14 - Package Management, Git Merging

CS 2043: Unix Tools and Scripting, Spring 2016 [1]

Stephen McDowell

February 29th, 2016

Cornell University

Table of contents

Some Logistics

- Happy leap day!

Some Logistics

- Happy leap day!
- Lots of great questions on Piazza, keep it up!

Some Logistics

- Happy leap day!
- Lots of great questions on Piazza, keep it up!
- Today is going to be a lot of fun (at least for me).

Package Management

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think of from your terminal.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think of from your terminal.
- Update to the latest version with one command.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think of from your terminal.
- Update to the latest version with one command.
    - No more download the latest installer nonsense!

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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*.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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.

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think 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:

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
    - High-level package managers you are likely to encounter:

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
    - High-level package managers you are likely to encounter:
        - Debian/Ubuntu: `apt-get`.

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
    - 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.

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
    - 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`.

Package Managers in the Wild

- GNU/Linux:
  - Two general families of *packages* exist: **`deb`**, and **`rpm`** (low-level).
  - 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+).

Package Managers in the Wild

- GNU/Linux:
  - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
  - 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. Though the dependency resolution phase is usually not the slowest part...installing the packages is. See [2] for more info.

Package Managers in the Wild

- GNU/Linux:
  - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
  - 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. Though the dependency resolution phase is usually not the slowest part…installing the packages is. See [2] for more info.
    - RHEL/CentOS: `yum` (until they adopt `dnf`).

Package Managers in the Wild

- GNU/Linux:
  - Two general families of *packages* exist: **`deb`**, and **`rpm`** (low-level).
  - 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. Though the dependency resolution phase is usually not the slowest part...installing the packages is. See [2] for more info.
    - RHEL/CentOS: **`yum`** (until they adopt **`dnf`**).
- Mac OSX:

Package Managers in the Wild

- GNU/Linux:
  - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
  - 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. Though the dependency resolution phase is usually not the slowest part...installing the packages is. See [2] for more info.
    - RHEL/CentOS: `yum` (until they adopt `dnf`).
- Mac OSX:
  - Others exist, but the only one you should ever use is `brew`.

Package Managers in the Wild

- GNU/Linux:
    - Two general families of *packages* exist: `deb`, and `rpm` (low-level).
    - 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. Though the dependency resolution phase is usually not the slowest part...installing the packages is. See [2] 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 the commands are different depending on your OS, the concepts are all the same.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, the concepts are all the same.
    - This lecture will focus on `apt-get`, `dnf`, and `brew`.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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!!!

Using Package Managers

- Though the syntax for the commands are different depending on your OS, 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!!!

* See next slide for a potential `update` pitfalls.

A Note on **`update`**

- The **`update`** command has importantly different meanings in different package managers.

A Note on `update`

- The `update` command has importantly different meanings in different package managers.
- Some (`deb`) do not default to system (read linux kernel) updates.

A Note on `update`

- The `update` command has importantly different meanings in different package managers.
- Some (`deb`) do not default to system (read linux kernel) updates.
- Some (`rpm`) DO default to system updates!

A Note on **`update`**

- The **`update`** command has importantly different meanings in different package managers.
- Some (**`deb`**) do not default to system (read linux kernel) updates.
- Some (**`rpm`**) DO default to system updates!
- The difference lies somewhat in philosophy, and somewhat in the differences between the two.

A Note on **`update`**

- The **`update`** command has importantly different meanings in different package managers.
- Some (**`deb`**) do not default to system (read linux kernel) updates.
- Some (**`rpm`**) DO default to system updates!
- The difference lies somewhat in philosophy, and somewhat in the differences between the two.
- 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:

A Note on Names and their Meanings

- You may see packages of the form:
    - `<package>.i[3456]86` (e.g. `.i686`): these are the `32-bit` packages.

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.i686`): these are the `32-bit` packages.
  - `<package>.x86_64`: these are the `64-bit` packages.

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.i686`): these are the `32-bit` packages.
  - `<package>.x86_64`: these are the `64-bit` packages.
  - `<package>.noarch`: these are independent of the architecture.

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
  - The header files are usually called something like:

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
  - The header files are usually called something like:
    - deb: usually `<package>-dev`

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
  - The header files are usually called something like:
    - `deb`: usually `<package>-dev`
    - `rpm`: usually `<package>-devel`

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
  - The header files are usually called something like:
    - **deb**: usually `<package>-dev`
    - **rpm**: usually `<package>-devel`
  - The library you will need to link against:

A Note on Names and their Meanings

- You may see packages of the form:
  - `<package>.i[3456]86` (e.g. `.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 installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
  - 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.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on Fedora, I would have to install

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on Fedora, I would have to install
    - `libXrandr`: the library.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on Fedora, I would have to install
  - `libXrandr`: the library.
  - `libXrandr-devel`: the header files.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on 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.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on 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.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on 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.
- The `deb` versions should be similarly named, but just use the `search` functionality of find the right names.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on 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.
- 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

- Installing and uninstalling:

Debian / Ubuntu Package Management

- Installing and uninstalling:
  - Install a package:

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

Debian / Ubuntu Package Management

- Installing and uninstalling:
  - Install a package:
    `apt-get install <pkg1> <pkg2> ... <pkgN>`
  - Remove a package:
    `apt-get remove <pkg1> <pkg2> ... <pkgN>`

Debian / Ubuntu Package Management

- 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.

Debian / Ubuntu Package Management

- 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.

Debian / Ubuntu Package Management

- 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:

Debian / Ubuntu Package Management

- 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:
  - Updating currently installed packages: `apt-get update`.

Debian / Ubuntu Package Management

- 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:
    - Updating currently installed packages: `apt-get update`.
    - Update lists of packages available: `apt-get upgrade`.

Debian / Ubuntu Package Management

- 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:
  - Updating currently installed packages: `apt-get update`.
  - Update lists of packages available: `apt-get upgrade`.
    - If you instead specify a `package` name, it will only update / upgrade that package.

Debian / Ubuntu Package Management

- 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:
    - Updating currently installed packages: `apt-get update`.
    - Update lists of packages available: `apt-get upgrade`.
        - If you instead specify a `package` name, it will only update / upgrade that package.
    - Update core (incl. kernel): `apt-get dist-upgrade`.

Debian / Ubuntu Package Management

- 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:
    - Updating currently installed packages: `apt-get update`.
    - Update lists of packages available: `apt-get upgrade`.
        - If you instead specify a `package` name, it will only update / upgrade that package.
    - Update core (incl. kernel): `apt-get dist-upgrade`.
- Searching for packages:

Debian / Ubuntu Package Management

- 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:
  - Updating currently installed packages: `apt-get update`.
  - Update lists of packages available: `apt-get upgrade`.
    - If you instead specify a `package` name, it will only update / upgrade that package.
  - Update core (incl. kernel): `apt-get dist-upgrade`.
- Searching for packages:
  - Different command: `apt-cache search <pkg>`

RHEL / Fedora (**`yum`** and **`dnf`**)

- Installing and uninstalling:

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

- Installing and uninstalling:
  - Install a package:

    ```
    dnf install <pkg1> <pkg2> ... <pkgN>
    ```

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

- Installing and uninstalling:
  - Install a package:
    `dnf install <pkg1> <pkg2> ... <pkgN>`
  - Remove a package:
    `dnf remove <pkg1> <pkg2> ... <pkgN>`

RHEL / Fedora (**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.

RHEL / Fedora (**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'`

RHEL / Fedora (**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:

RHEL / Fedora (**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`.

RHEL / Fedora (**`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`.

RHEL / Fedora (**`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.

RHEL / Fedora (**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`

RHEL / Fedora (**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:

RHEL / Fedora (**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>`

OSX Package Management: Install `brew` on your own

1. Install Xcode (if you have not already).
   `cmd+space` then type App Store. Search for Xcode and install.

   This method requires a valid apple login / password. Make one if you don't have it. Hint: you don't actually have to give them a credit card; I never do because I refuse to give them more money intentionally or accidentally. But updates will be ubiquitous and this method is preferred to alternatives.

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

1. Install Xcode (if you have not already).
   `cmd+space` then type App Store. Search for Xcode and install.

   This method requires a valid apple login / password. Make one if you don't have it. Hint: you don't actually have to give them a credit card; I never do because I refuse to give them more money intentionally or accidentally. But updates will be ubiquitous and this method is preferred to alternatives.
2. Install CMD Line Tools from terminal: `xcode-select --install`

OSX Package Management: Install `brew` on your own

1. Install Xcode (if you have not already).
   `cmd+space` then type App Store. Search for Xcode and install.

   This method requires a valid apple login / password. Make one if you don't have it. Hint: you don't actually have to give them a credit card; I never do because I refuse to give them more money intentionally or accidentally. But updates will be ubiquitous and this method is preferred to alternatives.
2. Install CMD Line Tools from terminal: `xcode-select --install`
3. Install XQuartz: http://www.xquartz.org/
   `brew` and many items you would install need linux-style `X11`.

OSX Package Management: Install `brew` on your own

1. Install Xcode (if you have not already).
   `cmd+space` then type App Store. Search for Xcode and install.
   This method requires a valid apple login / password. Make one if you don't have it. Hint: you don't actually have to give them a credit card; I never do because I refuse to give them more money intentionally or accidentally. But updates will be ubiquitous and this method is preferred to alternatives.
2. Install CMD Line Tools from terminal: `xcode-select --install`
3. Install XQuartz: http://www.xquartz.org/
   `brew` and many items you would install need linux-style `X11`.
4. Install brew: http://brew.sh/
   Follow directions at top: Install Homebrew (paste the text into your terminal and hit enter.)

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

1. Install Xcode (if you have not already).
   `cmd+space` then type App Store. Search for Xcode and install.
   This method requires a valid apple login / password. Make one if you don't have it. Hint: you don't actually have to give them a credit card; I never do because I refuse to give them more money intentionally or accidentally. But updates will be ubiquitous and this method is preferred to alternatives.
2. Install CMD Line Tools from terminal: `xcode-select --install`
3. Install XQuartz: http://www.xquartz.org/
   `brew` and many items you would install need linux-style `X11`.
4. Install brew: http://brew.sh/
   Follow directions at top: Install Homebrew (paste the text into your terminal and hit enter.)
5. 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: Using **brew**

- Installing and uninstalling:

OSX: Using **brew**

- Installing and uninstalling:
    - Install a *formula*:

      `brew install <fmla1> <fmla2> ... <fmla2>`

OSX: Using **brew**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:

    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:

    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - Only one **`fmla`** required, but can specify many.

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - Only one **`fmla`** required, but can specify many.
  - "Group" packages have no meaning in **`brew`**.

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - Only one **`fmla`** required, but can specify many.
  - "Group" packages have no meaning in `brew`.
- Updating components:

OSX: Using `brew`

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - 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 (more on next slide): `brew update`.

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - 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 (more on next slide): `brew update`.
  - Update just installed formulae: `brew upgrade`.

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - 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 (more on next slide): `brew update`.
  - Update just installed formulae: `brew upgrade`.
    - Specify a **`formula`** name to only upgrade that formula.

OSX: Using `brew`

- Installing and uninstalling:
    - Install a *formula*:
      `brew install <fmla1> <fmla2> ... <fmla2>`
    - Remove a formula:
      `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
    - 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 (more on next slide): `brew update`.
    - Update just installed formulae: `brew upgrade`.
        - Specify a `formula` name to only upgrade that formula.
- Searching for packages:

OSX: Using **`brew`**

- Installing and uninstalling:
  - Install a *formula*:
    `brew install <fmla1> <fmla2> ... <fmla2>`
  - Remove a formula:
    `brew uninstall <fmla1> <fmla2> ... <fmlaN>`
  - 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 (more on next slide): `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: **`brew`** is a special snowflake (Part I)

- Safe: confines itself (by default) in `/usr/local/Cellar`:

OSX: `brew` is a special snowflake (Part I)

- Safe: confines itself (by default) in `/usr/local/Cellar`:
  - No `sudo`, plays nicely with OSX (e.g. Applications, `python3`).

OSX: `brew` is a special snowflake (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.

OSX: `brew` is a special snowflake (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!!!

OSX: `brew` is a special snowflake (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. There is a main list of repositories, but there are also additional *taps*:

OSX: `brew` is a special snowflake (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. There is a main list of repositories, but there are also additional *taps*:
  - A tap is effectively another repository list, like what a `.rpm` or `.deb` would give you in linux.

OSX: `brew` is a special snowflake (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. There is a main list of repositories, but there are also additional *taps*:
  - A tap is effectively another repository list, like what a `.rpm` or `.deb` would give you in linux.
  - Common taps people use:

OSX: `brew` is a special snowflake (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. There is a main list of repositories, but there are also additional *taps*:
    - A tap is effectively another repository list, like what a `.rpm` or `.deb` would give you in linux.
    - Common taps people use:
        - `brew tap homebrew/science`
          Various "scientific computing" tools, e.g. `opencv`.

OSX: `brew` is a special snowflake (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. There is a main list of repositories, but there are also additional *taps*:
  - A tap is effectively another repository list, like what a `.rpm` or `.deb` would give you in linux.
  - 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*!
      E.g. `brew cask install vlc`

OSX: `brew` is a special snowflake (Part II)

- `brew` installs *formulas*.

OSX: `brew` is a special snowflake (Part II)

- `brew` installs *formulas*.
    - A formula is *not* a pre-compiled binary, it is a `ruby` script that provides rules for where to download something from / how to compile it.

OSX: **`brew`** is a special snowflake (Part II)

- `brew` installs *formulas*.
    - A formula is *not* a pre-compiled binary, it is a **`ruby`** script that provides rules for where to download something from / how to compile it.
    - You download a **`bottle`** that gets *poured*: download source and compile (ish).

OSX: `brew` is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a `ruby` script that provides rules for where to download something from / how to compile it.
  - You download a `bottle` that gets *poured*: download source and compile (ish).
  - Though more time consuming, can be quite convenient!

OSX: **`brew`** is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a **`ruby`** script that provides rules for where to download something from / how to compile it.
  - You download a **`bottle`** that gets *poured*: download source and compile (ish).
  - Though more time consuming, can be quite convenient!
    - `brew options opencv`

OSX: **`brew`** is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a **`ruby`** script that provides rules for where to download something from / how to compile it.
  - You download a **`bottle`** that gets *poured*: download source and compile (ish).
  - Though more time consuming, can be quite convenient!
    - `brew options opencv`
    - `brew install --with-cuda --c++11 opencv`

OSX: **`brew`** is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a **`ruby`** script that provides rules for where to download something from / how to compile it.
  - You download a **`bottle`** that gets *poured*: download source and compile (ish).
  - 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.

OSX: `brew` is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a `ruby` script that provides rules for where to download something from / how to compile it.
  - You download a `bottle` that gets *poured*: download source and compile (ish).
  - 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: `brew` is a special snowflake (Part II)

- `brew` installs *formulas*.
  - A formula is *not* a pre-compiled binary, it is a `ruby` script that provides rules for where to download something from / how to compile it.
  - You download a `bottle` that gets *poured*: download source and compile (ish).
  - 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`
- Of course, there is a whole lot more that `brew` does, just like the other package managers.

OSX: **brew** is a special snowflake (Part III)

- You REALLY need to 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
# (continued onto newline so you can read, it gives you copy-paste format!)
```

- Obviously I want to use **opencv** with **Python**, so I am going to follow what **brew** tells me to do.
- If it may cause problems, it will tell you what the problems might be.

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
    - `apt-get autoremove`

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
    - `apt-get autoremove`
    - `dnf autoremove`

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `apt-get autoremove`
  - `dnf autoremove`
  - `brew doctor`

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `apt-get autoremove`
  - `dnf autoremove`
  - `brew doctor`
- View the list of repositories being checked:

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `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)

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `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]`

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `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).

Less Common Package Management Operations

- Many people don't realize that if you install package X and it installed a bunch of dependencies, they don't remove the dependencies when you remove X.
  - `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!

Like What?

- There are so many package managers out there for different things, too many to list them all!
- Ruby: `gem`

Like What?

- There are so many package managers out there for different things, too many to list them all!
- Ruby: `gem`
- Anaconda Python: `conda`

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`

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`)

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`

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)

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)
- Perl: `cpan`

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)
- Perl: `cpan`
- Sublime Text has its own package manager: Package Control.

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)
- Perl: `cpan`
- Sublime Text has its own package manager: Package Control.
- Many many others...

Like How?

- Some notes and warnings about Python package management.

Like How?

- Some notes and warnings about Python package management.
- Notes:

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
  - 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.

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
  - 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.
  - This is why I am having you install `python3` on the next page.

Like How?

- Some notes and warnings about Python package management.
- Notes:
    - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
    - 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.
    - This is why I am having you install `python3` on the next page.
- Warnings:

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
  - 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.
  - This is why I am having you install `python3` on the next page.
- Warnings:
  - Don't mix `easy_install` and `pip`. Choose one, stick with it.

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
  - 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.
  - This is why I am having you install `python3` on the next page.
- Warnings:
  - Don't mix `easy_install` and `pip`. Choose one, stick with it.
  - Don't mix `pip` with `conda`. If you have Anaconda python, just stick to using `conda`.

Like How?

- Some notes and warnings about Python package management.
- Notes:
  - If you install something with `pip`, and try to use it with Python3, it will not work. You have to also install it with `pip3`.
  - 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.
  - This is why I am having you install `python3` on the next page.
- Warnings:
  - Don't mix `easy_install` and `pip`. Choose one, stick with it.
  - Don't mix `pip` with `conda`. If you have Anaconda python, just stick to using `conda`.
  - If you installed Anaconda Python 2, you can still install Python 3 and use `pip3`, but things may get a little weird with updating `pip3`. Don't update `pip3`, or install Anaconda Python3 as well.

Like Python3!!!

- Let's install Python 3 (system specific):

Like Python3!!!

- Let's install Python 3 (system specific):

```
# Ubuntu
apt-get install build-essential python3-dev python3-pip

# Fedora 23 [ALREADY HAVE IT! Need dev tools though]
dnf groupinstall 'Development Tools'
dnf install python3-devel

# OSX
brew install python3
```

Like Python3!!!

- Let's install Python 3 (system specific):

```
# Ubuntu
apt-get install build-essential python3-dev python3-pip

# Fedora 23 [ALREADY HAVE IT! Need dev tools though]
dnf groupinstall 'Development Tools'
dnf install python3-devel

# OSX
brew install python3
```

- Now that we have `python3`, lets install a cool debugger:

Like Python3!!!

- Let's install Python 3 (system specific):

```
# Ubuntu
apt-get install build-essential python3-dev python3-pip

# Fedora 23 [ALREADY HAVE IT! Need dev tools though]
dnf groupinstall 'Development Tools'
dnf install python3-devel

# OSX
brew install python3
```

- Now that we have `python3`, lets install a cool debugger:

```
pip3 install simplegeneric pickleshare pexpect ipdb
```

Like Python3!!!

- Let's install Python 3 (system specific):

```
# Ubuntu
apt-get install build-essential python3-dev python3-pip

# Fedora 23 [ALREADY HAVE IT! Need dev tools though]
dnf groupinstall 'Development Tools'
dnf install python3-devel

# OSX
brew install python3
```

- Now that we have `python3`, lets install a cool debugger:

```
pip3 install simplegeneric pickleshare pexpect ipdb
```

- You can now debug the lecture 14 demo script:

https://github.com/cs2043-sp16/lecture-demos/tree/master/lec14

References I

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.
Previous cornell cs 2043 course slides.

[2] Linux.com.
What you need to know about fedora's switch from yum to dnf.
https://www.linux.com/learn/tutorials/838176-what-you-need-to-know-about-fedoras-switc