

IT-566

Computer Scripting

Techniques

Week 2

Talking Points

Development Environment
Setup and Configuration
(Loose Ends)

Project Organization
& Layout

Bash Build Script

Git Development Workflow

Development Environment

Setup and Configuration (Loose Ends)

Package Managers

tree Command

Configure ssh for GitHub

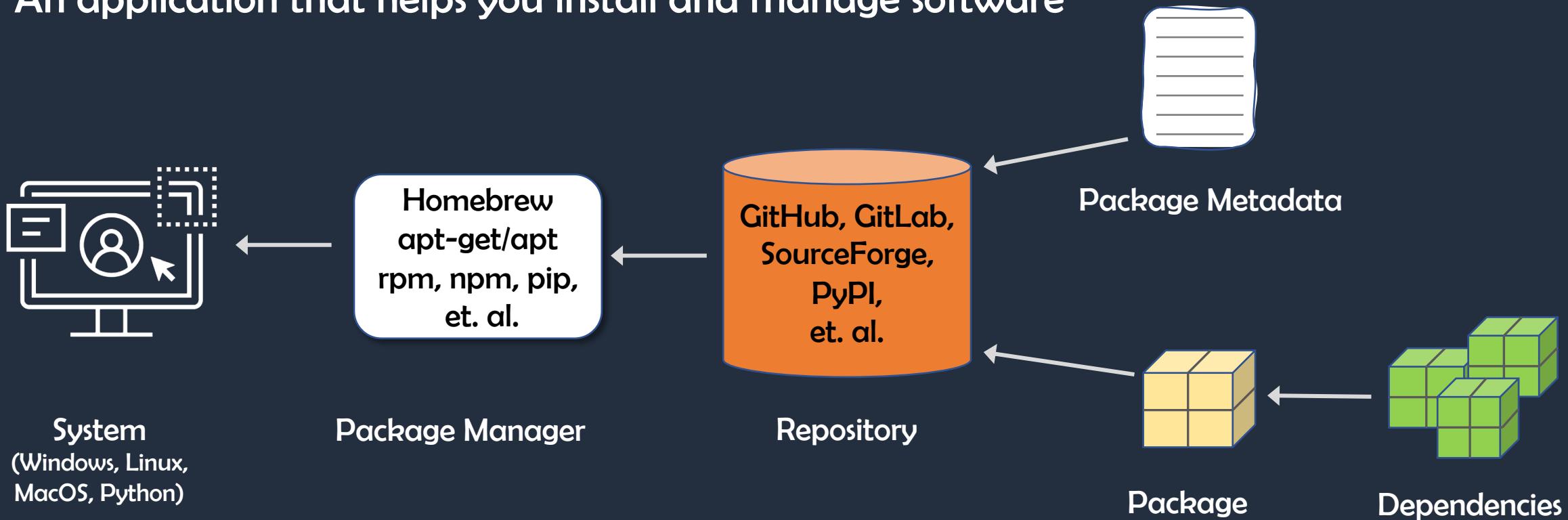
pipenv
Setup, Usage & Why You
Need It

.bash_profile
Helpful Settings

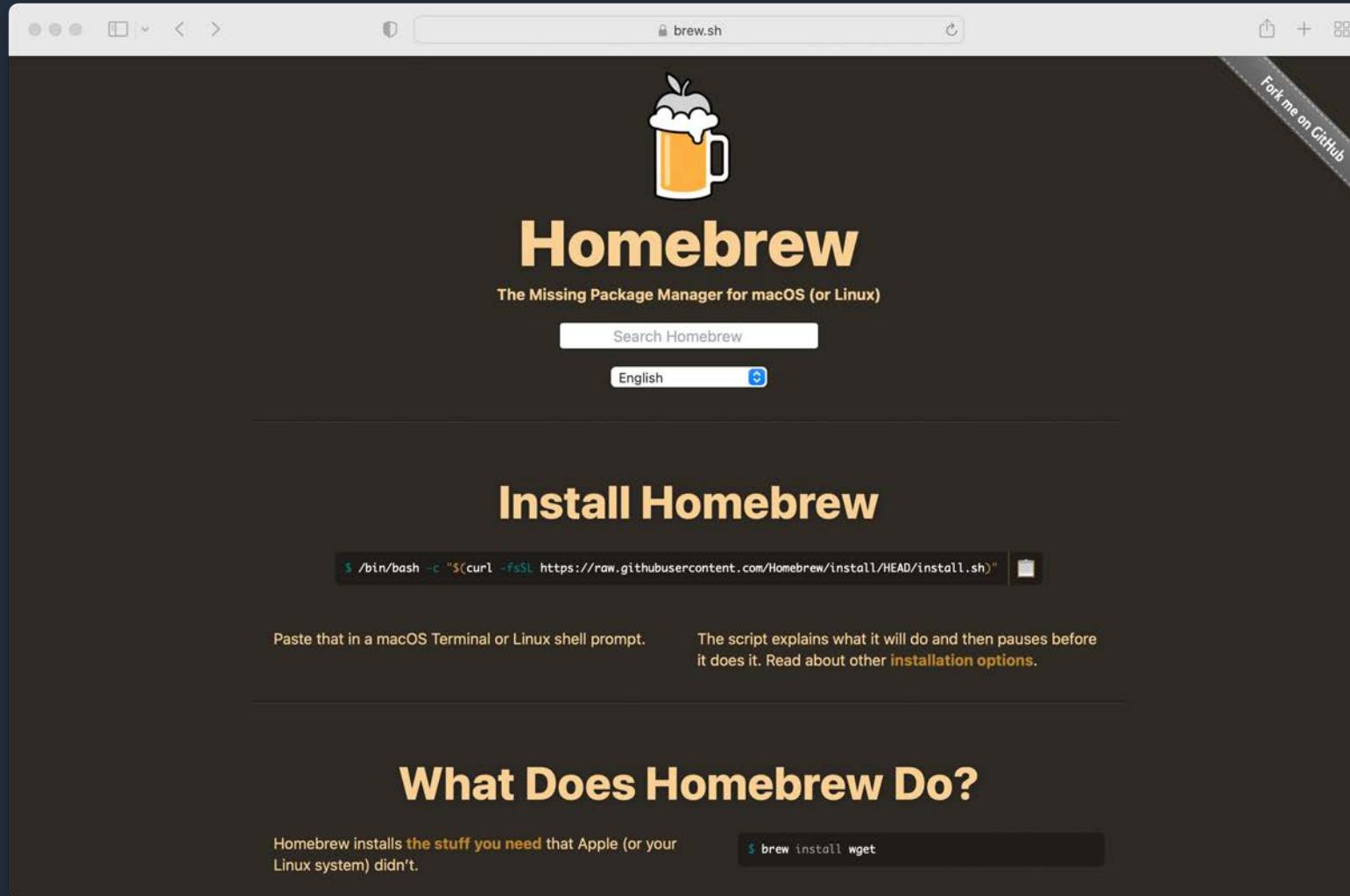
Package Managers

What Is A Package Manager?

- An application that helps you install and manage software



Homebrew MacOS (And Linux) Package Manager



- Package Manager for MacOS
 - x86 & arm64
- Simplifies Software Installation
 - Especially Unix utils
 - Developer tools
- Linux, too
 - x86 (no arm64 binaries)

Linux Package Managers



Depends on Distribution

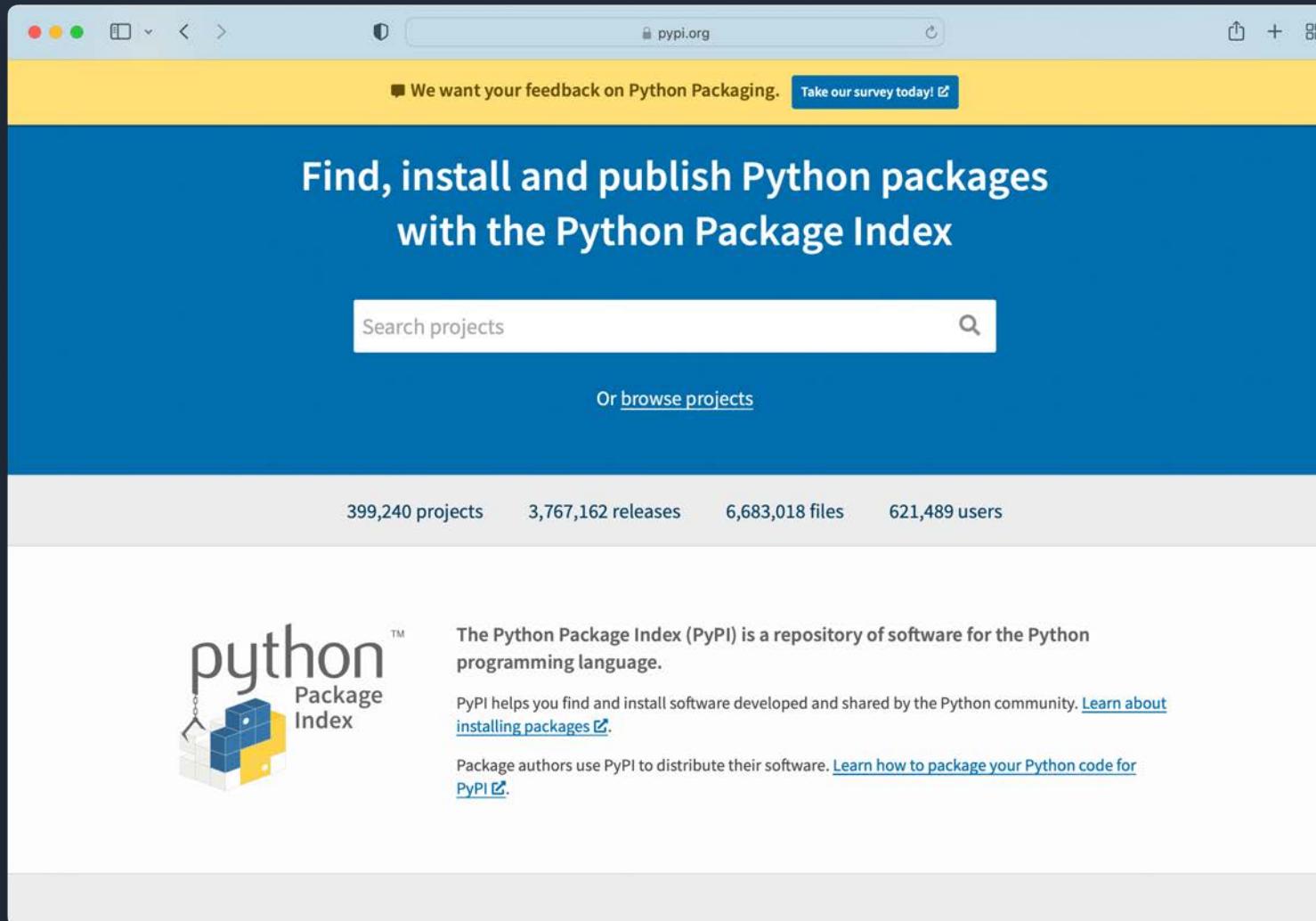


apt-get / apt



rpm

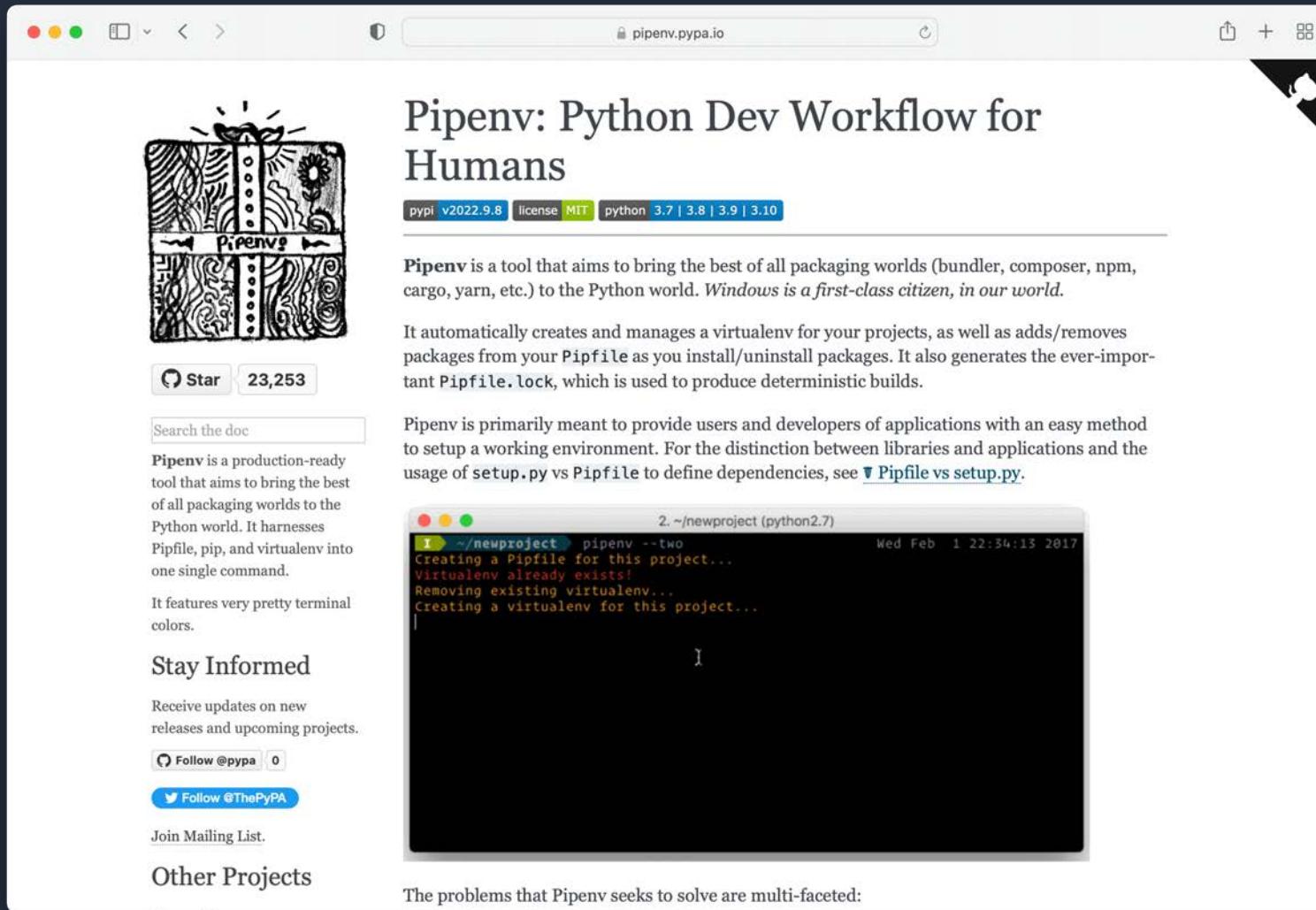
Python Package Manager — pip



- **pip or pip3**
 - **pip for Python 2**
 - **pip3 for Python 3**
 - **Generally...**
- **PyPI**
 - **Python Package Index**

pipenv Setup, Usage, & Why You Need It

Pipenv — Why You Need It



The screenshot shows the official Pipenv website at pipenv.pypa.io. The page features a decorative illustration of a gift box labeled "Pipenv" on the front. The main heading is "Pipenv: Python Dev Workflow for Humans". Below it, there's a badge for "pypi v2022.9.8", "license MIT", and "python 3.7 | 3.8 | 3.9 | 3.10". A brief description states: "Pipenv is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world. *Windows is a first-class citizen, in our world.*" It explains that Pipenv creates and manages virtual environments and generates a `Pipfile.lock`. Another section discusses the distinction between `setup.py` and `Pipfile`. A terminal window at the bottom shows the command `pipenv --two` being run, which creates a new virtual environment for Python 2.7.

Pipenv: Python Dev Workflow for Humans

pypi v2022.9.8 license MIT python 3.7 | 3.8 | 3.9 | 3.10

Pipenv is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world. *Windows is a first-class citizen, in our world.*

It automatically creates and manages a virtualenv for your projects, as well as adds/removes packages from your `Pipfile` as you install/uninstall packages. It also generates the ever-important `Pipfile.lock`, which is used to produce deterministic builds.

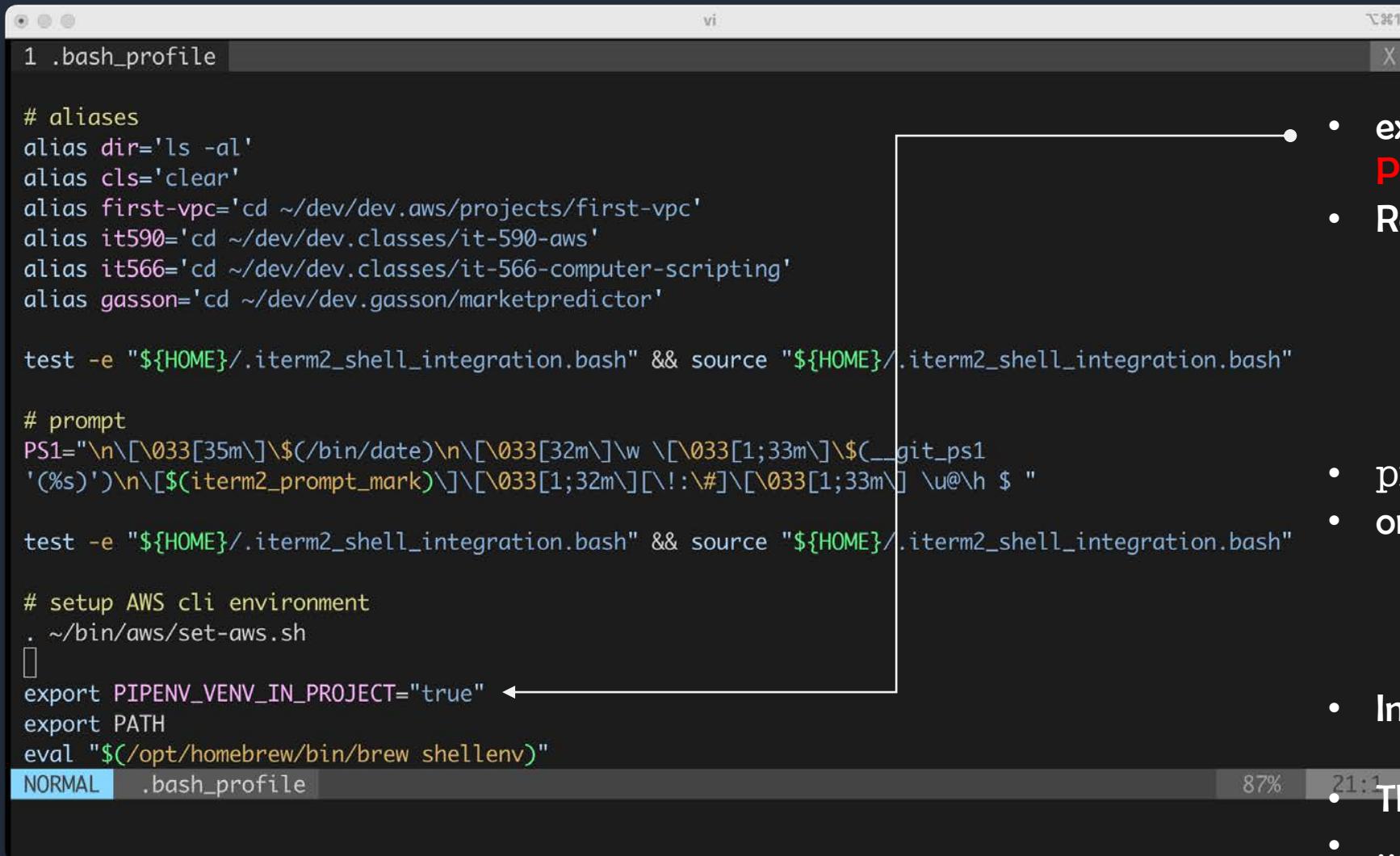
Pipenv is primarily meant to provide users and developers of applications with an easy method to setup a working environment. For the distinction between libraries and applications and the usage of `setup.py` vs `Pipfile` to define dependencies, see [Pipfile vs setup.py](#).

```
2. ~/newproject (python2.7) Wed Feb 1 22:34:13 2017
~/newproject pipenv --two
Creating a Pipfile for this project...
Virtualenv already exists!
Removing existing virtualenv...
Creating a virtualenv for this project...
```

The problems that Pipenv seeks to solve are multi-faceted:

- **Create and Manage Python Virtual Environments**
- **Why?**
 - You may have multiple python projects based on different Python versions
 - Helps manage and avoid python package dependency conflicts
 - Enables deterministic builds
- **With very few exceptions, avoid installing Python packages globally**
- **Create a new venv with pipenv and install there**

pipenv – Configuration and Usage



A screenshot of a terminal window showing the contents of the .bash_profile file in a vi editor. The file contains various shell aliases, a prompt configuration usingANSI escape codes, AWS setup code, and the final command to export PIPENV_VENV_IN_PROJECT="true". A red arrow points from the line "export PIPENV_VENV_IN_PROJECT="true"" to a bulleted list of instructions.

```
1 .bash_profile
vi
X
100% 21:1
NORMAL .bash_profile

# aliases
alias dir='ls -al'
alias cls='clear'
alias first-vpc='cd ~/dev/dev.aws/projects/first-vpc'
alias it590='cd ~/dev/dev.classes/it-590-aws'
alias it566='cd ~/dev/dev.classes/it-566-computer-scripting'
alias gasson='cd ~/dev/dev.gasson/marketpredictor'

test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

# prompt
PS1="\n[\033[35m]\$(/bin/date)\n[\033[32m]\w \[\033[1;33m\]\$(__git_ps1
'(%s)')\n[\${iterm2_prompt_mark}]\[\033[1;32m\][\!:#\]\[\033[1;33m\] \u@\h \$ "
test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

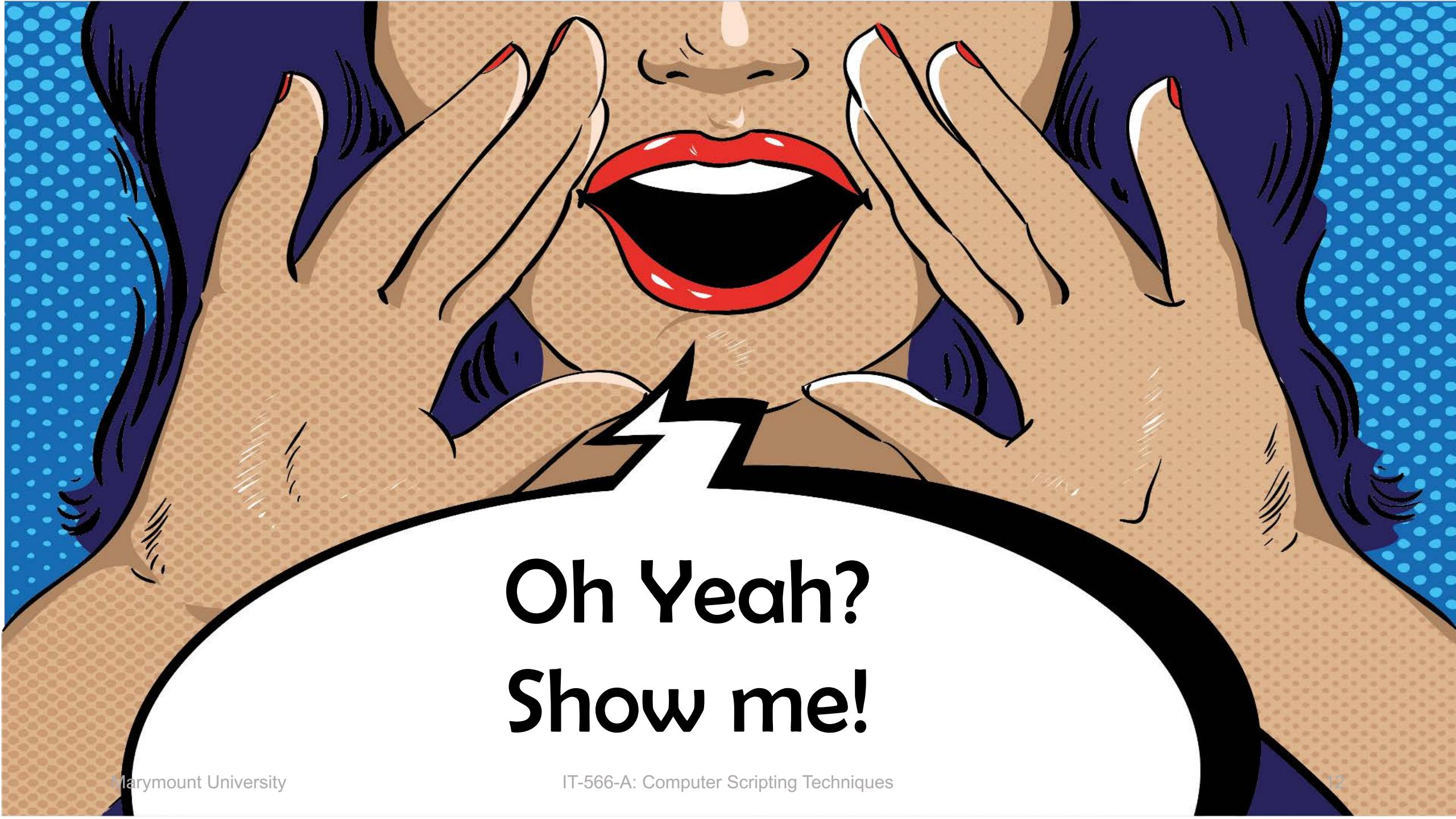
# setup AWS cli environment
~/bin/aws/set-aws.sh
[]

export PIPENV_VENV_IN_PROJECT="true" ←
export PATH
eval "$(opt/homebrew/bin/brew shellenv)"
NORMAL .bash_profile
```

- **export the environment variable:
PIPENV_VENV_IN_PROJECT="true"**
- **Restart shell**

Usage

- pipenv -- python 3
- or be more specific
 - pipenv --python 3.6
 - pipenv --python 3.10
 - etc....
- **Install a package**
 - pipenv install requests
- **This will create a Pipfile**
- ...and a .venv directory



**Oh Yeah?
Show me!**

tree Command

tree Command (MacOS)

```
● ● ●   1 -bash
└ ~ /d/d/it-590-aws/vpc      ↵ dev
    ↵ build  ×  ⟲ ⟳  ⏴ 9/10, 10:42 AM
~/dev/dev.classes/it-590-aws/vpc (dev)
[613:116] swodog@RicksMacPro $ tree
.
├── README.md
├── build.sh
├── cloudformation
│   └── vpc.yaml
└── diagrams
    └── vpc
        ├── CustomVPC.png
        ├── DefaultVPC.png
        └── VPC.pdf

3 directories, 6 files

Sat Sep 10 10:41:35 EDT 2022
~/dev/dev.classes/it-590-aws/vpc (dev)
[614:117] swodog@RicksMacPro $
```

- Tree view of directory structure
- Install with brew

tree Command (Windows)

The diagram illustrates the setup process for the tree command in a Git Bash environment on Windows. It consists of two windows and associated arrows:

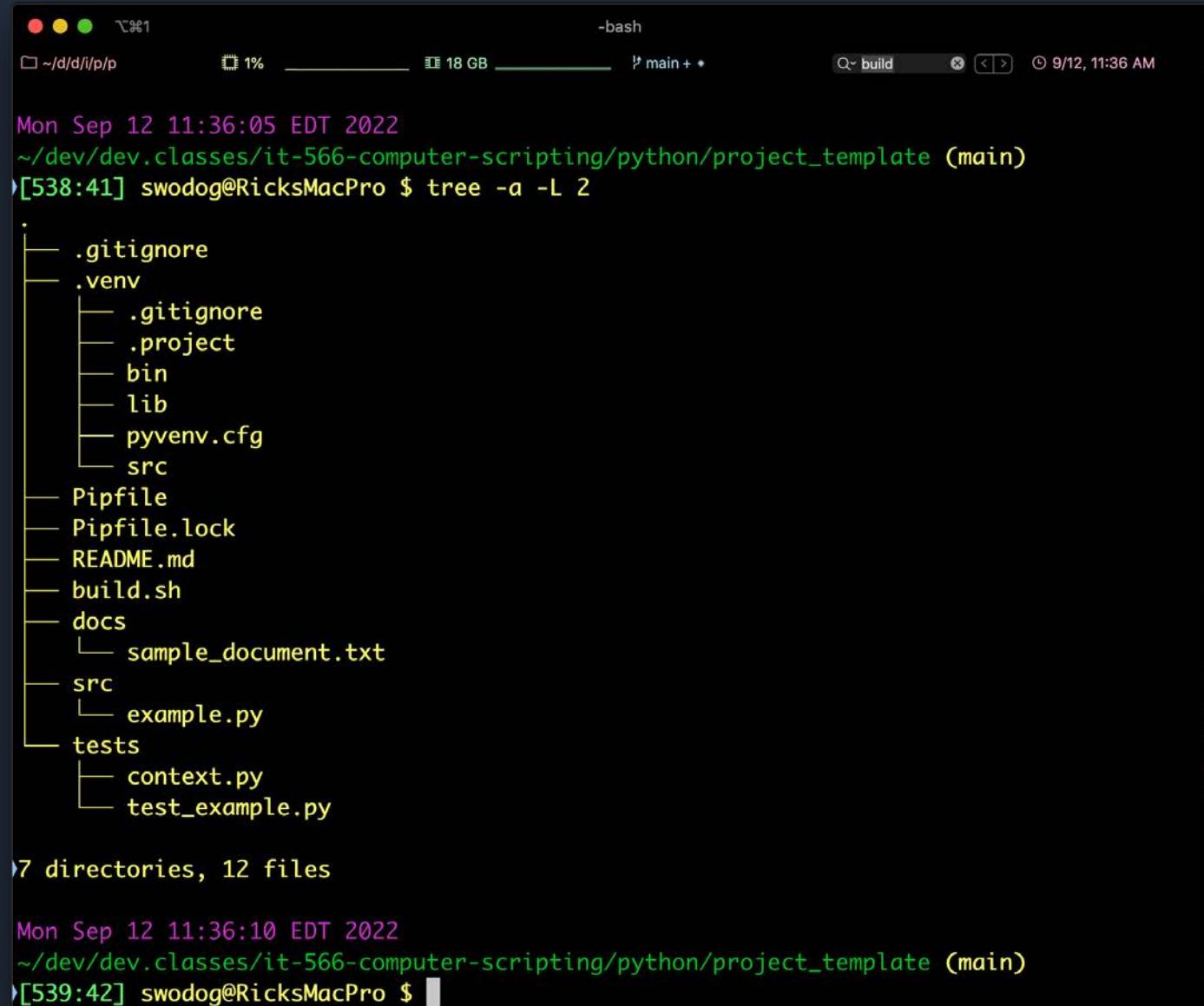
- Top Window (MINGW64 terminal):** Shows the command `alias tree='cmd //c tree //a'` being run in the terminal. A blue arrow points from this window to the text "Add alias to tree command in .bash_profile".
- Bottom Window (.bash_profile editor):** Shows the file `.bash_profile` open in an editor with the message "`-- INSERT --`". The text `alias tree='cmd //c tree //a'` is being typed into the editor. A blue arrow points from this window to the text "Relaunch Git Bash".
- Text Labels:**
 - "Add alias to tree command in .bash_profile"
 - "Relaunch Git Bash"

```
MINGW64:/c/Users/swodog
alias tree='cmd //c tree //a'
~
```

```
.bash_profile [ ] MINGW64:/c/Users/swodog/Projects/it-590-aws/vpc
-- INSERT --
swodog@RICKMILLERA0C8 MINGW64 ~/Projects/it-590-aws/vpc (main)
$ tree
Folder PATH listing
Volume serial number is 8445-6825
C:.
    +---cloudformation
    \---diagrams
        \---vpc

swodog@RICKMILLERA0C8 MINGW64 ~/Projects/it-590-aws/vpc (main)
$
```

tree Command (Show Hidden Files)



The screenshot shows a macOS terminal window with the following details:

- Top bar: Shows battery level (1%), disk usage (18 GB), and a tab labeled "main + *".
- Header: "-bash"
- Timestamp: "Mon Sep 12 11:36:05 EDT 2022"
- Working directory: "~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)"
- User: "swodog@RicksMacPro \$"
- Command: "tree -a -L 2"
- Output:

```
.
├── .gitignore
├── .venv
│   ├── .gitignore
│   ├── .project
│   ├── bin
│   ├── lib
│   └── pyvenv.cfg
└── src
    ├── Pipfile
    ├── Pipfile.lock
    ├── README.md
    ├── build.sh
    ├── docs
    │   └── sample_document.txt
    ├── src
    │   └── example.py
    └── tests
        ├── context.py
        └── test_example.py

7 directories, 12 files
```
- Bottom timestamp: "Mon Sep 12 11:36:10 EDT 2022"
- Bottom working directory: "~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)"
- Bottom user: "swodog@RicksMacPro \$"

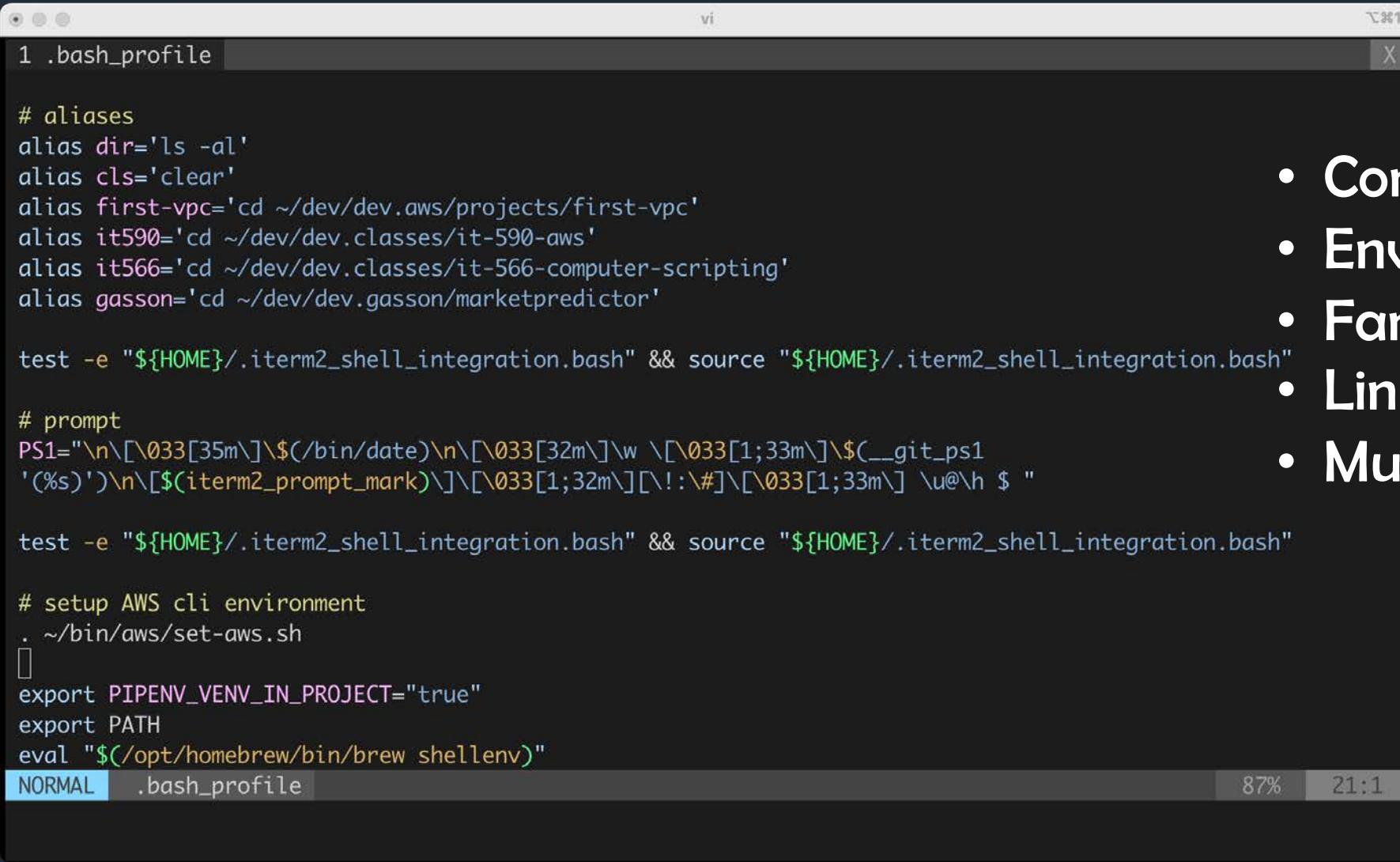
tree -a -L 2



**Really?
Show me!**

.bash_profile Helpful Settings

.bash_profile Helpful Settings



A screenshot of a terminal window titled "vi" showing the contents of the ".bash_profile" file. The file contains several sections of shell script code, including command aliases, environment variables, and a custom prompt setup. The code uses color coding for syntax highlighting.

```
1 .bash_profile

# aliases
alias dir='ls -al'
alias cls='clear'
alias first-vpc='cd ~/dev/dev.aws/projects/first-vpc'
alias it590='cd ~/dev/dev.classes/it-590-aws'
alias it566='cd ~/dev/dev.classes/it-566-computer-scripting'
alias gasson='cd ~/dev/dev.gasson/marketpredictor'

test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

# prompt
PS1="\n[\033[35m]\$(/bin/date)\n[\033[32m]\w \[\033[1;33m\]\$(__git_ps1
'(%s'))\n[\${(iterm2_prompt_mark)}]\[\033[1;32m\][\!:#\]\[\033[1;33m\] \u@\h \$ "

test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

# setup AWS cli environment
. ~/bin/aws/set-aws.sh
[]
export PIPENV_VENV_IN_PROJECT="true"
export PATH
eval "$(opt/homebrew/bin/brew shellenv)"
NORMAL .bash_profile 87% 21:1
```

- Command Aliases
- Environment Variables
- Fancy Prompt
- Links to other shell scripts
- Much, much, more...

Configure SSH for GitHub

Configure SSH for GitHub

The screenshot shows a browser window displaying the GitHub Documentation website at <https://docs.github.com/en/authentication/connecting-to-github-with-ssh>. The page title is "Authentication / Connect with SSH". The left sidebar contains a navigation menu with sections like "All products", "Authentication", "Account security", "Secure your account with 2FA", "Connect with SSH" (which is expanded), "Troubleshooting SSH", "Verify commit signatures", and "Troubleshoot verification". Under "Connect with SSH", there are links for "About SSH", "Check for existing SSH key", "Generate new SSH key", "Add a new SSH key", "Test your SSH connection", and "SSH key passphrases". The main content area is titled "Connecting to GitHub with SSH" and explains that you can connect to GitHub using the Secure Shell Protocol (SSH), which provides a secure channel over an unsecured network. Below this, there is a list of links under the heading "About SSH": "About SSH", "Checking for existing SSH keys", "Generating a new SSH key and adding it to the ssh-agent", "Adding a new SSH key to your GitHub account", "Testing your SSH connection", and "Working with SSH key passphrases". At the bottom of the page, there are two sections: "Did this doc help you?" with thumbs up and down icons, and "Help us make these docs great!" with a "Make a contribution" button and a note about GitHub docs being open source.

- **Secure Repository Connection**
- **SSH Key Generation and Installation**
- **Can be confusing**
- **Let's walk through**



GitHub SSH Key Generation and Configuration

Project Organization & Layout

Directory Structure & Project Artifacts

Python Modules & Packages



There's Bill, I'll ask him...

Hey Bill, me and Beth are having relationship issues. What do you think I should do?

Fix your Python project structure!

Project Organization

- Project Organization Refers to Several Issues
 - Directory Structure and Layout
 - Source Code Organization
 - Module and Package Organization
 - Application Architecture
- Today — Focus on Directory Structure and Project Artifacts
- Later Today — Focus on Application Architecture
 - ...and how to organize source code into modules and packages

Directory Structure & Project Artifacts

Directory Structure and Project Artifacts

```
Mon Sep 12 10:14:04 EDT 2022
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[526:29] swodog@RicksMacPro $ tree -a -L 1
.
├── .gitignore
├── .venv
├── Pipfile
├── Pipfile.lock
├── README.md
├── build.sh
├── docs
└── src
└── tests

4 directories, 5 files

Mon Sep 12 10:14:22 EDT 2022
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[527:30] swodog@RicksMacPro $
```

.gitignore — List of files not to add to repo
.venv — Virtual Environment created by pipenv
Pipfile — List of required project packages
Pipfile.lock — List of specific package versions
README.md — Markdown document displayed
build.sh — Bash build file
docs — Documentation directory
src — Python source code directory
tests — Unit and integration tests directory

`.gitignore` – List of files not to add to repo

.venv – Virtual Environment created by pipenv

Pipfile – List of required project packages

Pipfile.lock – List of specific package versions

README.md – Markdown document displayed by GitHub

build.sh – Bash build file

docs – Documentation directory

src — Python source code directory

tests — Unit and integration tests directory

.gitignore

```
1 .gitignore
# Stray MacOS files
.DS_Store

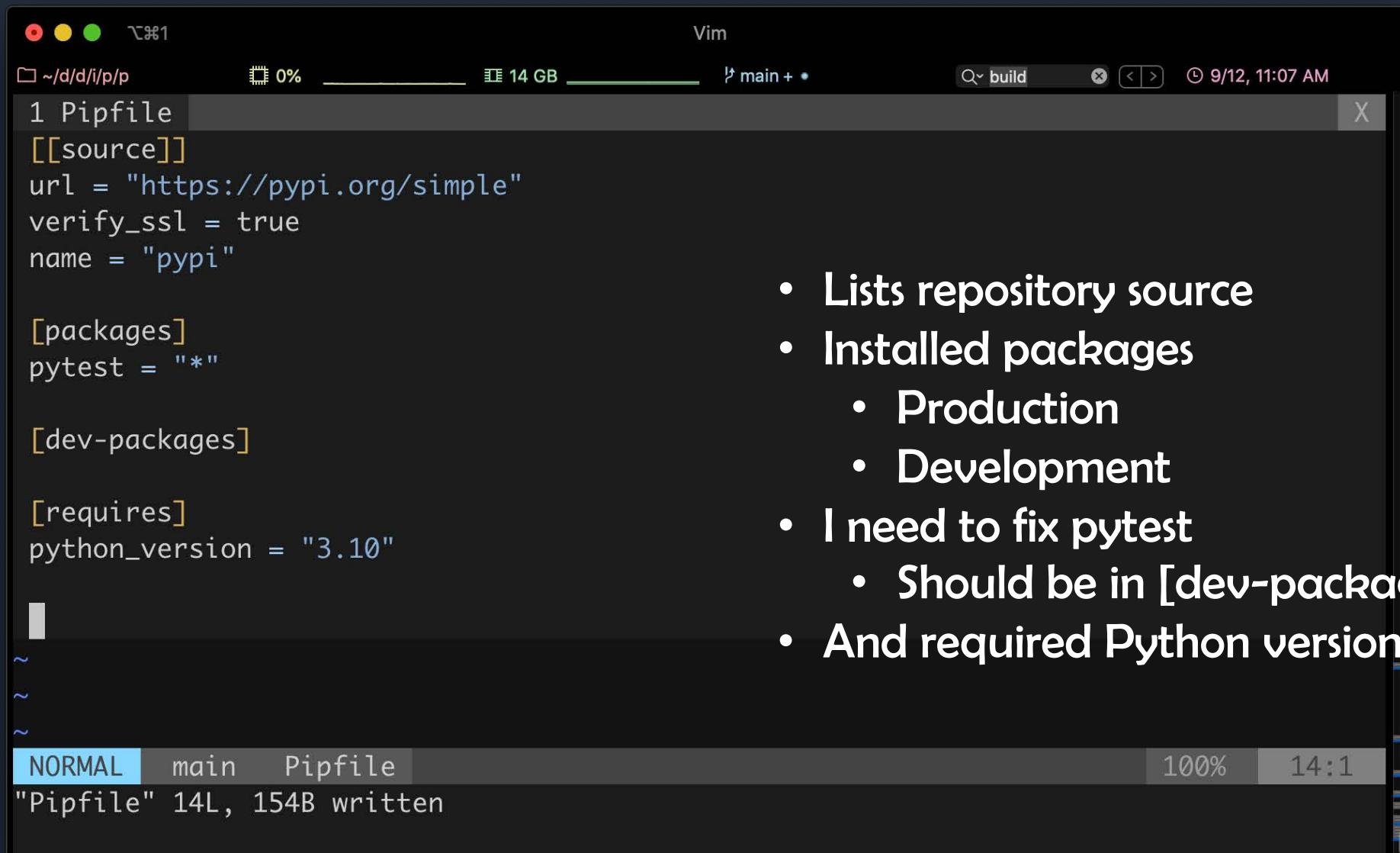
# Virtual Environments
.venv

NORMAL main .gitignore
".gitignore" 7L, 62B written
```

- List files and directories to ignore
- Can add when creating repository
- This is a very simple example
- I like to start simple...
 - And add items as I go
- Let's see a complete python .gitignore

- List files and directories to ignore
 - Can add when creating repo in GitHub
 - This is a very simple example
 - I like to start simple...
 - And add items as I go
 - Let's see a complete python .gitignore

Pipfile



A screenshot of a Vim editor window displaying a `Pipfile`. The file contains the following configuration:

```
1 Pipfile
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
pytest = "*"

[dev-packages]

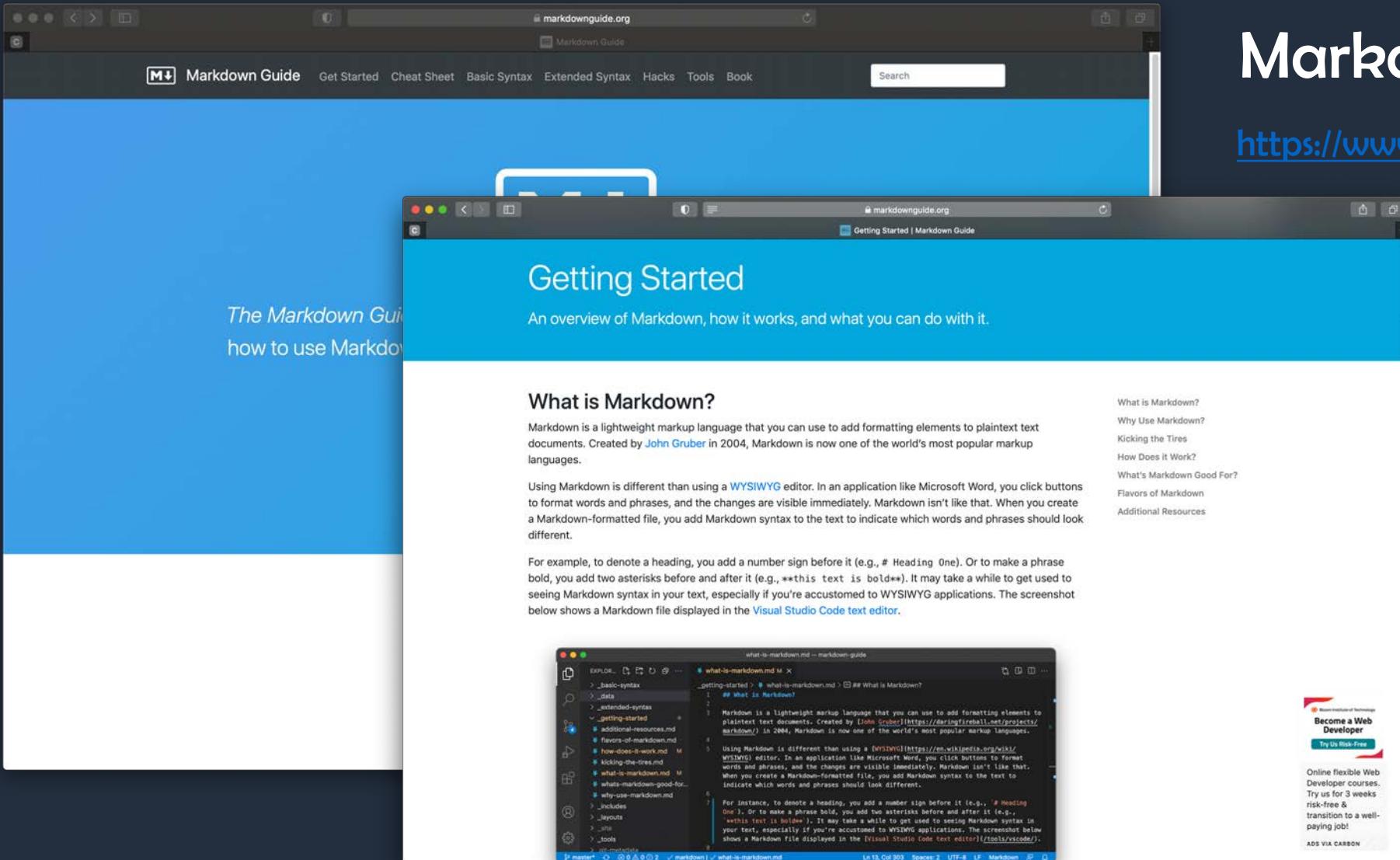
[requires]
python_version = "3.10"
```

The Vim status bar at the bottom shows the mode as `NORMAL`, the buffer name as `main`, the file name as `Pipfile`, and the message `"Pipfile" 14L, 154B written`. The status bar also indicates `100% 14:1`.

To the right of the Vim window, a list of bullet points highlights key features of the `Pipfile`:

- Lists repository source
- Installed packages
 - Production
 - Development
- I need to fix pytest
 - Should be in [dev-packages] section
- And required Python version

README.md



Markdown File

<https://www.markdownguide.org>

Directory Structure and Project Artifacts

```
● ● ●  ~d/i/p/p -bash
0% 14 GB main + •  build  9/12, 10:51 AM
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[527:30] swodog@RicksMacPro $ tree
.
├── Pipfile
├── Pipfile.lock
├── README.md
├── build.sh
├── docs
│   └── sample_document.txt
└── src
    └── example.py
└── tests
    ├── context.py
    └── test_example.py
3 directories, 8 files
Python source files go in src directory
Python unit and integration tests go in tests folder
context.py simplifies Python module and package location for unit tests
Mon Sep 12 10:50:31 EDT 2022
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[528:31] swodog@RicksMacPro $
```



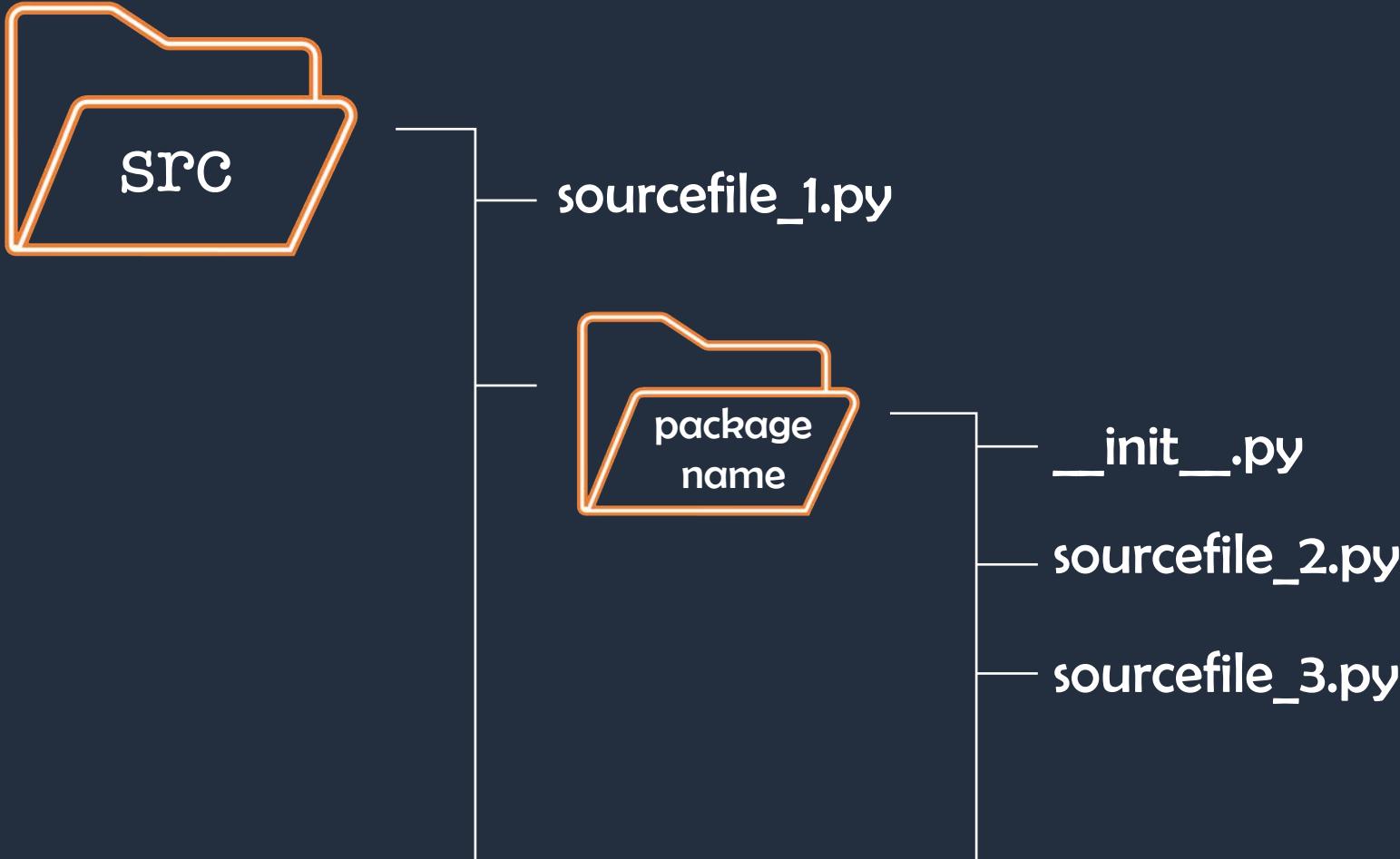
Maybe they
won't notice...

Hey, not so fast.
What about the
Pipfile.lock?

Python Modules and Packages

Python Modules and Packages

<https://docs.python.org/3/reference/import.html#namespace-packages>



Bash Build Script

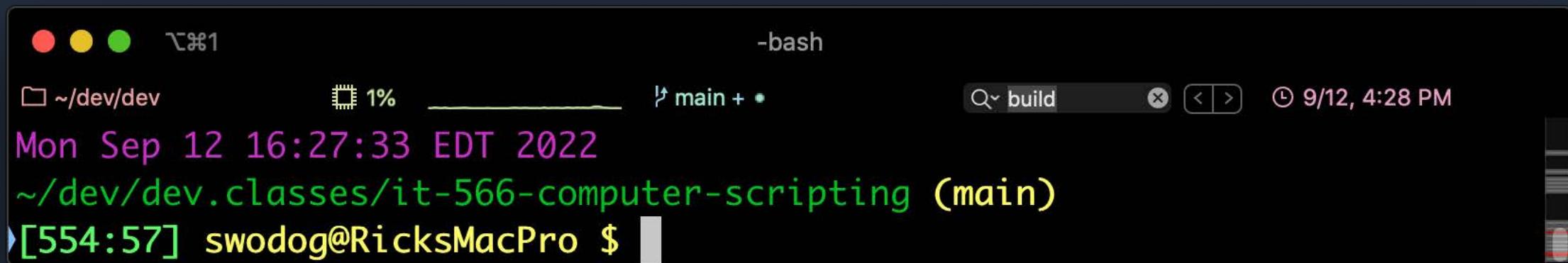
Bash Build Script (build.sh)

- Simplifies Project Management
- Consolidate Complex Commands
- Let's take a look...

Git Development Workflow

Git Development Workflow

- Initial Activities
 - Create GitHub Repository
 - Add README.md and .gitignore
 - You will either start with a **main** or a **master** branch
- Clone The Repository Locally
 - `git clone git@github.com:repository_name.git`
 - Example: `git clone git@github.com:pulpfreepress/it-566-computer-scripting.git`



A screenshot of a macOS terminal window titled "-bash". The window shows a cloned Git repository at `~/dev/dev`. The status bar indicates the current branch is `main + •`. The terminal output shows the date and time as `Mon Sep 12 16:27:33 EDT 2022`, the repository path as `~/dev/dev.classes/it-566-computer-scripting (main)`, and the user's prompt as `[554:57] swodog@RicksMacPro $`.

Git Development Workflow

```
● ● ● 361 -bash
~ /d/d/i/p/p 2% 20 GB main + *
Mon Sep 12 16:32:41 EDT 2022
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[562:65] swodog@RicksMacPro $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified: ../../.DS_Store
modified: ../../Presentations/Week_2.pptx
modified: .gitignore
modified: Pipfile

Untracked files:
(use "git add <file>..." to include in what will be committed)

    ../../Presentations/~$Week_2.pptx
    docs/

no changes added to commit (use "git add" and/or "git commit -a")

Mon Sep 12 16:32:48 EDT 2022
~/dev/dev.classes/it-566-computer-scripting/python/project_template (main)
[563:66] swodog@RicksMacPro $
```

- Add, Modify, Delete Files and Directories
- Check Status with `git status`

Followed by:

`git add .`
`git commit -m “commit message”`
`git push`



Would you
like to see a
demo?

Yes darling!