

13 – Networking and Package Management

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

Matthew Milano

February 18, 2019

Cornell University

Table of Contents

1. THE INTERNET
2. Package Management
3. System Specific Package Managers
4. Other Managers

As always: Everybody! ssh to wash.cs.cornell.edu

- Quiz time! Everybody! run **quiz-02-20-19**
- You can just explain a concept from last class, doesn't have to be a command this time.

THE INTERNET

How do computers communicate?

- Send data back and forth
- Data takes the form of *packets*



Figure 1: Paper airplane

Throwing paper airplanes blind???

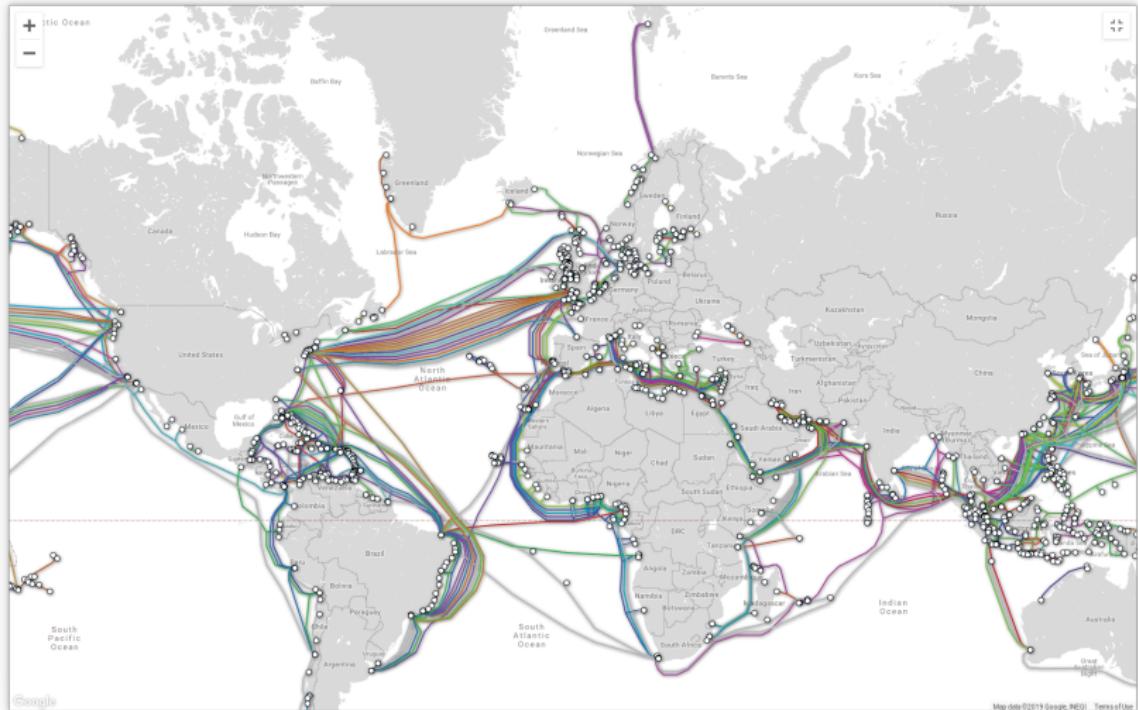
- Surprisingly accurate metaphor.
- You might miss! Need to acknowledge receipt
- You might throw at different speeds! Need sequence numbers
- these things (and more!) handled by the TCP protocol

Ok, but what's really going on?

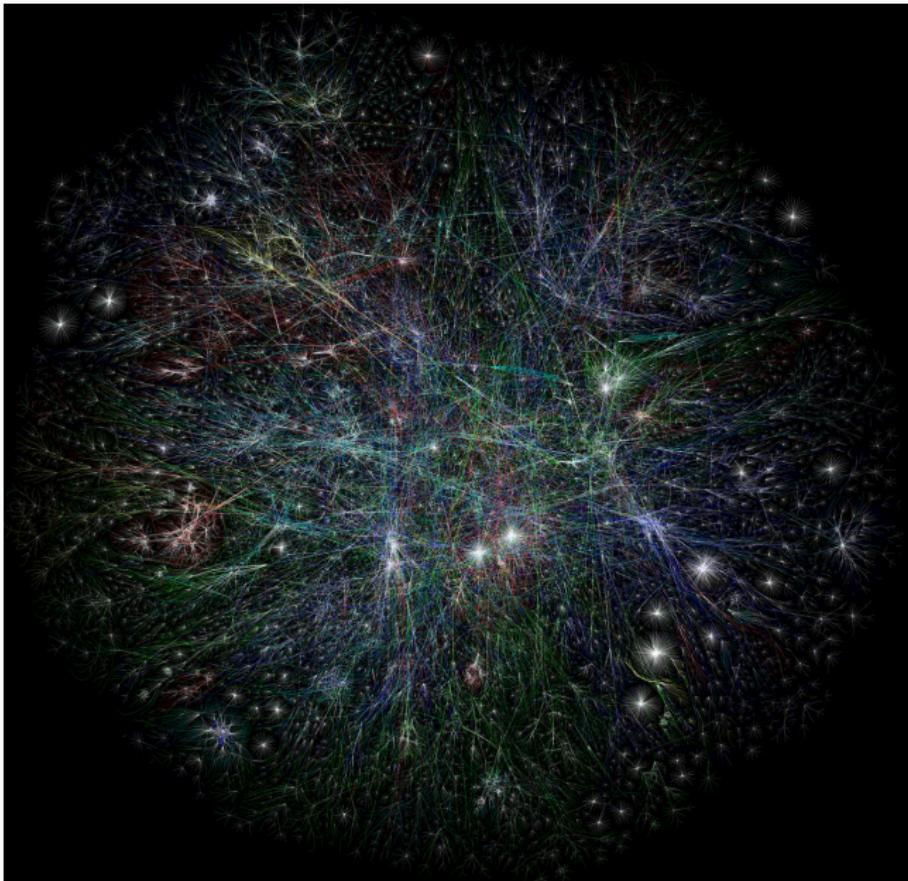
- Network cards convert local communication to network communication
- Limited memory and processing power
- Packet is small, “safe” size
 - will always fit in network hardware memory
 - is very fast to transfer; can’t easily “split” a packet
- once a packet arrives, networking hardware moves it as “data” to your application
- gives the illusion of a stream of packets
 - or an unending torrent of paper airplanes...

WHAT IS... THE INTERNET

- interconnected networks of computers



WHAT IS... THE INTERNET



Ok, smaller scale

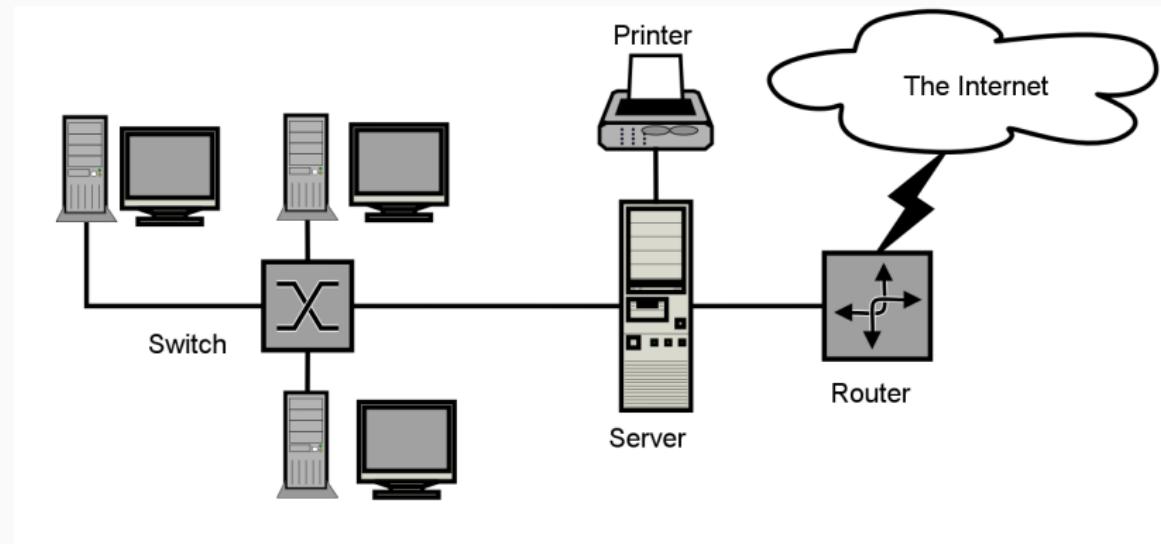


Figure 4: Small network

Finding your target

- You are *directly* connected to a switch
 - you send all your traffic there
 - the *switch* needs to know where to forward it
- Need an *address*: a destination the switch can understand
- We use IP addresses
 - just a sequence of numbers
- Can also *broadcast*
 - everyone gets it
 - usually blocked by firewalls outside of your local network

IPv4 Addresses:

- Usually displayed as **XXX.XXX.XXX.XXX**
 - example: **192.168.1.101**
 - example: **128.84.216.194**
- GLOBALLY routable: (most) IP addresses globally unique!
 - a few small exceptions: **192.168.***, **10.***
 - You can use these locally, but nobody can reach them from the wider internet.
 - think of them like private roads in the address system.
- How do we know they're globally unique?
 - Convention! The ICANN tells you what your number is, gets angry when you use the wrong one

Getting an IP address

- IP addresses change based on your network
- Each internet connection has its own “slice” of the IP space
 - computers attached to that network use its “slice”
- Can just give yourself an IP address
 - things break if you go outside the slice
- Can ask for help via DHCP
 - some server will give you an address
 - it probably knows which addresses are valid
 - it can tell when they’re used.
- ever see a **169.254.XX.YY** address? That’s your computer wildly guessing!

MAC Addresses:

- based on your actual exact hardware
- does not change based on network
- only used for *local* forwarding
 - within the same route
 - we'll talk about that later
- How switches actually connect to your device
- use the ARP protocol to map IP to MAC
 - Literally broadcast “who has 192.168.1.1? Tell 192.168.1.5”

Command break: ifconfig

configure your internet devices

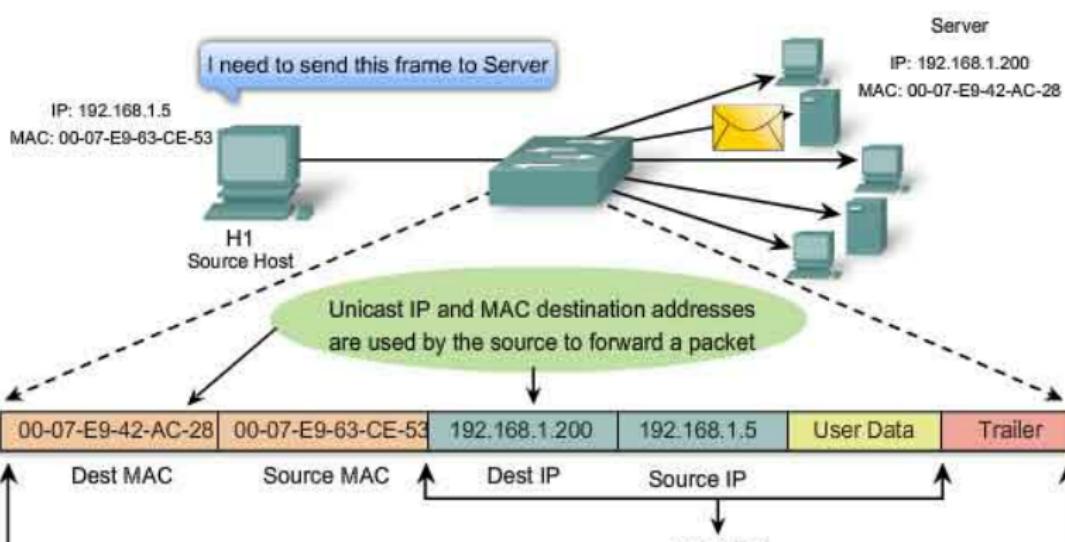
ifconfig [Options...]

```
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 40 bytes 2357 (2.3 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 40 bytes 2357 (2.3 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 10.132.6.227 netmask 255.255.128.0 broadcast 10.132.127.255
      inet6 fe80::4141:aa4f:3c83:5a88 prefixlen 64 scopeid 0x20<link>
        ether 00:23:15:f0:91:41 txqueuelen 1000 (Ethernet)
        RX packets 97595 bytes 34824177 (33.2 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 69065 bytes 34033683 (32.4 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Making packets

- Ethernet “frames” and IP “packets” are not exactly the same thing
- Like putting smaller envelopes in bigger ones
- “Wrap” Ethernet information around IP information



Routing packets

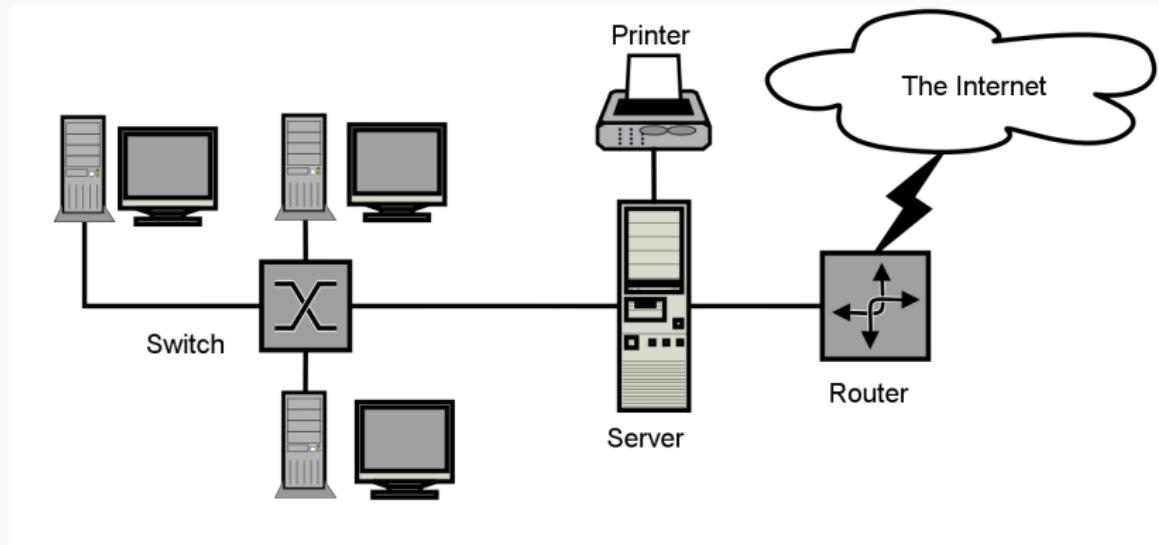


Figure 6: Small network

Routing packets

- Routers jump across network boundaries
- Limit of MAC protocol; MAC routes locally, IP routes globally
- Your *netmask* defines local IP addresses
 - Example: My IP is **192.168.1.100** and my netmask is **255.255.255.0**
 - Everything that starts with **192.168.1** is *local*; everything else needs *routing*
- Your *gateway* handles routing
 - Example: My IP is **192.168.1.100** and my gateway is **192.168.1.1**
 - Gateways have lots of IP addresses; they're connected to lots of networks!
 - Like this one Island in Canada...

Routing packets

- All IP addresses outside of local network use Router as initial destination
- Router re-packages packet for new network, acts as sender in that network
 - it will eventually hit a new network, and do that again

trace the **route** your packet takes

traceroute <dest_addr>

- lists each routing hop between you and destination
- displays *DNS name* or IP address

Traceroute example

Example:

```
$ traceroute google.com
traceroute to google.com (172.217.12.174), 30 hops max, 60 byte packets
1 rhodes1-ex-vl1245.net.cornell.edu (128.84.216.1) 1.372 ms 1.334 ms 1.355 ms
2 core1-mx-xe-11-0-4.net.cornell.edu (128.253.222.161) 0.720 ms 0.706 ms core2-mx-xe-11-0
3 wan2-mx-et-0-0-0.net.cornell.edu (128.253.222.58) 0.570 ms 0.568 ms 0.564 ms
4 199.109.109.25 (199.109.109.25) 3.089 ms 3.019 ms 3.021 ms
5 199.109.107.162 (199.109.107.162) 8.076 ms 8.042 ms 8.064 ms
6 72.14.202.166 (72.14.202.166) 8.007 ms 8.083 ms 8.054 ms
7 108.170.248.1 (108.170.248.1) 10.017 ms 10.052 ms 9.915 ms
8 108.170.226.201 (108.170.226.201) 9.055 ms 108.170.226.199 (108.170.226.199) 9.017 ms 1
9 lga25s62-in-f14.le100.net (172.217.12.174) 8.996 ms 8.992 ms 8.961 ms
```

DNS names

- A big dictionary in the sky
- translates english *domain* names (like **google.com**) into IP addresses
- Also managed by ICANN

domain information proper

```
dig [@DNS server] [domain name]
```

- gives you a lot of information behind the domain name
- includes IP address, owner, owner's e-mail address, and more

dig example

```
$ dig @1.1.1.1 google.com

; <>>> DiG 9.12.2-P2 <>>> @1.1.1.1 google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22816
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1452
;; QUESTION SECTION:
;google.com.           IN      A

;; ANSWER SECTION:
google.com.        215     IN      A      216.58.219.206

;; Query time: 10 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Feb 20 10:51:56 EST 2019
;; MSG SIZE  rcvd: 55
```

Package Management

Package Management Overview

- If I had to give *only one reason* why Unix systems are superior to Windows: Package Management.
- Can install almost anything with ease of from your terminal.
- Update to the latest version with one command.
 - No more download the latest installer nonsense!
- Various tools can be installed by installing a *package*.
 - A package contains the files and other instructions to setup a piece of software.
 - Many packages depend on each other.
 - High-level package managers download packages, figure out the dependencies for you, and deal with groups of packages.
 - Low-level managers unpack individual packages, run scripts, and get the software installed correctly.
- In general, these are “pre-compiled binaries”: no compilation necessary. It’s already packaged nice and neat just for you!

Package Managers in the Wild

- GNU/Linux:
 - Low-level: two general families of *packages* exist: **deb**, and **rpm**.
 - High-level package managers you are likely to encounter:
 - Debian/Ubuntu: **apt-get**.
 - Some claim that **aptitude** is superior, but I will only cover **apt-get**. They are roughly interchangeable.
 - SUSE/OpenSUSE: **zypper**.
 - Fedora: **dnf** (Fedora 22+).
 - **zypper** and **dnf** use **SAT**-based dependency solvers, which many argue is fundamentally superior. The dependency resolution phase is usually not the slowest part though...installing the packages is. See [3] for more info.
 - RHEL/CentOS: **yum** (until they adopt **dnf**).
- Mac OSX:
 - Others exist, but the only one you should ever use is **brew**.
 - Don't user others (e.g. **port**), they are outdated / EOSL.

Using Package Managers

- Though the syntax for each package manager is different, the concepts are all the same.
 - This lecture will focus on `apt-get`, `dnf`, and `brew`.
 - The `dnf` commands are almost entirely interchangeable with `yum`, by design.
 - Note that `brew` is a “special snowflake”, more on this later.
- What does your package manager give you? The ability to
 - `install` new packages you do not have.
 - `remove` packages you have installed.
 - `update` installed packages.
 - update the lists to search for files / updates from.
 - view `dependencies` of a given package.
 - a whole lot more!!!

A Note on **update**

- The **update** command has importantly different meanings in different package managers.
- Some **do**, and some do **not** default to system (read linux kernel) updates.
 - Ubuntu: default is *no*.
 - Fedora: default is *yes*.
 - RHEL: default is *no*.
- It depends on your operating system, and package manager.
 - Know your operating system, and look up what the default behavior is.
- If your program needs a specific version of the linux kernel, you need to be very careful!

A Note on Names and their Meanings

- You may see packages of the form:
 - <package>.i**[3456]86** (e.g. .i386 or .i686):
 - These are the **32-bit** packages.
 - <package>.x86_64: these are the **64-bit** packages.
 - <package>.noarch: these are independent of the architecture.
- Development tools can have as many as three packages:
 - The header files are usually called something like:
 - deb: usually <package>-dev
 - rpm: usually <package>-devel
 - The library you will need to link against:
 - If applicable, lib<package> or something similar.
 - The binaries (executables), often provided by just <package>.
 - Most relevant for C and C++, but also Python and others.
 - Use the **search** functionality of your package manager.

Example Development Tool Installation

- If I needed to compile and link against **Xrandr** (X.Org X11 libXrandr runtime library) pn Fedora, I would have to install
 - **libXrandr**: the library.
 - **libXrandr-devel**: the header files.
 - Not including **.x86_64** is OK / encouraged, your package manager knows which one to install.
 - Though in certain special cases you may need to get the **32-bit** library as well.
 - In this case, if I were compiling a program that links against **libXrandr**, but I want to release a pre-compiled **32bit** library, it must be installed in order for me to link against it.
- The **deb** versions should be similarly named, but just use the **search** functionality of find the right names.
- This concept has no meaning for **brew**, since it compiles everything.

System Specific Package Managers

Debian / Ubuntu Package Management (**apt-get**)

- Installing and uninstalling:

- Install a package:

```
apt-get install <pkg1> <pkg2> ... <pkgN>
```

- Remove a package:

```
apt-get remove <pkg1> <pkg2> ... <pkgN>
```

- Only one **pkg** required, but can specify many.

- “Group” packages are available, but still the same command.

- Updating components:

- Update lists of packages available: **apt-get update**.

- No arguments, it updates the whole list (even if you give args).

- Updating currently installed packages: **apt-get upgrade**.

- Specify a **package** name to only update / upgrade that package.

- Update core (incl. kernel): **apt-get dist-upgrade**.

- Searching for packages:

- Different command: **apt-cache search <pkg>**

RHEL / Fedora Package Managers (**yum** and **dnf**)

- Installing and uninstalling:
 - Install a package:
`dnf install <pkg1> <pkg2> ... <pkgN>`
 - Remove a package:
`dnf remove <pkg1> <pkg2> ... <pkgN>`
 - Only one **pkg** required, but can specify many.
 - “Group” packages are available, but different command:
 - `dnf groupinstall 'Package Group Name'`
- Updating components:
 - Update EVERYTHING: `dnf upgrade`.
 - `update` exists, but is essentially `upgrade`.
 - Specify a **package** name to only upgrade that package.
 - Updating repository lists: `dnf check-update`
- Searching for packages:
 - Same command: `dnf search <pkg>`
- **yum** and **dnf** (**Dandified Yum**) nearly interchangeable: [3].

dnf: Cautionary Tales

- **WARNING:** if you install package Y, which installs X as a dependency, and later **remove** Y
 - By default, X will be removed!
 - Refer to [2] for workarounds.
 - Generally, won't know you needed to **mark** until it is too late.
- Solution?
 - Basically, **pay attention to your package manager**.
 - It gets removed because nothing *explicitly* depends on it.
 - So one day you may realize “OH NO! I’m missing package X”...
 - ...so just **dnf install X**.
 - So while **mark** is available, personally I don’t use it.
 - Sad face, I know. Just the way of the world.

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`:
 - 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:
 - Essentially what a `.rpm` or `.deb` would give you in linux.
 - These are 3rd party repos, not officially sanctioned by `brew`.
- 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.
- 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. No need to understand the **opencv** build flags, because the authors of the **brew** formula are kind and wonderful people.
 - `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.

Less Common Package Management Operations

- Sometimes when dependencies are installed behind the scenes, and you no longer need them, you will want to get rid of them.
 - `apt-get autoremove`
 - `dnf autoremove`
 - `brew doctor`
- View the list of repositories being checked:
 - `apt-cache policy` (well, sort of...`apt` doesn't have it)
 - `dnf repolist [enabled|disabled|all]`
 - Some repositories for `dnf` are *disabled* by default (with good reason). Usually you want to just
`dnf --enablerepo=<name> install <thing>`
e.g. if you have `rawhide` (development branch for fedora).
 - `brew tap`

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`.

References

- [1] Stephen McDowell, Bruno Abrahaao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years.
“Previous Cornell CS 2043 Course Slides”.
- [2] Reddit.com. *DNF Remove Package, keep dependencies??* 2016. URL: https://www.reddit.com/r/Fedora/comments/3pqr9/dnf_remove_package_keep_dependencies/.
- [3] Jack Wallen. *What You Need to Know About Fedora’s Switch From Yum to DNF*. 2015. URL: <https://www.linux.com/learn/tutorials/838176-what-you-need-to-know-about-fedoras-switch-from-yum-to-dnf>.