulae. Note that this does **NOT** have anything to do with whether or not the package or packages is/are installed on your system. Think of it like the app store for your preferred smart device; regardless of whether the app is installed on your preferred smart device, apps in the app store still `update` to new versions. This command, `brew update`, is updating all "apps", in this case Formulae or Formulae definitions, in the "`Homebrew` app store", which is just `Homebrew` itself. The command responsible for getting your system installed packages up-to-date will be covered in the next command section and will hopefully make this all make more sense as a whole so bear with me. In the mean time, here is `brew update` in action:

```
[Blogs@ECE2524 ~] $ brew update
```

```
Updated Homebrew from 18723f46 to c2cc5816.
==> New Formulae
cityhash
==> Updated Formulae
gmp
==> Deleted Formulae
gsutil
[Blogs@ECE2524 ~] $
```

"Updated `Homebrew` from ######## to ########.", "New Formulae", "Updated Formulae", and "Deleted Formulae" are the four statements one can potentially see as output after running the `brew update` command successfully.

An `update`'s cool and all, but an `upgrade`, now that's what's up

```
[Blogs@ECE2524 ~] $ brew upgrade
```

Similar to the `update` command, the `upgrade` command gets your system up-to-date. This time, however, all your Formulae, the Formulae installed on your system, are upgraded to the most recent version `Homebrew` has to offer. `Homebrew` knows what the latest and greatest versions of these Formulae are because that is what the previous command we covered, `brew update`, did. In short, `brew update` updates the Formulae definitions and `brew upgrade` reinstalls the new Formulae if they are currently present on your system thus replacing the Formulae on your system `Homebrew` found to be outdated. Here are a couple examples of potential terminal output after running `brew upgrade` starting with the best case scenario:

```
[Blogs@ECE2524 ~] $ brew upgrade
[Blogs@ECE2524 ~] $
```

Yup, that's right. Nothing happens if everything currently installed on your system by `Homebrew` is still the most current and up-to-date version according to `Homebrew`. Next, an example of some packages being upgraded:

```
[Blogs@ECE2524 ~] $ brew upgrade
==> Upgrading 5 outdated packages, with result:
node 0.10.7, python 2.7.5, python3 3.3.2, ruby 2.0.0-p195, vim 7.3.969
==> Upgrading node
==> Downloading http://nodejs.org/dist/v0.10.7/node-v0.10.7.tar.gz
######################################################################## 100.0%
==> Patching
...
/usr/local/Cellar/node/0.10.7: 1055 files, 15M, built in 2.1 minutes
```

```
==> Upgrading python
==> Downloading http://www.python.org/ftp/python/2.7.5/
Python-2.7.5.tar.bz2
######################################################################## 100.0%
...
==> Downloading https://pypi.python.org/packages/source/d/
distribute/distribute-0.6.40.tar.gz
######################################################################## 100.0%
...
==> Downloading https://pypi.python.org/packages/source/p/pip/
pip-1.3.1.tar.gz
######################################################################## 100.0%
...
==> Summary
/usr/local/Cellar/python/2.7.5: 5200 files, 80M, built in 2.1 minutes
==> Upgrading python3
==> Downloading http://python.org/ftp/python/3.3.2/Python-3.3.2.tar.bz2
######################################################################## 100.0%
...
==> Downloading https://pypi.python.org/packages/source/d/
distribute/distribute-0.6.40.tar.gz
######################################################################## 100.0%
...
==> Downloading https://pypi.python.org/packages/source/p/pip/
pip-1.3.1.tar.gz
######################################################################## 100.0%
...
==> Summary
/usr/local/Cellar/python3/3.3.2: 4700 files, 92M, built in 2.3 minutes
==> Upgrading ruby
==> Downloading http://ftp.ruby-lang.org/pub/ruby/2.0/
ruby-2.0.0-p195.tar.bz2
######################################################################## 100.0%
...
==> make install
==> Caveats
NOTE: By default, gem installed binaries will be placed into:
  /usr/local/opt/ruby/bin

You may want to add this to your PATH.
==> Summary
/usr/local/Cellar/ruby/2.0.0-p195: 878 files, 19M, built in 3.1
minutes
==> Upgrading vim
==> Cloning https://vim.googlecode.com/hg/
Updating /Library/Caches/Homebrew/vim--hg
```

```
...
44 files updated, 0 files merged, 0 files removed, 0 files unresolved
==> Checking out tag v7-3-969
...
==> make install prefix=/usr/local/Cellar/vim/7.3.969 STRIP=/usr/
bin/true
/usr/local/Cellar/vim/7.3.969: 1536 files, 24M, built in 47 seconds
[Blogs@ECE2524 ~] $
```

Time to clean your room; `cleanup` time

With that previous `upgrade` command, you (may) have some new versions of your Formulae. So, occasionally, you want to clean out your Cellar of those old Kegs so you, in a sense, have room for new Kegs. Although your Mac will undoubtedly never run into a situation where your Kegs in your Cellar are taking up such an enormous amount of space that such a thing would be noticeable, but hey, sometimes it's nice to pick up after yourself and more importantly, make sure that the old Formulae don't interfere with the newer Formulae you're using now. That's why this is the third of four commands that tend to get run when the `doctor` shows you potential errors with your Cellar or when you run into issues when installing new Formulae / packages on your system. Here are a couple of examples for you here showing `brew cleanup` in action, the first being, yup, you guessed it, the best case scenario:

```
[Blogs@ECE2524 ~] $ brew cleanup
Pruned 0 dead formula
Tapped 0 formula
[Blogs@ECE2524 ~] $
```

Or, if you've tapped a tap:

```
[Blogs@ECE2524 ~] $ brew cleanup
Pruned 0 dead formula
Tapped 41 formula
[Blogs@ECE2524 ~] $
```

No dead Formulae to prune and just making sure your taps are all solid by re-tapping them. Now, say you hadn't run `brew cleanup` in a good while. What would that look like, you ask? Well, you're in luck because I hadn't run `brew cleanup` in maybe a month or so. So, to answer the question you may or may not have actually asked while reading this, it results in a fair amount of information about the `cleanup` being output to your terminal, believe it or not, so I'll be shortening that output some, but you'll get the idea:

```
[Blogs@ECE2524 ~] $ brew cleanup
Removing: /usr/local/Cellar/git/1.8.2.2...
Removing: /usr/local/Cellar/mobile-shell/1.2.3...
Removing: /usr/local/Cellar/mongodb/2.2.3-x86_64...
Removing: /usr/local/Cellar/node/0.8.21...
Removing: /usr/local/Cellar/tmux/1.7...
Removing: /usr/local/Cellar/vim/7.3.943...
Removing: /usr/local/Cellar/weechat/0.3.9.2...
Removing: /usr/local/Cellar/wine/1.4...
Removing: /usr/local/Cellar/wireshark/1.8.5...
...
Removing: /Library/Caches/Homebrew/emacs-24.2.tar.bz2...
Removing: /Library/Caches/Homebrew/git-1.8.2.2.tar.gz...
Removing: /Library/Caches/Homebrew/mobile-shell-1.2.3.tar.gz...
Removing: /Library/Caches/Homebrew/node-0.8.21.tar.gz...
Removing: /Library/Caches/Homebrew/python3-3.3.0.tar.bz2...
Removing: /Library/Caches/Homebrew/ruby-1.9.3-p385.tar.gz...
Removing: /Library/Caches/Homebrew/tmux-1.7.tar.gz...
Removing: /Library/Caches/Homebrew/weechat-0.3.9.2.tar.bz2...
Removing: /Library/Caches/Homebrew/wireshark-1.8.5.tar.bz2...
Pruned 0 dead formula
Tapped 41 formula
Pruned 4 symbolic links and 23 directories from /usr/local
[Blogs@ECE2524 ~] $
```

As you can see, it's good to run `brew cleanup` sometimes as it cleans some of the cobwebs and empty Kegs out of your Cellar. And with that, on to the next command!

Fun fact: Mac OS X has a hierarchical `link` to BSD kernel roots; `brew link`

So, for the fourth of four commands that are commonly used when the `doctor` tells us we have some potential issues with our `Homebrew`, I present to you `brew link`. Usually, you won't have to run this command unless `Homebrew` specifically says that you need to give it a hand and run the `brew link` command to `link` specific Formulae. Linking using `brew link` creates symbolic links for `Homebrew` linking your Formulae from where `Homebrew` installs them (`/usr/local/Cellar/...`) to, in most cases, where Mac (and all GNU/Linux / Unix systems) `install` binaries for your programs (`/usr/local/bin/...`).

```
[Blogs@ECE2524 ~] $ brew link FORMULA...
[Blogs@ECE2524 ~] $
```

So, for two examples again, the first of which will be the programming running successfully:

```
[Blogs@ECE2524 ~] $ brew link lame
Linking /usr/local/Cellar/lame/3.99.5... 20 symlinks created
[Blogs@ECE2524 ~] $
```

Success! The symlinks for all necessary components of the program have been created! Now, let's look at the way it really went down when I ran this command:

```
[Blogs@ECE2524 ~] $ brew link lame
Linking /usr/local/Cellar/lame/3.99.5... Warning: Could not link lame.
Unlinking...

Error: Could not symlink file: /usr/local/Cellar/lame/3.99.5/bin/lame
Target /usr/local/bin/lame already exists. You may need to delete it.
To force the link and overwrite all other conflicting files, do:
  brew link --overwrite formula_name

To list all files that would be deleted:
  brew link --overwrite --dry-run formula_name
[Blogs@ECE2524 ~] $
```

As you can see in the above terminal output, at the location I am trying to create my symlink, `/usr/local/bin`, `lame` already exists: `/usr/local/bin/lame`. This is most likely due to an old symlink existing in `/usr/local/bin` or potentially due to me having previously installed the program, directly or as the result of another program having it as a dependency, without using `Homebrew` to do so. Since this is case, `Homebrew` has no safe means of executing `cleanup` on any old symlinks it didn't make itself and thankfully won't risk deleting this symlink without my authorization. I mean, after all, other programs in my system could vitally depend on this symlink or specific version of this installed program. Although maybe a little annoying now because I know it's not a vitally important program, this kind of caution is really important and well implemented in `Homebrew`. The last thing I want my package manager (or any program, really) doing is going around deleting things without my prior authorization that could potentially be important to other programs I have installed, especially if those programs are crucial to my workflow. In this case, this symlink or installed program is not a crucial component of my system and I know it's safe to `remove` it and relink to the new version `Homebrew` is trying to `link` currently. However, let's step back for a second and pretend we are in a scenario where we are *relatively* confident that it's safe to `remove`, but we are being extra careful and want to be absolutely certain. `Homebrew` is awesome enough to give us a means of being absolutely certain. Do you remember what that means is? Yes, we have covered it already, I assure you. That's correct! We can use the "*dry run*" flag to have a look at what would be overwritten by us relinking this new version of the program we are installing with `Homebrew`. As the `doctor` suggested in the previous code terminal segment, you can `list` all files that would deleted by

forcing the `link` and overwriting all other conflicting files. This is our second example where `Homebrew`s awesome troubleshooting abilities, specifically the ability to do "*dry run*"s on a command the `doctor` suggests you run to fix a potential problem on your system:

```
[Blogs@ECE2524 ~] $ brew link --overwrite --dry-run lame
Would link:
/usr/local/bin/lame
/usr/local/include/lame
/usr/local/share/man/man1/lame.1
/usr/local/share/doc/lame/html/vbr.html
/usr/local/share/doc/lame/html/usage.html
/usr/local/share/doc/lame/html/ms_stereo.html
/usr/local/share/doc/lame/html/list.html
/usr/local/share/doc/lame/html/links.html
/usr/local/share/doc/lame/html/introduction.html
/usr/local/share/doc/lame/html/index.html
/usr/local/share/doc/lame/html/history.html
/usr/local/share/doc/lame/html/detailed.html
/usr/local/share/doc/lame/html/contributors.html
/usr/local/share/doc/lame/html/contact.html
/usr/local/share/doc/lame/html/cbr.html
/usr/local/share/doc/lame/html/abr.html
/usr/local/share/doc/lame/html/about.html
/usr/local/lib/libmp3lame.dylib
/usr/local/lib/libmp3lame.a
/usr/local/lib/libmp3lame.0.dylib
[Blogs@ECE2524 ~] $
```

So, overwriting all these conflicting files and linking to the most recent version would result in what looks to be 20 links being removed and 20 new links being created. I know that looks intimidating because I certainly don't know the specifics about all (or any) of these files for the program `lame`, but most look like HTML files and the first two look like the common places where the binary gets linked so it can be used by other programs so I'll go ahead and force the overwrite of these 20 symlinks and see what happens. Here's what the output looks like:

```
[Blogs@ECE2524 ~] $ brew link --overwrite lame
Linking /usr/local/Cellar/lame/3.99.5... 20 symlinks created
[Blogs@ECE2524 ~] $
```

Excellent! Success and no issues reported back by `Homebrew` which is always a plus. Alright, so, we know how to handle issues when trying to `install` commands and when the `doctor` tells us we may have potential conflicts, but

none of that is any good until we know how to `install` Formulae in the first place! So, without further a due, let's `install` some Formulae!

`$SHELL`'s OVER 9000 – `brew install`

Alright, time to unleash all the power and possibilities `Homebrew` has to offer! Excited!? Good. You should be! You are about to give your Mac power it hasn't ever previously experienced, certainly not for free and certainly not as easily as `Homebrew` has made it. So, what program should we `install`? Well, right now, we are going to `install` a few to get the blood flowing: vim, emacs, and subversion (svn). I am choosing these because you should try out vim and emacs if you haven't already done so (but if it is your first time with either of them, wait until you have some free time to really try them out and aren't in the middle of learning something else like `Homebrew`). And I am having you `install` subversion because I wanted you to `install` three things instead of two. Actually, that's not the real reason. You'll find out the real reason in a bit so just bear with me in the mean time, ok? Awesome, thank you. But alright, enough chatter! Let's get rolling with that `install` thing of which is super simple! The general formula for installing Formulae with `Homebrew` is as follows:

```
brew install FORMULA...
```

As you can see, it's incredibly simple to `install` various Formulae with `Homebrew`. And just to clarify, the "..." after "FORMULA" signifies that you can have one or more Formulae after `install` so long as they are separated by at least one space character. Let's try `brew install` out with vim, emacs, and subversion now:

```
[Blogs@ECE2524 ~] $ brew install vim emacs subversion
==> Installing vim dependency: mercurial
==> Downloading http://mercurial.selenic.com/release/mercurial-2.6.tar.gz
######################################################################## 100.0%
==> make local
...
==> Summary
/usr/local/Cellar/mercurial/2.6: 544 files, 7.2M, built in 16 seconds
==> Installing vim
==> Cloning https://vim.googlecode.com/hg/
...
==> make
==> make install prefix=/usr/local/Cellar/vim/7.3.944 STRIP=/usr/bin/true
/usr/local/Cellar/vim/7.3.944: 1533 files, 24M, built in 2.7 minutes
==> Downloading http://ftpmirror.gnu.org/emacs/emacs-24.3.tar.gz
######################################################################## 100.0%
```

```
...
==> make
==> make install
...
==> Summary
/usr/local/Cellar/emacs/24.3: 3843 files, 96M, built in 11.5 minutes
==> Downloading http://www.apache.org/dyn/closer.cgi?path=subversion/
subversion-1.7.9.tar.bz2
==> Best Mirror http://www.eng.lsu.edu/mirrors/apache/subversion/
subversion-1.7.9.tar.bz2
######################################################################## 100.0%
...
==> make
==> make install
...
==> Summary
/usr/local/Cellar/subversion/1.7.9: 98 files, 7.4M, built in 2.4 minutes
```

So, did you notice that emacs is a rather large download? Yes, tons of fun, I know. Did you also notice that `Homebrew` installed the necessary dependencies for you before trying to `install` the Kegs you requested? Pretty neat, yeah? And who doesn't love the little pint of `Homebrew` poured specially for you upon successful installation of each of your requested Kegs!? The little things in life, I tell you what, you just have to savor them! Anyways, moving on!

What!? You're tired of vim AND emacs already!? No, of course you aren't! Whew, thank goodness. What blasphemy such a thing would be! But you are, on the other hand, probably tired of subversion because I made you `install` it on your precious Mac device for no apparent reason. Alright, I confess, the only reason I had you `install` subversion on your system was to have it as something to `uninstall` in the next section with the `remove` command.

Easy come, easy go: `brew remove` || `brew uninstall`

So, time to `remove` subversion from your system:

```
[Blogs@ECE2524 ~] $ brew remove subversion
Uninstalling /usr/local/Cellar/subversion/1.7.9...
[Blogs@ECE2524 ~] $
```

And just to prove a point, `brew uninstall` will have the same output as `brew remove` when uninstalling subversion:

```
[Blogs@ECE2524 ~] $ brew uninstall subversion
Uninstalling /usr/local/Cellar/subversion/1.7.9...
[Blogs@ECE2524 ~] $
```

Let's cover one more thing with `remove`/`uninstall` before we move on to our last two commands; when you try to `remove` a Keg that doesn't exist / isn't currently installed, you will get an error similar to the following:

```
[Blogs@ECE2524 ~] $ brew remove hg
Error: No such keg: /usr/local/Cellar/hg
[Blogs@ECE2524 ~] $
```

Whatcha got for me, `Homebrew-ski?` `brew search`

So, you have a few other things you want to `install`, huh? But, darn it, what exactly is the right format for the Formulae you want to `install`? Well, don't look at me! I just work here... But in all seriousness, we don't have to leave this terminal window to figure out the answer to your question(s). For example, I wonder what kind of things they have relating to the `bash` (which is the default shell for pretty much everything except Windows):

```
[Blogs@ECE2524 ~] $ brew search bash
bash         bash-completion  bashdb       bashish      calabash
[Blogs@ECE2524 ~] $
```

Ohhh, what's that? `bash-completion`? You all might want to check that out and give it a whirl; sounds like it could be pretty handy, don't you think? ;)

And for you `zsh` people out there, we can try something similar:

```
[Blogs@ECE2524 ~] $ brew search zsh
zsh      zsh-completions  zsh-lovers
[Blogs@ECE2524 ~] $
```

Ohhh, `zsh-completions`! Shiny! And apparently Virginia isn't the only place for lovers.

Want to see everything you could possibly `install`? Well, if you type `brew search` with no `search` criteria, the command simply lists everything you could possibly `install`. Just a heads up, the output is rather large.

```
[Blogs@ECE2524 ~] $ brew search
... ... ... ... ... ... ... ... ... ...
... *BOOM*  ... *BOOM*  ... *BOOM*  ...
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
... *BOOM*  ... *BOOM*  ... *BOOM*  ...
... ... ... ... ... ... ... ... ... ...
[Blogs@ECE2524 ~] $
```

ALL THE THINGS!! Wait, I forget what I've installed... `brew list`

Think you might have gone a little overboard with the `brew search` command? That's totally up to interpretation, but regardless, it's ok. `Homebrew` has provided a simple means to see what Kegs are currently installed so we can have a look and, if the situation calls for it, `remove` some unnecessary installs; certainly not a requirement though. To see the Kegs you have currently installed on your system, simply run the following command:

```
[Blogs@ECE2524 ~] $ brew list
... ... ... ... ... ... ... ... ... ... ... ... ... ...
... *Manageable*    ... *Length*    ...   *List*    ...
... ... ... ... ... ... ... ... ... ... ... ... ... ...
[Blogs@ECE2524 ~] $
```

Your mission, should you choose to accept it

Alright, time for you to get down and dirty ladies and gentlemen. Please `install` the following programs using `Homebrew` if you have not already done so. These helpful tools can and will aid you in reaching your current and future goals so go wild:

- `ack`
- `cmake`
- `coreutils`
- `emacs`
- `git`
- `gnu-sed`
- `grep`
  - **Should** already be available on OS X.
  - If, for some reason, `grep` is not installed or useable, do the following:
    - `brew tap homebrew/homebrew-dupes`
    - `brew install grep`
- `haskell-platform`
  - `brew install haskell-platform`
  - `echo 'export PATH=$HOME/.cabal/bin:$PATH' >> ~/.bashrc`
  - `cabal update`
  - `vim ~/.cabal/config`
    - "– library-profiling: False" to "library-profiling: True".
  - `cabal install -fhighlighting pandoc`
- `irssi`

- `mobile-shell`
- `mongodb`
- `node`
- `openssl`
- `perl`
- `python`
- `ruby`
- `tmux`
- `vim`
  - May also want to consider `macvim` as well (`mvim`)
- `weechat`
- `zsh`
- `zsh-completions`
- `zsh-lovers`

Also, `install` MacTeX from the MacTeX MacTeX website and since the download is absolutely insane for the entire MacTeX.pkg, I believe that all you will really need is what is under the "Smaller Packages" `link` below the "MacTeXtras.zip" `link`. So, navigate to the MacTeX Smaller Packages section and download the "mactex-additions.pkg". I also strongly suggest you download whatever you decide to download via the *torrent network*. For this, I recommend "Transmission" (available via `Homebrew` and the Transmission Transmission website).

I suggest MacTeX because it's great for making crisp, clean, and well organized documents. Regardless of whether you agree with me on the organization of this document, you have to admit that it is crisp and clean because I did indeed use MacTeX with pandoc-markdown to make this document.

Thanks for brewin'

Not to start this little section on a "bad note", but I'm sorry to say that despite `Homebrew` being one of the most powerful programs on your Mac, the command `brew coffee` did not result in anything but an unknown command error message so don't get your hopes up:

```
[Blogs@ECE2524 ~] $ brew coffee
Error: Unknown command: coffee
[Blogs@ECE2524 ~] $
```

That being said, the command `brew beer` produces a fun output so check that out if you're interested. Please drink responsibly:

```
[Blogs@ECE2524 ~] $ brew beer
... ... ... ... ...
... *Surprise*  ...
... ... ... ... ...
Thanks for brewin'
[Blogs@ECE2524 ~] $
```

Misc. Resources, Suggestions, Etc.

- New font:
  - Inconsolata website
- New terminal:
  - iTerm2 website
- Remap CAPS LOCK key to "Control" key (or "Escape" key via PCKeyBoardHack)
  - System Preference > Keyboard > Modifier Keys... (Bottom Right) > Caps Lock Key: > Select Dropdown Option "^ Control".
- New color theme:
  - Solarized website
- Unhide the ~/Library folder"
  - `chflags nohidden ~/Library`
- Some excellent 'dotfiles':
  - Mathias's dotfiles website
  - Janus's Vim dotfiles website: Vim Distribution
  - The Vim Configuration of Champions website (not my words, that's what it's actually titled and with that being said, it's a really good vim config)
  - YADR website
    - oh-my-zsh website: an open source, community-driven framework for managing your ZSH configuration; bundled with a ton of helpful functions, helpers, plugins, themes, and few things that make you shout... "OH MY ZSHELL!"
    - Prezto website: "Instantly Awesome Zsh"; enriches the zsh command line interface experience.
    - Vundle website: Short for *Vim bundle* and is a fantastic `Vim` plugin manager.
    - Uses `macvim` instead of `vim` that comes with OS X:
      - `brew install macvim`

        - `alias vim='/usr/local/Cellar/macvim/7.3-66/bin/mvim -v'`
    - Just Another Vim Configuration website
- YouCompleteMe website: a code-completion engine for Vim.
- `tmux` is amazing for multi-window programming, especially with `vim`.
    - Note: `tmux` is integrated / built directly into iTerm2 for ease of use and is a Keg available for installation via `Homebrew`:
        - `brew install tmux`
- Sublime Text 2 website: Sublime Text 3 expected late 2013.
    - Oh, the wonderful, wonderful things you can do with this program... This is a whole pandoc-markdown document in itself. That being said, take a look at the program anyways because one license purchase gives you the ability to `install` it on OS X, Windows x86, Windows x64, GNU/Linux x86, AND GNU/Linux x64. It's everything Textmate website was in its glory days and so much more. I can't recommend Sublime Text 2 enough even with all the amazing `vim` / `mvim` customization files provided above.
        - That being said, Sublime Text 2 does need to work on its `vim mode`, but regardless, it's a most excellent program that continues to get better and better every day. Check it out. You won't regret it.

`Homebrew` Resource(s)

- Braumeister website: Online package browser for `Homebrew`.
- [`Homebrew` for OS X doc website](https://github.com/pengii23/Homebrew-for-OS-X-doc ""`Homebrew` for OS X" doc") – the github repo `link` for this document.

Bonus Section(s)

Bonus Round #1: Installing `coffee-script`

Just a little warm up before bonus round #2 and because `coffee-script` is awesome in my opinion. Alright, let's do this. First, we need to `brew install node`:

```
[Blogs@ECE2524 ~] $ brew install node
==> Downloading http://nodejs.org/dist/v0.10.6/node-v0.10.6.tar.gz
######################################################################## 100.0%
==> Patching
```

```
patching file tools/gyp/pylib/gyp/xcode_emulation.py
==> ./configure --prefix=/usr/local/Cellar/node/0.10.6
==> make install
==> Caveats
Homebrew installed npm.
We recommend prepending the following path to your PATH environment
variable to have npm-installed binaries picked up:
  /usr/local/share/npm/bin
==> Summary
/usr/local/Cellar/node/0.10.6: 951 files, 14M, built in 116 seconds
[Blogs@ECE2524 ~] $
```

And next, as `Homebrew` so kindly recommended to us, we should prepend `/usr/local/share/npm/bin` to our PATH environment variable so that npm-installed binaries get picked up. We can do this by adding `export PATH=/usr/local/share/npm/bin:$PATH` to our respective .rc files, meaning, for `bash` users, that is your .bashrc file which is located under your home folder on your Mac. And for `zsh` users, that is your .zshrc file which is also located under your home folder on your Mac. I will show the commands for opening both .rc files below using `vim` as the text editor of choice, but you can use `nano` or whatever your text editor of choice is. Here are the commands, for `bash` users and `zsh` users respectively:

`bash`:

```
[Blogs@ECE2524 ~] $ vim ~/.bashrc

##############################################
## In the .bashrc file, add the following: ##
##############################################

export PATH=/usr/local/share/npm/bin:$PATH

##############################################
## When finished, save and quit the editor ##
##############################################

[Blogs@ECE2524 ~] $
```

`zsh`:

```
[Blogs@ECE2524 ~] $ vim ~/.bashrc

##############################################
## In your .zshrc file, add the following: ##
```

```
##############################################

export PATH=/usr/local/share/npm/bin:$PATH

##############################################
## When finished, save and quit the editor ##
##############################################

[Blogs@ECE2524 ~] $
```

Don't forget that you'll have to close and reopen your terminal in order for what you added to your respective .rc file to take affect. So, just to be safe, let's go ahead and do that now and then we'll finish up with that actual installation of `coffeescript`.

Alright, last, but not least, installing `coffeescript`:

```
[Blogs@ECE2524 ~] $ npm install -g coffee-script
npm http GET https://registry.npmjs.org/coffee-script
npm http 304 https://registry.npmjs.org/coffee-script
/usr/local/share/npm/bin/coffee -> /usr/local/share/npm/lib/
node_modules/coffee-script
/bin/coffee
/usr/local/share/npm/bin/cake -> /usr/local/share/npm/lib/
node_modules/coffee-script
/bin/cake
coffee-script@1.6.2 /usr/local/share/npm/lib/node_modules/coffee-script
[Blogs@ECE2524 ~] $
```

And that's that! Oh, and if you want to know more about CoffeeScript and see why it's the bomb dot com and why it's also money times twenty, check out this CoffeeScript website.

Also, I recently found a neat tool called `nvm` (Node Version Manager) and the following installation along with more information about the tool is (mostly) covered NVM website.

First, we need to get the `install.sh` script using `cURL`:

```
[Blogs@ECE2524 ~] $ curl https://raw.github.
com/creationix/nvm/master/install.sh | sh
*** *** *** *** *** *** *** *** ***
*** NVM *** INSTALL *** MAGIC   ***
*** *** *** *** *** *** *** *** ***
[Blogs@ECE2524 ~] $
```

If it installed successfully, it will have put an `NVM` specific line in your `.bash_profile` file on your computer under your home directory. `NVM` states the path in the output. If you are using `zsh`, you will need to move the line `NVM` added to the .bash_profile file to your .zshrc file.

After that, restart your terminal so that the .zshrc change gets recognized and then try `nvm -v` to ensure everything is installed correctly. The output this command will give if installed correctly will `help` you with playing around with `nvm` and for installing whatever version of `node` you want. Now, go play kids!

Bonus Round #2: Updating OS X's `GCC` to either a newer `GCC` or LLVM `clang`

Let's walk through the exploration process of how I determined how to `upgrade` your `GCC` version on OS X since the `GCC` version on OS X is a bit behind that which GNU/Linux systems are using right now. Have no fear though for, like I said, we will get that worked out in no time.

First, the websites for `GCC` website and `clang` LLVM website, respectively. And now that that's out of the way, let's get to it! So, assuming you've gone through this whole document at least once, we should be well equipped to tackle this task, or at the least, have this be an excellent problem that applies that which we've recently learned to a "real life" problem to really drive those commands and skills home (or should I say, `Homebrew`). And with the lame jokes out of the way, let's `update` our OS X `GCC`.

Alright, first, I want to see if `Homebrew` even has a Formulae for `GCC` so let's run `brew search GCC` and find out:

```
[Blogs@ECE2524 ~] $ brew search GCC
GCC is now maintained in homebrew-versions, with major version
number in formula name as suffix. Please tap using:

    brew tap homebrew/versions

and then install GCC based on its version, e.g., 'brew install gcc47'.
[Blogs@ECE2524 ~] $
```

Well, it looks like `GCC` is now maintained in the `homebrew/versions` tap. I guess we should tap into `homebrew/versions`. That was nice of them to literally tell us exactly what to type; let's type the command and see what happens:

```
[Blogs@ECE2524 ~] $ brew tap homebrew/versions
Cloning into '/usr/local/Library/Taps/homebrew-versions'...
remote: Counting objects: 899, done.
remote: Compressing objects: 100% (487/487), done.
```

```
remote: Total 899 (delta 515), reused 776 (delta 412)
Receiving objects: 100% (899/899), 217.83 KiB | 64 KiB/s, done.
Resolving deltas: 100% (515/515), done.
Tapped 85 formula
[Blogs@ECE2524 ~] $
```

And we're in. Let's try that `search` command again now that we're tapped in to the `homebrew/versions` tap:

```
[Blogs@ECE2524 ~] $ brew search GCC
gcc43       gcc44       gcc45       gcc46
gcc47       gcc48       gcc49       llvm-gcc28

If you meant `GCC' precisely:

GCC is now maintained in homebrew-versions, with major version
number in formula name as suffix. Please tap using:

    brew tap homebrew/versions

and then install GCC based on its version, e.g., 'brew install gcc47'.
[Blogs@ECE2524 ~] $
```

Alright, sweet. At the time of the initial writing of this document, OS X has `GCC` 4.2 installed. `Homebrew` gives us the option of installing any `GCC` from 4.3 up to the developmental version of `GCC`, `GCC` 4.9 at the time of the initial writing of this document. If you are going to `install GCC`, you know what to do:

```
[Blogs@ECE2524 ~] $ brew install gcc48
==> Installing gcc48 dependency: mpfr
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/
mpfr-3.1.2.mountain_lion.bottle.tar.gz
######################################################################## 100.0%
==> Pouring mpfr-3.1.2.mountain_lion.bottle.tar.gz
/usr/local/Cellar/mpfr/3.1.2: 23 files, 3.3M
==> Installing gcc48 dependency: libmpc
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/
libmpc-1.0.1.mountain_lion.bottle.
######################################################################## 100.0%
==> Pouring libmpc-1.0.1.mountain_lion.bottle.tar.gz
/usr/local/Cellar/libmpc/1.0.1: 9 files, 292K
==> Installing gcc48 dependency: isl
==> Downloading http://www.kotnet.org/~skimo/isl/isl-0.11.2.tar.bz2
######################################################################## 100.0%
==> ./configure --prefix=/usr/local/Cellar/isl/0.11.2
```

```
==> make install
/usr/local/Cellar/isl/0.11.2: 52 files, 3.0M, built in 39 seconds
==> Installing gcc48 dependency: cloog
==> Downloading http://www.bastoul.net/cloog/pages/download/
count.php3?url=./cloog-0.18.0.tar.gz
######################################################################## 100.0%
==> ./configure --prefix=/usr/local/Cellar/cloog/0.18.0
--with-isl-prefix=/usr/local/opt/isl
==> make install
/usr/local/Cellar/cloog/0.18.0: 31 files, 532K, built in 104 seconds
==> Installing gcc48
==> Downloading ftp://gcc.gnu.org/pub/gcc/releases/gcc-4.8.0/
gcc-4.8.0.tar.bz2
######################################################################## 100.0%
==> ../configure --build=x86_64-apple-darwin12.3.0 --prefix=/usr/
local/Cellar/gcc48/4.8.0/gcc --dataroo
==> make bootstrap
==> make install
/usr/local/Cellar/gcc48/4.8.0: 962 files, 90M, built in 27.9 minutes
[Blogs@ECE2524 ~] $
```

And if you are looking to `install` LLVM `clang`, a competitive alternative to `GCC` that is looking to replace `GCC`, then you also know what to do:

```
[Blogs@ECE2524 ~] $ brew install gcc48
==> Downloading https://downloads.sf.
net/project/machomebrew/Bottles/
llvm-3.2.mountain_lion.bottle.1.tar.gz
######################################################################## 100.0%
==> Pouring llvm-3.2.mountain_lion.bottle.1.tar.gz
==> Caveats
Extra tools and bindings are installed in /usr/local/Cellar/llvm/
3.2/share/llvm and /usr/local/Cellar/llvm/3.2/share/clang.

If you already have LLVM installed, then "brew upgrade llvm" might
not work.
Instead, try:
    brew rm llvm && brew install llvm
==> Summary
/usr/local/Cellar/llvm/3.2: 628 files, 110M
[Blogs@ECE2524 ~] $
```

And that's that. On to the next!