

18 – More Package Management, and some distros

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

March 4, 2019

Cornell University

Table of Contents

1. Homebrew time
2. Other Managers
3. Demo: the language-specific package managers I have installed.
4. Choosing a Linux Distro, revisited
5. Back to desktop environments

Homebrew time

OSX Package Management: Install **brew** on your own

- Sitting in class right now with a Mac?
- **DON'T DO THIS IN CLASS.** You will want to make sure you do not have to interrupt the process.
 - Make sure you have the “Command Line Tools” installed.
 - Instructions are on the [First Things First Config Page](#)
 - Visit <http://brew.sh/>
 - Copy-paste the given instructions in the terminal *as a regular user (not root.)*.
- **VERY IMPORTANT:** READ WHAT THE OUTPUT IS!!!! It will tell you to do things, and you *have* to do them. Specifically
You should run '`brew doctor`' BEFORE you install anything.

OSX Package Management (**brew**)

- Installing and uninstalling:
 - Install a *formula*:
`brew install <fm1a1> <fm1a2> ... <fm1aN>`
 - Remove a formula:
`brew uninstall <fm1a1> <fm1a2> ... <fm1aN>`
 - Only one **fmla** required, but can specify many.
 - “Group” packages have no meaning in **brew**.
- Updating components:
 - Update **brew**, all *taps*, and installed formulae listings. This does not update the actual software you have installed with **brew**, just the definitions: **brew update**.
 - Update just installed formulae: **brew upgrade**.
 - Specify a **formula** name to only upgrade that formula.
- Searching for packages:
 - Same command: **brew search <formula>**

OSX: One of These Kids is Not Like the Others (Part I)

- Safe: confines itself (by default) in `/usr/local/Cellar`:
 - common feature of “non-system” package managers
 - No `sudo`, plays nicely with OSX (e.g. Applications, `python3`).
 - Non-linking by default. If a conflict is detected, it will tell you.
 - **Really important to read what `brew` tells you!!!**
- `brew` is modular. Additional repositories (“taps”) available:
 - This concept exists for all package managers
- Common taps people use:
 - `brew tap homebrew/science`
 - Various “scientific computing” tools, e.g. `opencv`.
 - `brew tap caskroom/cask`
 - Install `.app` applications! Safe: installs in the “Cellar”, symlinks to `~/Applications`, but *now these update with brew all on their own when you brew update!*
 - E.g. `brew cask install vlc`

OSX: One of These Kids is Not Like the Others (Part II)

- `brew` installs *formulas*.
 - A **ruby** script that provides rules for where to download something from / how to compile it. Similar concept to **portage**'s bash files
- Sometimes the packager creates a “**Bottle**”:
 - If a bottle for your version of OSX exists, you don't have to compile locally.
 - The bottle just gets *downloaded* and then “*poured*”.
- Otherwise, **brew** downloads the source and compiles locally.
- Though more time consuming, can be quite convenient!
 - `brew options opencv`
 - `brew install --with-cuda --c++11 opencv`
 - It really really really is magical. Just like USE flags in Gentoo!
 - `brew reinstall --with-missed-option formula`

OSX: One of These Kids is Not Like the Others (Part III)

- Reiteration: pay attention to **brew** and what it says. Seriously.
- Example: after installing **opencv**, it tells me:

==> Caveats

Python modules have been installed and Homebrew's site-packages is not in your Python sys.path, so you will not be able to import the modules this formula installed. If you plan to develop with these modules, please run:

```
mkdir -p /Users/sven/.local/lib/python2.7/site-packages  
echo 'import site; site.addsitedir(          \  
      "/usr/local/lib/python2.7/site-packages")' >> \  
/Users/sven/.local/lib/python2.7/site-packages/homebrew.pth
```

- **brew** gives copy-paste format, above is just so you can read.
- I want to use **opencv** in **Python**, so I do what **brew** tells me.

Language-specific package management

- Modern programming language environments have their own package managers
 - Haskell: **cabal**
 - Ocaml: **opam**
 - Python: **conda/pip/pip3**
 - Ruby: **bundler / gem**
 - Rust: **cargo**
- Works basically exactly like **brew**
 - separate, user-specific install directory
 - preferred to system packages but does not replace them
- Be careful when using these!
 - system packages are not preferred, but sometimes get used anyway
 - when languages rely on external packages, things get really hairy

Other Managers

Like What?

- There are so many package managers out there for different things, too many to list them all!
- Ruby: `gem`
- Anaconda Python: `conda`
- Python: `pip`
- Python: `easy_install` (but really, just use `pip`)
- Python3: `pip3`
- LaTeX: `tlmgr` (uses the CTAN database)
 - Must install TeX from source to get `tlmgr`
- Perl: `cpan`
- Sublime Text: `Package Control`
- Many many others...

Like How?

- Some notes and warnings about Python package management.
- Notes:
 - If you want X in Python 2 **and** 3:
 - `pip install X and pip3 install X`
 - OSX Specifically: advise only using **brew** or Anaconda Python.
The system Python can get really damaged if you modify it, you are better off leaving it alone.
 - So even if you want to use **python2** on Mac, I strongly encourage you to install it with **brew**.
- Warnings:
 - Don't mix **easy_install** and **pip**. Choose one, stick with it.
 - But the internet told me if I want **pip** on Mac, I should **easy_install pip**
 - NO! Because this **pip** will modify your **system** python, **USE BREW**.
 - Don't mix **pip** with **conda**. If you have Anaconda python, just stick to using **conda**.

Concepts in language-specific (per-user) package management

- Packages do not require root to install
- Packages installed to *per-user* directory
 - normally a “dotfile” directory in your home
 - better-behaved things in `~/.local/share`
- need to change your environment variables to use correctly
 - usually at least `$PATH` and `$LD_LIBRARY_PATH`
 - sometimes also `$JAVA_HOME`, `$PYTHON_PATH`, etc
- can control selection of package managers with edits to `$PATH`

Demo: the language-specific
package managers I have installed.

Choosing a Linux Distro, revisited

What is a linux distro?

- Custom combination of
 - kernel version,
 - default shell
 - package manager
 - graphical interface
- there are TOO MANY of these
 - open source: anyone can make one
- Most of the differences between distros are cosmetic
- Only very few “families” of distros with serious and important differences

What to consider when choosing a distro

- familiarity
 - how much of a learning curve will this be for me?
- popularity
 - how likely am I to find people on the internet who've seen my problems?
- community
 - Linux is **very** user-supported. How nice people on the internet are matters for your daily life.
 - Want to find a community where *you* feel supported and welcome
 - different distros are popular with different languages
- your use case
 - why do you want linux?
 - how often do you need or want bleeding-edge stuff?
 - what programs need to work for you?

Evaluating familiarity

- Package manager is most important
 - Ubuntu from *debian* family (uses .deb)
 - Fedora from *RedHat* family (uses .rpm)
 - distros will tell you where they're from
- desktop environment is second-most important
- Rest of it doesn't matter too much.

More about desktop environments

- Refers to “Graphical Shell” – the actual graphical part of the OS
 - Windows Explorer is the Windows Desktop Environment
 - Cocoa was the Mac Desktop environment (I think they changed that now?)
- Most important part of your daily computer experience
- Defines the look and feel of your OS
- Lots and **lots** of alternatives out there
- We’ll look at these at the end of lecture (and maybe next time too)

Evaluating popularity

- distrowatch.com
- Check their forums and website
- ask your friends
- look in the windows store (no really)

Evaluating community

- Read through random forum posts, especially of the “how do I install it” variety
- go on IRC (or whatever has replaced it) for the distro
 - really old chat service
 - basically only used for linux user support
- Check the wikis or other user-contribute items

Your use case

- Need stability and easy access to a terminal?
 - Maybe MacOS terminal / Windows Subsystem for Linux are good enough
- Need stability, terminal, and linux-specific hardware or graphics management?
 - Ubuntu and Debian
 - there are **lots** of distros based on one of these
 - they're all basically just as good as the next – differences are in customization, not essential
- Need serious security?
 - Linux in general is very secure
 - if you're very invested in security, find a security-focused distro

Your use case

- Need access to bleeding-edge software without upgrading your system?
 - docker might be good enough for you
 - if not, consider a rolling-release distro
 - can also consider a “bleeding” distro that emphasizes early package access
- Want to seriously get into the internals of your distro/customize packages?
 - Gentoo or Arch, or something based on those.

Back to desktop environments

Option 1: KDE



Figure 1: KDE

Option 2: GNOME



Figure 2: GNOME

Option 3: XFCE4

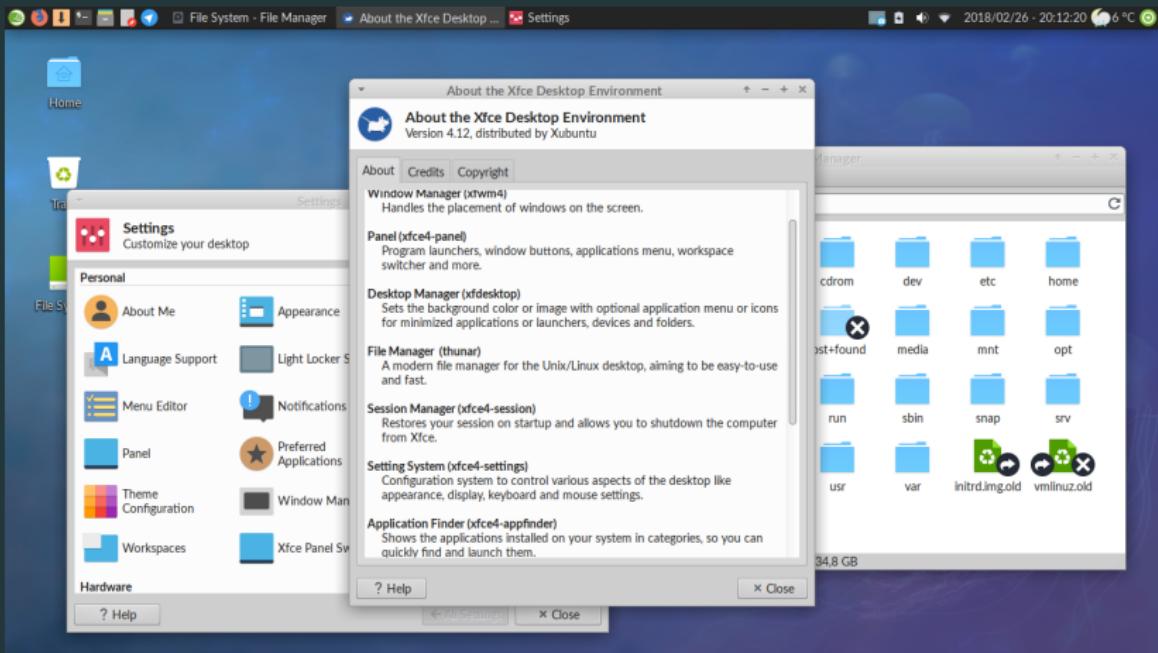


Figure 3: XFCE4

Tiling window managers

- Basic concept: windows don't "float" or "resize"; they always take all available area
- Can sub-divide screen into smaller regions (in half, in quarters, etc).
- Applications automatically resize to snap to your new "grid"
- don't move things with the mouse; change "splits" your screen with keyboard shortcuts
- some developers swear by these
- basically unused outside of serious developer circles.

XMonad

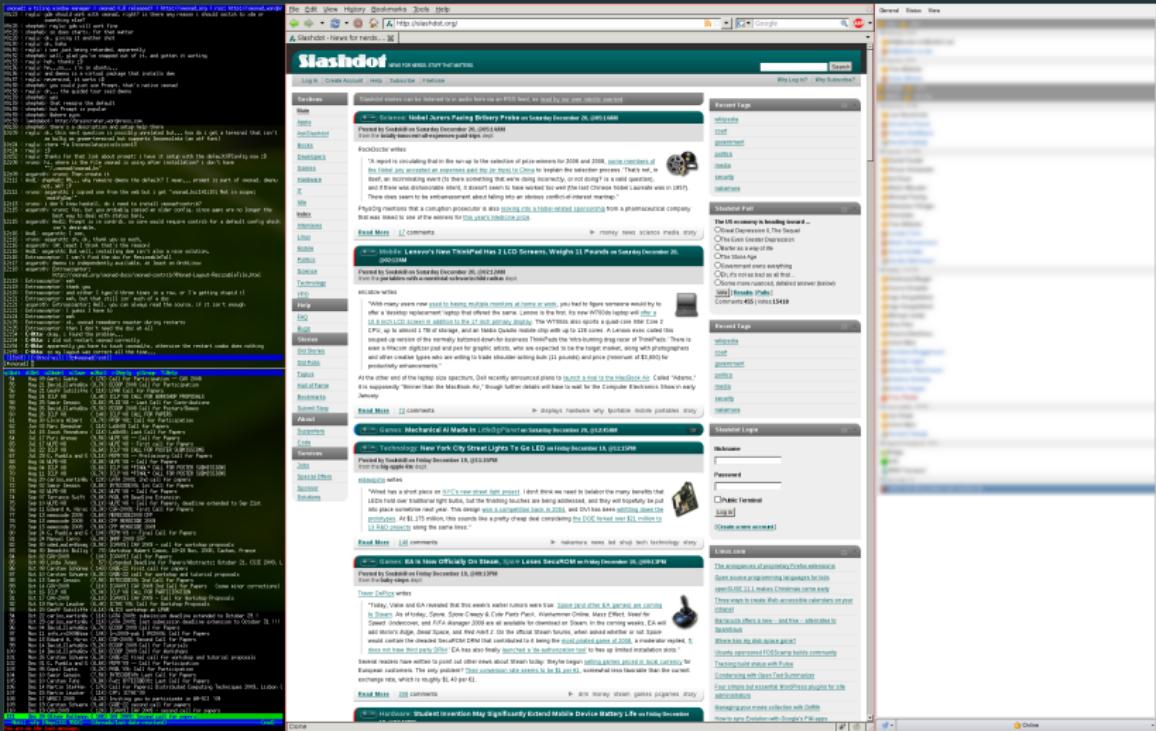


Figure 4: xmonad

Awesome

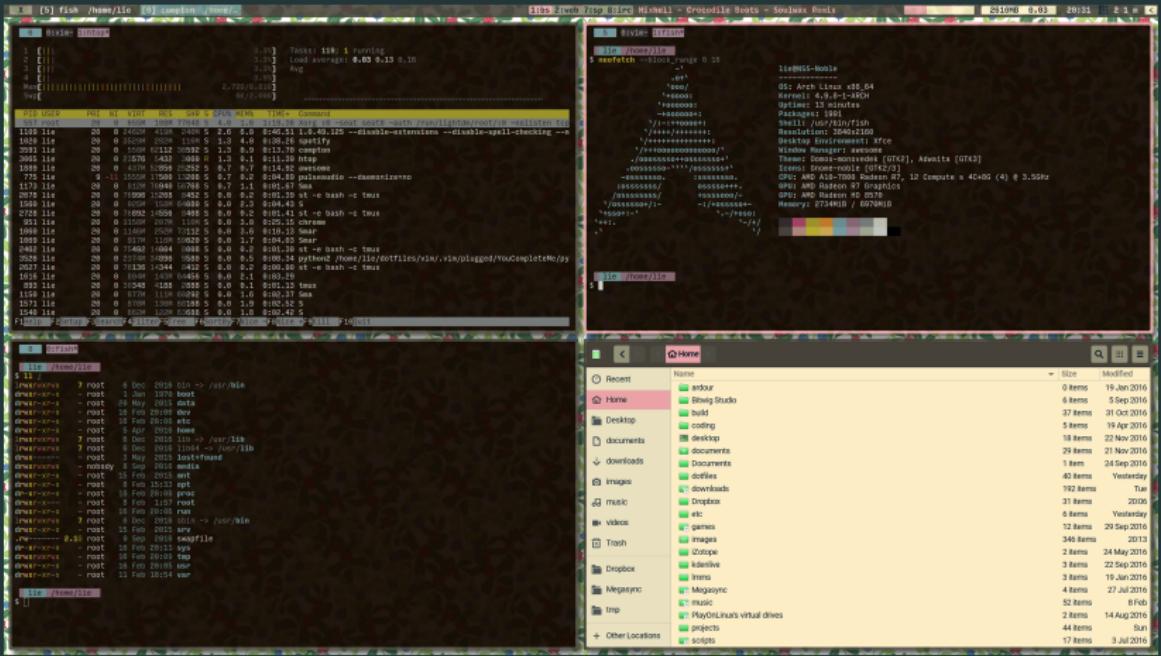


Figure 5: Awesome

References

- [1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years.
“Previous Cornell CS 2043 Course Slides”.