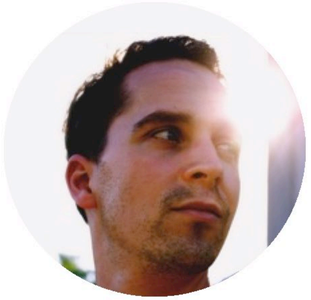
Runtimes and Tools

Efforts to cope with ~~package &~~ version management



Henry Stamerjohann, MDO:YVR - June 16 2022

zentral
pro
services

MAC
DEV
OPS
YVR

Goal 

MAC
DEV
OPS
YVR

Goal

Create and maintain a local development environment

Who, why, and how?

Create and maintain a local development environment

MAC
DEV
OPS
YVR

Administrators, Developers, DevOps

Day by day, collaboration, and evaluation

Take care of version conflicts and constraints

Tools help to manage runtimes and dependencies

What is the issue here?

Platform obstacles

MAC
DEV
OPS
YVR

✅ Linux - existing Package Managers

→DPKG, RPM, Pacman, Zypper

❌ macOS - no built-in Package Manager

→Historically grown approaches

Packages, management?

Apple specific helpers

mac OS

MDO:YVR 2018:
Let's talk about brew!

You Tube

3.1.11. port install

The action `install` is used to install a port. Once you determined the name of a port you want (possibly using port search), you can install it using this command. See Section 3.2.1, "Invoking Variants" on how to choose variants when installing a new port. For example,

```
$ sudo port install apache2 -preforkmpm +workermpm
```

installs the `apache2` port without the `preforkmpm`, but with the `workermpm` variant.

If the installation of a port fails, you can enable verbose or debug output by giving the `-v` or `-d` flag to port:

```
$ sudo port -v install apache2
```

All debug information is also kept in `main.log` for the port you installed. Its path will be printed automatically if the installation fails. You can manually get the path using **port logfile portname**. Note that logfiles will automatically be deleted on successful installation.

If the installation of a port fails, you should always clean and try again, i.e., run

```
$ sudo port clean portname
```

and re-execute the command you ran before.

You might also want to try enabling trace mode, which can prevent conflicts caused by files installed by other ports or in common system locations, such as `/usr/local`. To do that, re-run the installation with the `-t` flag, i.e.,

```
$ sudo port -t install portname
```

If the port still fails to install after you have followed these steps, please file a ticket and attach the `main.log` of a clean attempt.

> **Note**
>
> The installation of a single port consists of multiple phases. These phases are fetch, extract, patch, configure, build, destroot, archive, and finally install. You may break up a port's installation into smaller steps for troubleshooting by using the name of one of these phases as action rather than `install`. For example
>
> ```
> $ sudo port destroot apache2
> ```
>
> will run the installation of `apache2` until the destroot phase. See Section 5.3, "Port Phases" for a complete list of phases and a detailed description.

`install` takes the following switches:

`--no-rev-upgrade`

By default, a binary sanity check called `rev-upgrade` is run automatically after each successful installation. Pass this flag, if you want to avoid running this step, for example if you want to run it explicitly later after a number of installations using **sudo port rev-upgrade**, or if you know it will detect problems but want to defer dealing with them.

`--unrequested`

By default, each port you install using the `install` explicitly (contrary to ports installed as a dependency of a different port) is marked as "requested". If you want MacPorts to treat a port you installed manually as if it was automatically installed as a dependency (e.g., if a dependency failed to build and you re-tried installing the dependency only), pass this flag.

3.1.12. port notes

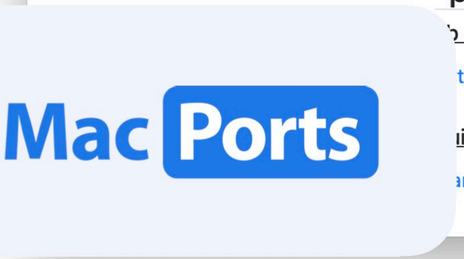
The `notes` action is used to display any notes that a port's author included. These can contain anything, but by convention are brief, and typically contain quick start steps for configuring and using the port, pitfalls to watch out for, or other information that users should be aware of. These same notes are also displayed after installing a port. Many ports have no notes. More extensive documentation can often be found at a port's homepage, or in its installed files.

```
$ port notes xinit
```

```
--->  xinit has the following notes:
  To use MacPorts' X11 as the default server, install xorg-server, log out, and
  log back in.
```

3.1.13. port clean

The action `clean` deletes intermediate files created by MacPorts while installing a port. A **port clean** is often necessary when builds fail and should be



ports.macports.org

MacPorts | Search for ports... | Home | Ports | Builds | Statistics | Watchlist

py39-python-install (python/py-python-install)

+ Add to my watchlist 0

A simple, correct PEP427 wheel installer.

Version: 0.0.3 **License:** MIT GitHub

Details | Builds | Installation Stats | Trac Tickets 0 | Report an Issue with this port | API | Always skip to Details page

Maintainers	jmroot
Categories	python
Homepage	https://pypi.python.org/pypi/python-install/
Platforms	darwin
Variants	-

Subport(s) (5)

py310-python-install py36-python-install py37-python-install py38-python-install
py-python-install

"py39-python-install" depends on

(1)

thon39

ild (3)

ang-9.0 py-bootstrap-modules py39-setuptools

Port Health:

✓ Monterey (arm64)	Files (45)
? Monterey (x86_64)	
✓ Big Sur (arm64)	Files (45)
✓ Big Sur (x86_64)	Files (45)
✓ Catalina	Files (45)
✓ Mojave	Files (45)
✓ High Sierra	Files (45)
✓ Sierra	Files (45)
✓ El Capitan	Files (45)
✓ Yosemite	Files (45)
✓ Mavericks	Files (45)
✓ Mountain Lion	Files (45)
✓ Lion	Files (45)
✓ Snow Leopard (x86_64)	Files (45)
✓ Snow Leopard (i386)	Files (45)
? Leopard (ppc legacy)	

? - No history in app's database

INSTALLATIONS (30 DAYS)

66

REQUESTED INSTALLATIONS (30 DAYS)

search.nixos.org

Back to nixos.org | Packages | Options | Flakes Experimental

Search more than **80 000 packages**

python

Search

Channel: 21.11 | 22.05 (Beta) | unstable

Package sets

- python38Packages 4750
- python39Packages 4742
- No package set 269
- haskellPackages 33
- emacs27Packages 21
- kodiPackages 9
- vimPlugins 8
- pythonDocs 8
- ocamlPackages 4
- matrix-synapse-plugins 4
- nodePackages_latest 3
- nodePackages 3
- gnome2 3
- gnome 3
- acheHttpdPackages 3
- icklispPackagesClisp 2
- ver 2
- Packages 2

Showing results 1-50 of **9894 packages.**

Sort: Best match

pythonFull

A high-level dynamically-typed programming language

Name: `python` Version: **2.7.18** Homepage Source License: Python Software Foundation License version 2

python38

A high-level dynamically-typed programming language

Name: `python3` Version: **3.8.13** Outputs: `out` `debug` Homepage Source
License: Python Software Foundation License version 2

▲▲▲ Hide package details ▲▲▲

How to install *python38*? On non-NixOS | On NixOS

```
$ nix-env -iA nixos.python38
```

Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains. Some of its key distinguishing features include: clear, readable syntax; strong introspection capabilities; intuitive object orientation; natural expression of procedural code; full modularity, supporting hierarchical packages; exception-based error handling; and very high level dynamic data types.

Maintainers

- Frederik Rietdijk <fridh@fridh.nl>

Platforms

- aarch64-linux
- i686-linux
- x86_64-linux
- x86_64-darwin
- aarch64-darwin

NixOS

docs.brew.sh

Fork me on GitHub

Homebrew Documentation

Search Documentation

Python

This page describes how Python is handled in Homebrew for users. See **Python for Formula Authors** for advice on writing formulae to install packages written in Python.

Homebrew should work with any **CPython** and defaults to the macOS system Python.

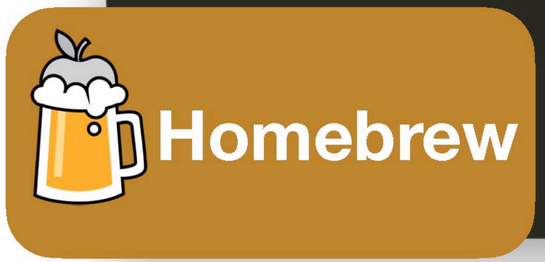
Homebrew provides formulae to brew Python 3.y. A `python@2` formula was provided until the end of 2019, at which point it was removed due to the Python 2 deprecation.

Important: If you choose to use a Python which isn't either of these two (system Python or brewed Python), the Homebrew team cannot support any breakage that may occur.

Python 3.y

Homebrew provides formulae for maintained releases of Python 3.y (`python@3.y`).

Important: Python may be upgraded to a newer version at any time. Consider using a version manager such as `pyenv` if you require stability of minor or patch versions for virtual environments.



README.md

Simple Python Version Management: pyenv

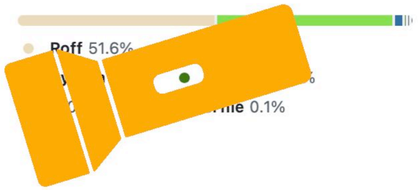
gitter join chat

pyenv lets you easily switch between multiple versions of Python. It's simple, unobtrusive, and follows the UNIX tradition of single-purpose tools that do one thing well.

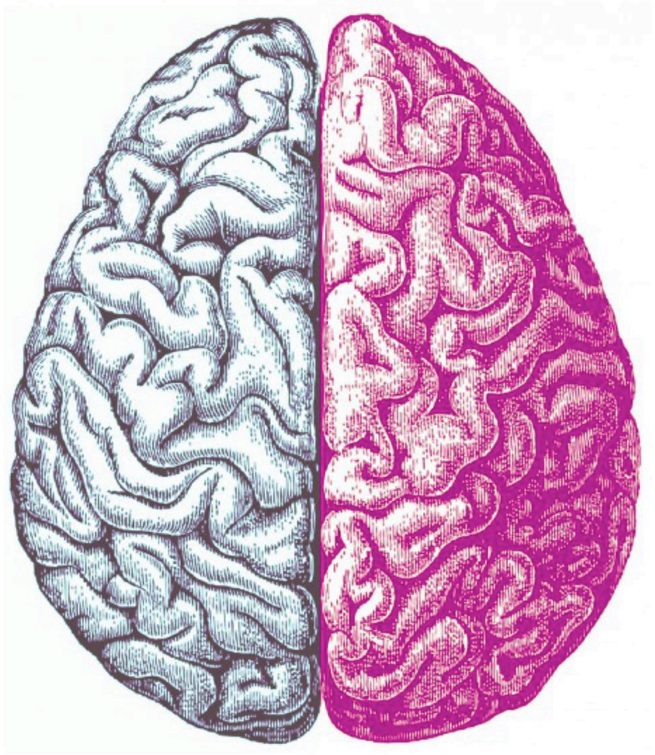
This project was forked from rbenv and ruby-build, and modified for Python.

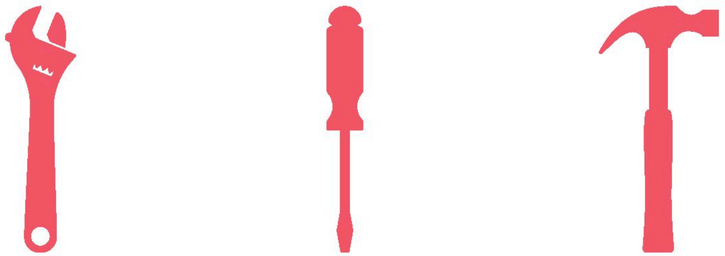
```
$ pyenv versions
  2.7.10
* 3.5.0 (set by /Users/yuu/.pyenv/version)
  miniconda3-3.16.0
  pypy-2.6.0

$ python --version
Python 3.5.0

$ pyenv global pypy-2.6.0

$ python --version
Python 2.7.9 (295ee98b69288471b0fcf2e0ede82ce5209eb90b, Jun 01 2015, 17:30:13)
[PyPy 2.6.0 with GCC 4.9.2]

$ cd /Volumes/treasuredata/jupyter

$ pyenv version
miniconda3-3.16.0 (set by /Volumes/treasuredata/.python-version)

$ python --version
Python 3.4.3 :: Continuum Analytics, Inc.
```

Languages

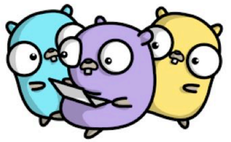


Packages

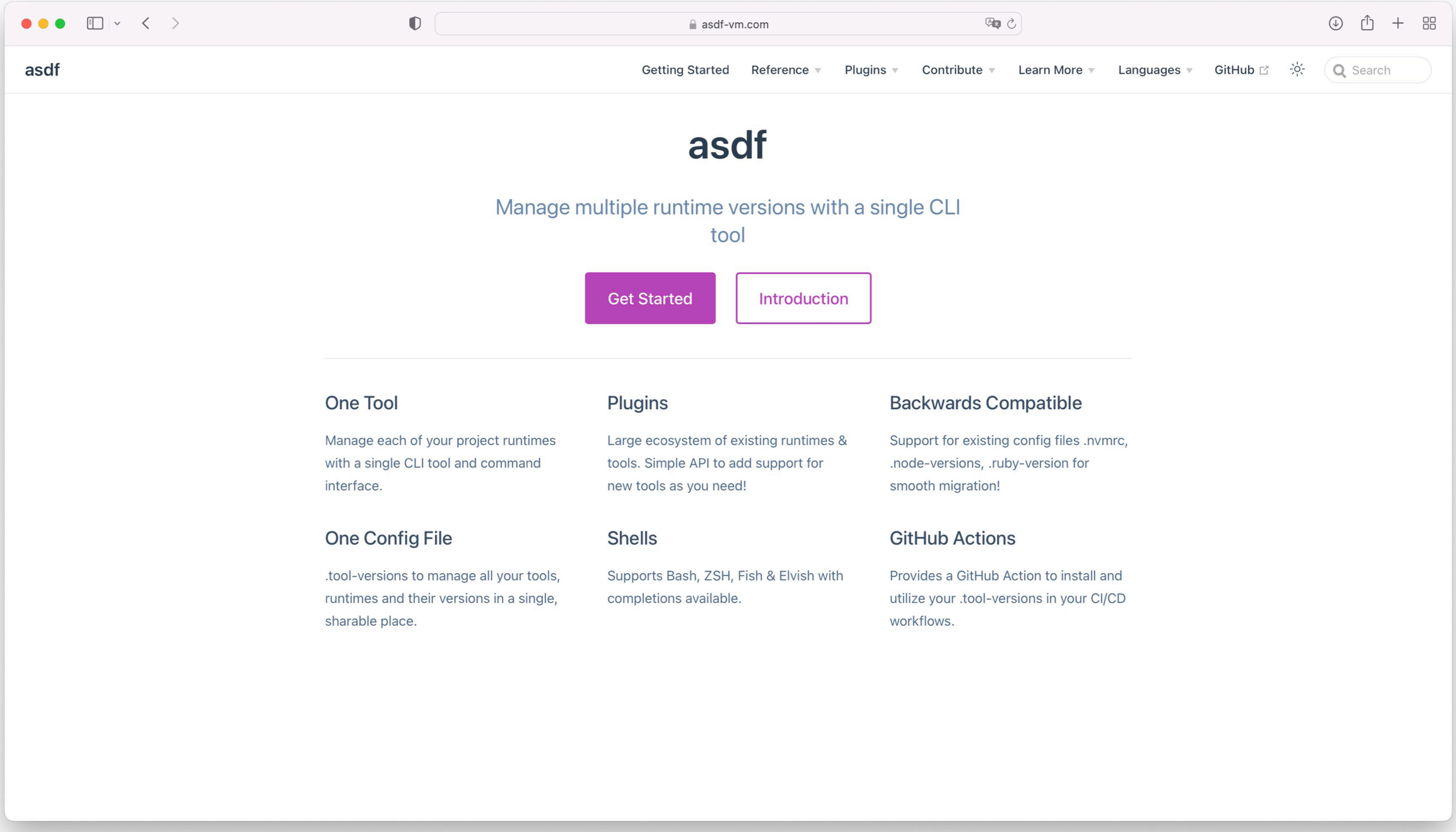


Versions
+Dependencies



Pipenv Poetry

Keep it stupid simple


asdf-vm.com
asdf
Getting Started
Reference
Plugins
Contribute
Learn More
Languages
GitHub
Search
asdf
Manage multiple runtime versions with a single CLI tool
Get Started
Introduction
One Tool
Manage each of your project runtimes with a single CLI tool and command interface.
Plugins
Large ecosystem of existing runtimes & tools. Simple API to add support for new tools as you need!
Backwards Compatible
Support for existing config files .nvmrc, .node-versions, .ruby-version for smooth migration!
One Config File
.tool-versions to manage all your tools, runtimes and their versions in a single, sharable place.
Shells
Supports Bash, ZSH, Fish & Elvish with completions available.
GitHub Actions
Provides a GitHub Action to install and utilize your .tool-versions in your CI/CD workflows.

Homebrew

> The Missing Package Manager for macOS (or Linux)

Homebrew manages your packages and their upstream dependencies. `asdf` does not manage upstream dependencies, it is not a package manager, that burden is upon the user, though we try and keep the dependency list small.

See Homebrew docs for more.

NixOS

> Nix is a tool that takes a unique approach to package management and system configuration

NixOS aims to build truly reproducible environments by managing exact versions of packages up the entire dependency tree of each tool, something `asdf` does not do. NixOS does this with its own programming language, many CLI tools and a package collection of over 60,000 packages.

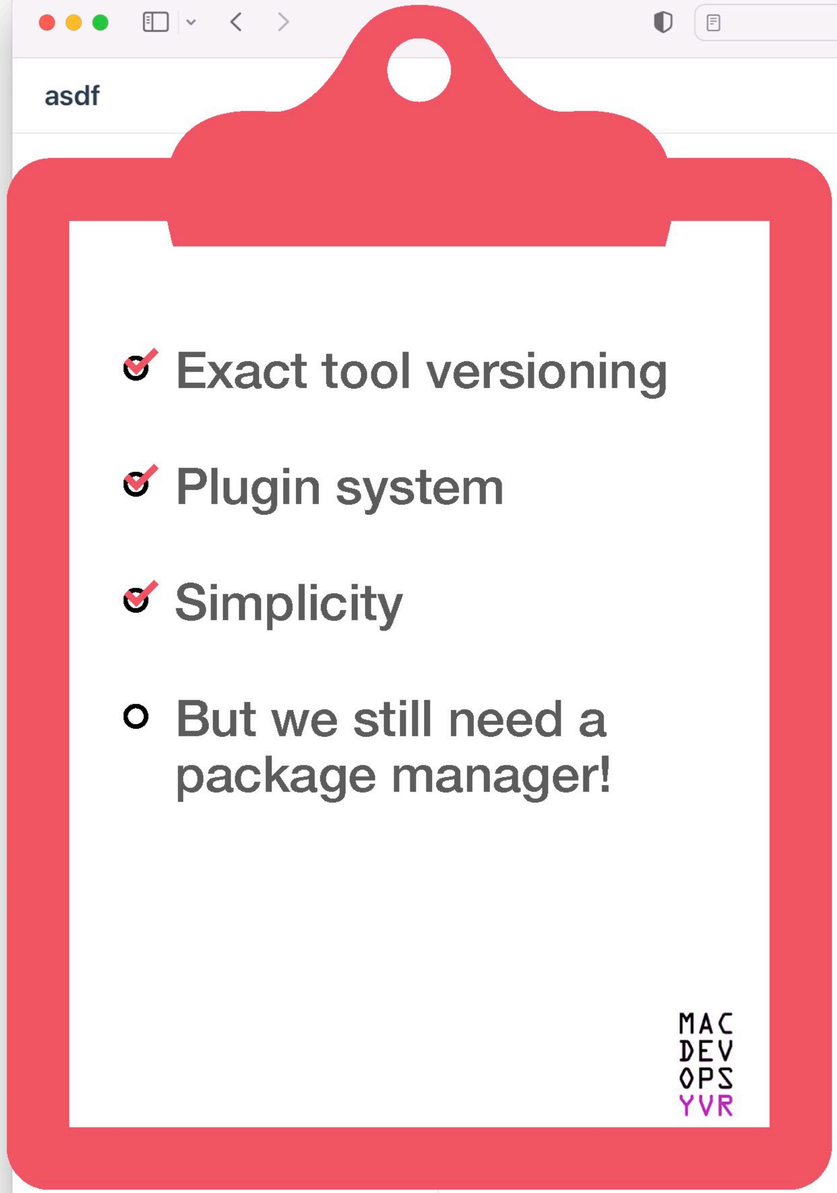
Again, `asdf` does not manage upstream dependencies and is not a package manager.

See NixOS docs for more.

Why use asdf?

`asdf` ensures teams are using the **exact** same versions of tools, with support for **many** tools via a plugin system, and the *simplicity and familiarity* of being a single **Shell** script you include in your Shell config.

> **Note**
>
> `asdf` is not intended to be a system package manager. It is a tool version manager. Just because you can create a plugin for any tool and manage its versions with `asdf`, does not mean that is the best course of action for that specific tool.



asdf-vm.com

asdf

Getting Started | Reference | Plugins | Contribute | Learn More | Languages | GitHub | Search

Homebrew

> The Missing Package Manager for macOS (or Linux)

Homebrew manages your packages and their upstream dependencies. `asdf` does not manage upstream dependencies, it is not a package manager, that burden is upon the user, though we try and keep the dependency list small.

See Homebrew docs for more.

NixOS

> Nix is a tool that takes a unique approach to package management and system configuration

NixOS aims to build truly reproducible environments by managing exact versions of packages up the entire dependency tree of each tool, something `asdf` does not do. NixOS does this with its own programming language, many CLI tools and a package collection of over 60,000 packages.

Again, `asdf` does not manage upstream dependencies and is not a package manager.

See NixOS docs for more.

Why use asdf?

`asdf` ensures teams are using the **exact** same versions of tools, with support for **many** tools via a plugin system, and the *simplicity and familiarity* of being a single **Shell** script you include in your Shell config.

> **Note**
>
> `asdf` is not intended to be a system package manager. It is a tool version manager. Just because you can create a plugin for any tool and manage its versions with `asdf`, does not mean that is the best course of action for that specific tool.

`asdf` installation involves:

1. Installing dependencies
2. Downloading `asdf` core
3. Installing `asdf`
4. Installing a plugin for each tool/runtime you wish to manage
5. Installing a version of the tool/runtime
6. Setting global and project versions via `.tool-versions` config files

1. Install Dependencies

Linux:

> **Note**
>
> `sudo` may be required depending on your system configuration.

Package Manager	Command
Aptitude	`apt install curl git`
DNF	`dnf install curl git`
Pacman	`pacman -S curl git`
Zypper	`zypper install curl git`

macOS:

Package Manager	Command
Homebrew	Dependencies will be automatically installed by Homebrew.
Spack	`spack install coreutils curl git`

Untangling Depenedecies

Streamlined installation procedure

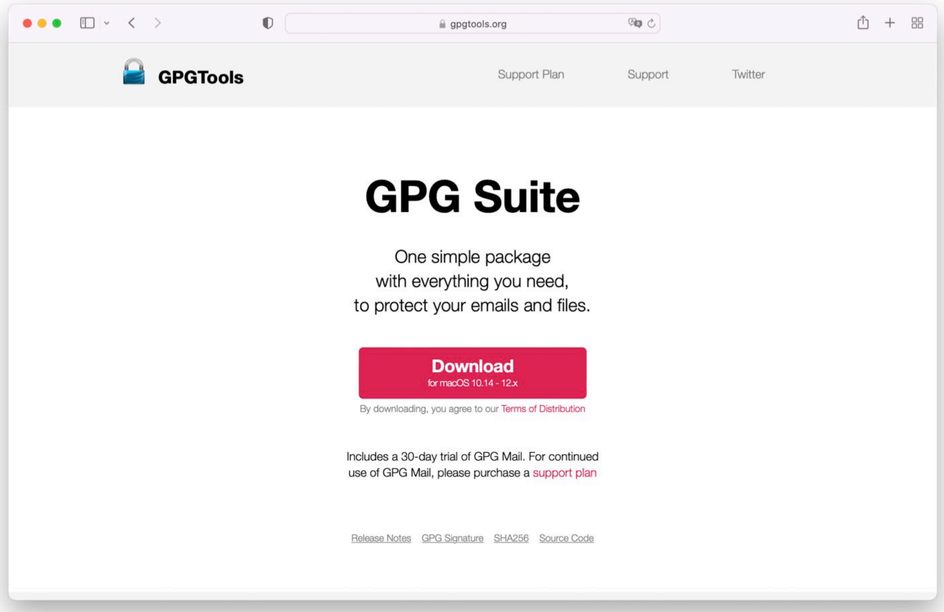
Method A - Homebrew

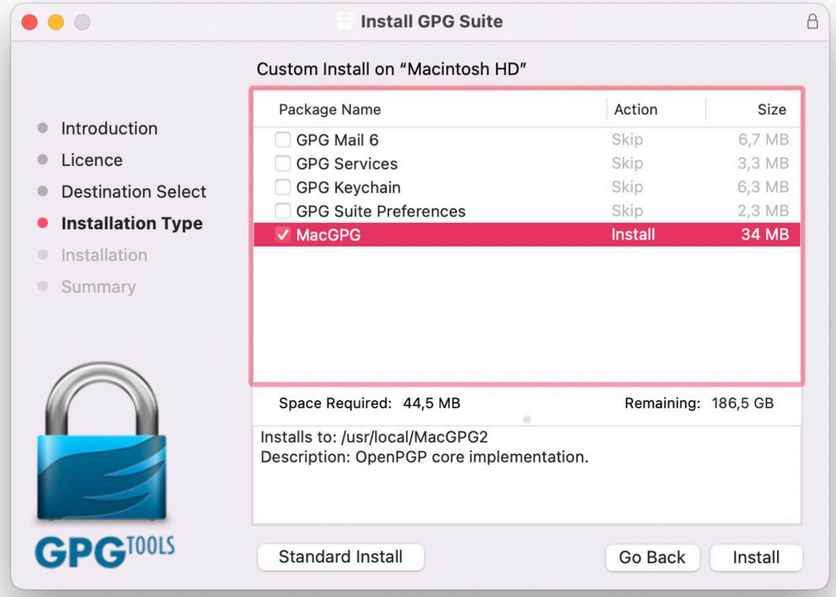
Untangling Depenedecies

Streamlined installation procedure

Method A - Homebrew

Method B - Utilize GPG Tools for the remainder dependency `[coreutils]`





Fast-track …

Installation & Setup

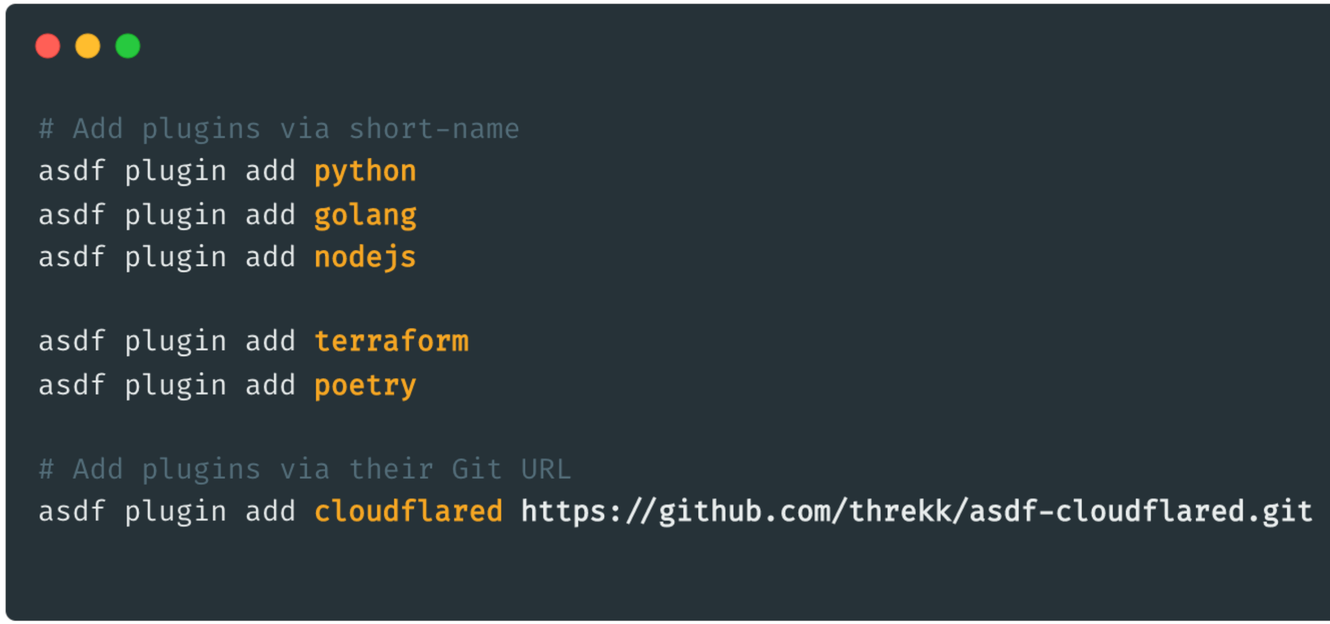
1) Install Xcode CLI tools
2) Install coreutils
3) Clone asdf repository
4) Add entries to ~/.zshrc

```
# 1 - Install Xcode CLI tools
# 2 - Install coreutils via brew or GPG Tools

# 3 - clone asdf repository
git clone https://github.com/asdf-vm/asdf.git ~/.asdf --branch v0.10.1

# 4 - Add the following to ~/.zshrc
. $HOME/.asdf/asdf.sh
# append completions
fpath=(${ASDF_DIR}/completions $fpath)
# compinit ZSH completions
autoload -Uz compinit && compinit
```

Manage Plugins

Efficient installation process

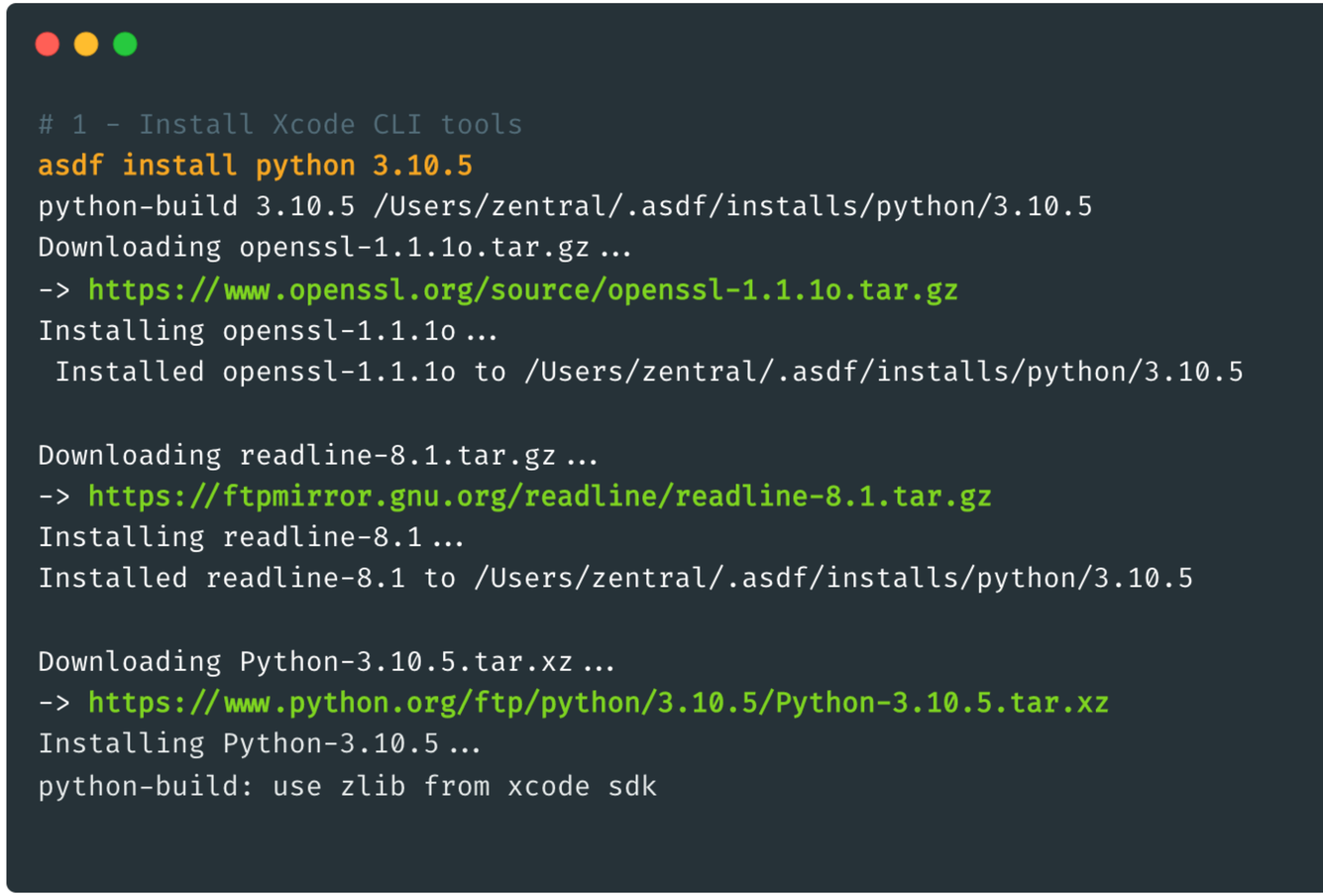
```
asdf plugin add <name>

asdf plugin add <name> <url>
```



Install runtime versions

Download from source

```
asdf install <name> <vers>
asdf install <name> latest
```

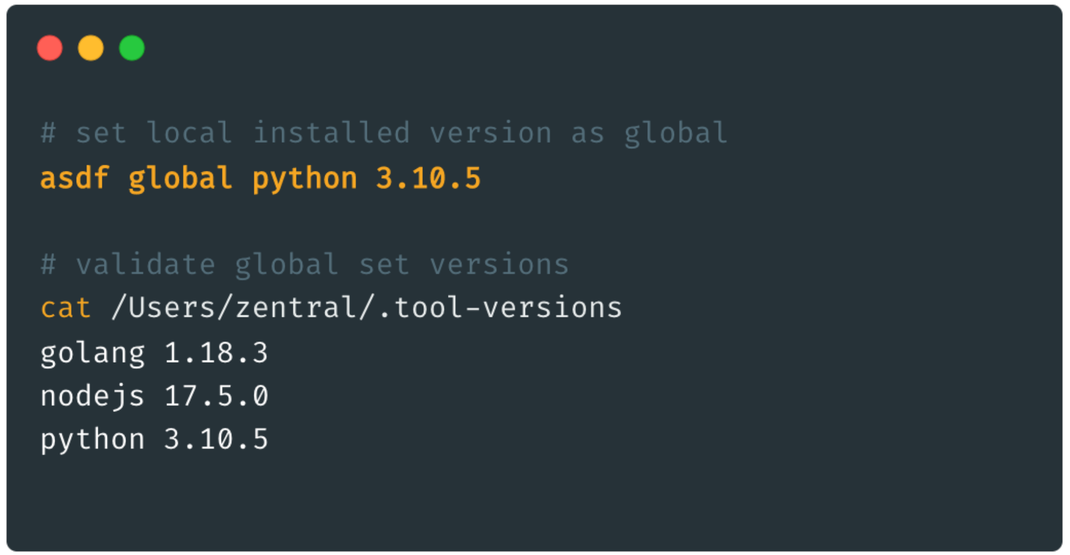


Set version

Set global or local version

```
asdf global <name> <vers>
```

```
asdf local <name> <vers>
```



3 step process

It’s simple as that

1) Install a Plugin
2) Install a Version*
3) Set a Version

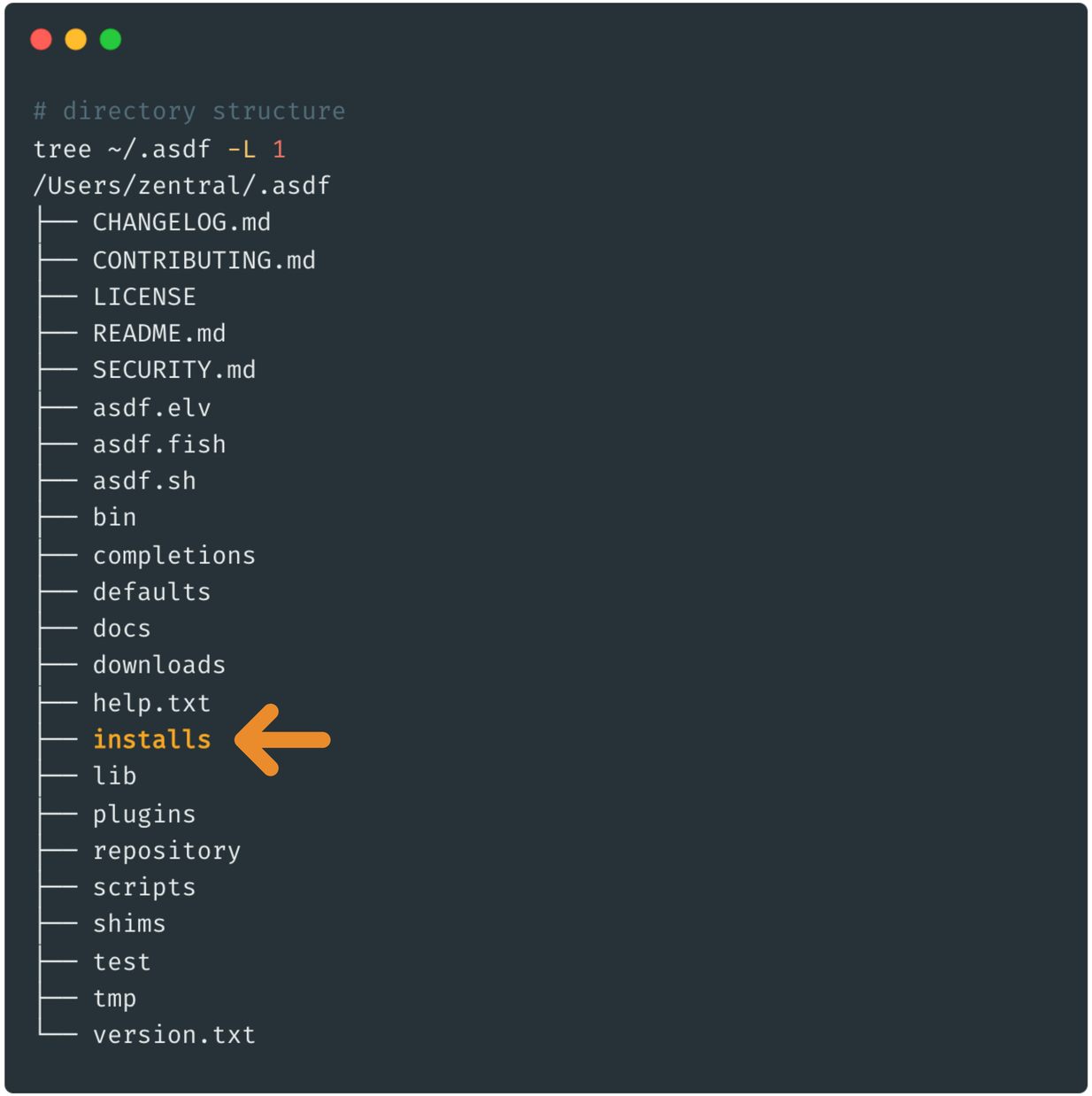
*optional: include default packages

So what’s more to come?

Folders structure

Content

~/.asdf



Folder structure

Versions

`~/.asdf/installs/<name>`

Folder structure

Binaries and Libraries

Common directory structure

Runtime executable files/binary

Core dependencies

Neat features

Plugin specific default packages

`$HOME/.default-python-packages`

Pipenv

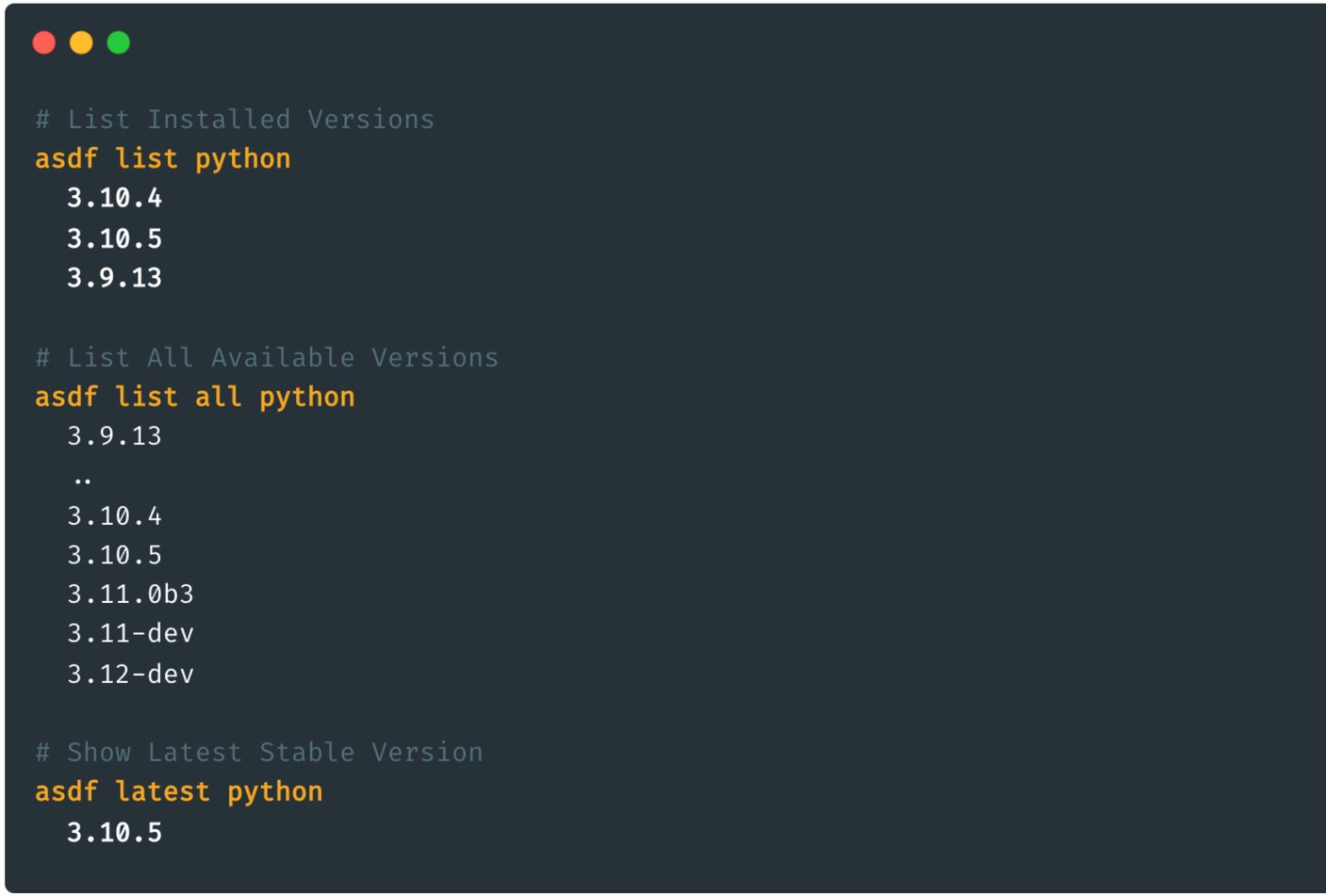
```
cat $HOME/.default-python-packages
ansible
pipenv
```

Install runtime versions

Locate versions

```
asdf list <name>
```

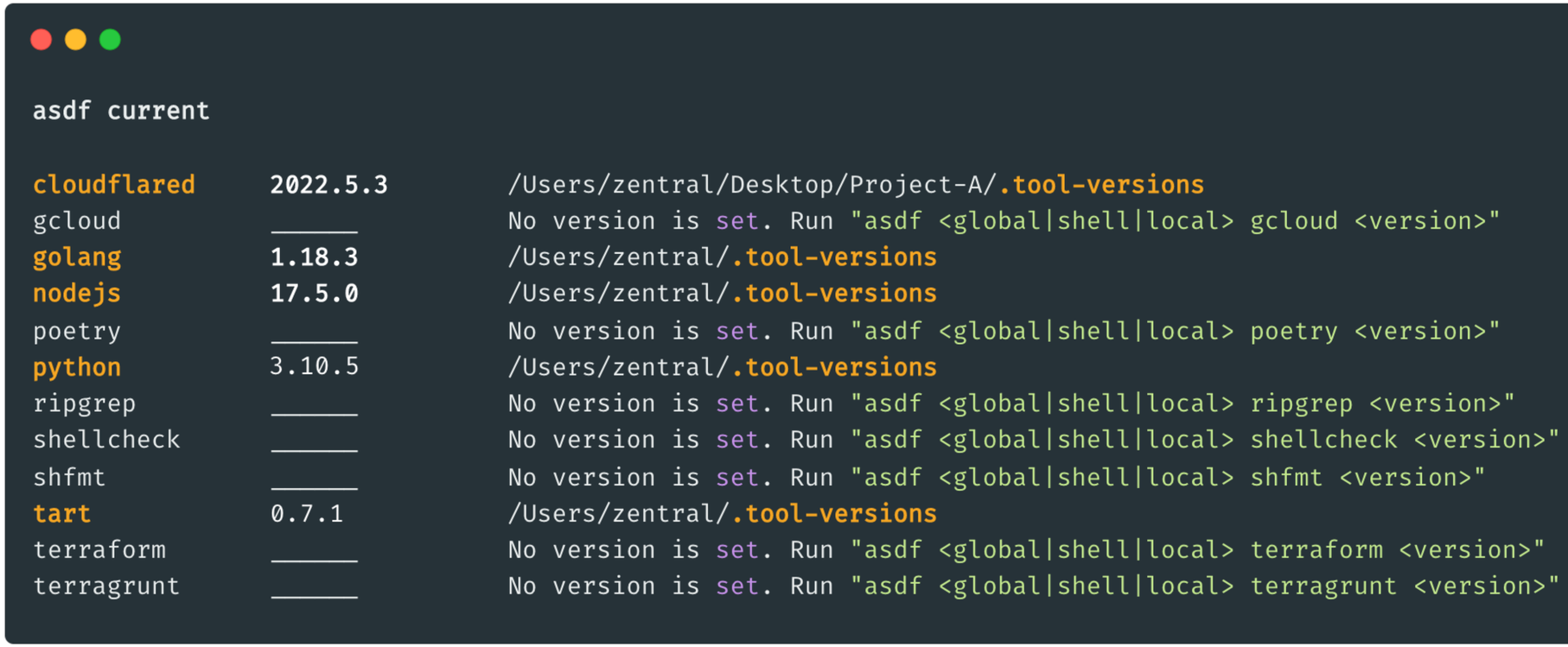
```
asdf list all <name>
```



Update tooling

Extra tools & dependencies

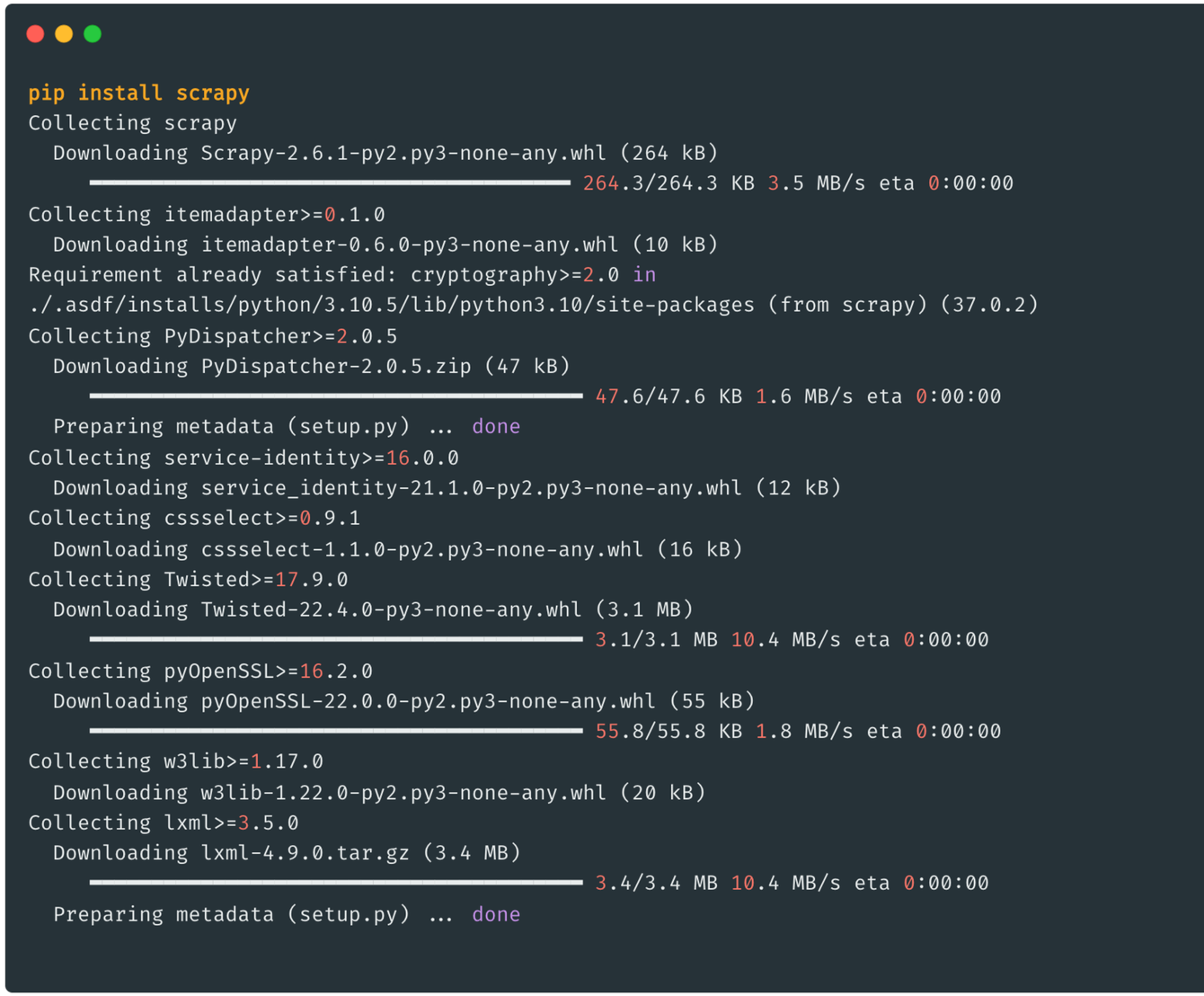
`asdf current`



Shim?

Rescan dependencies

```
asdf reshim <name> <vers>
```



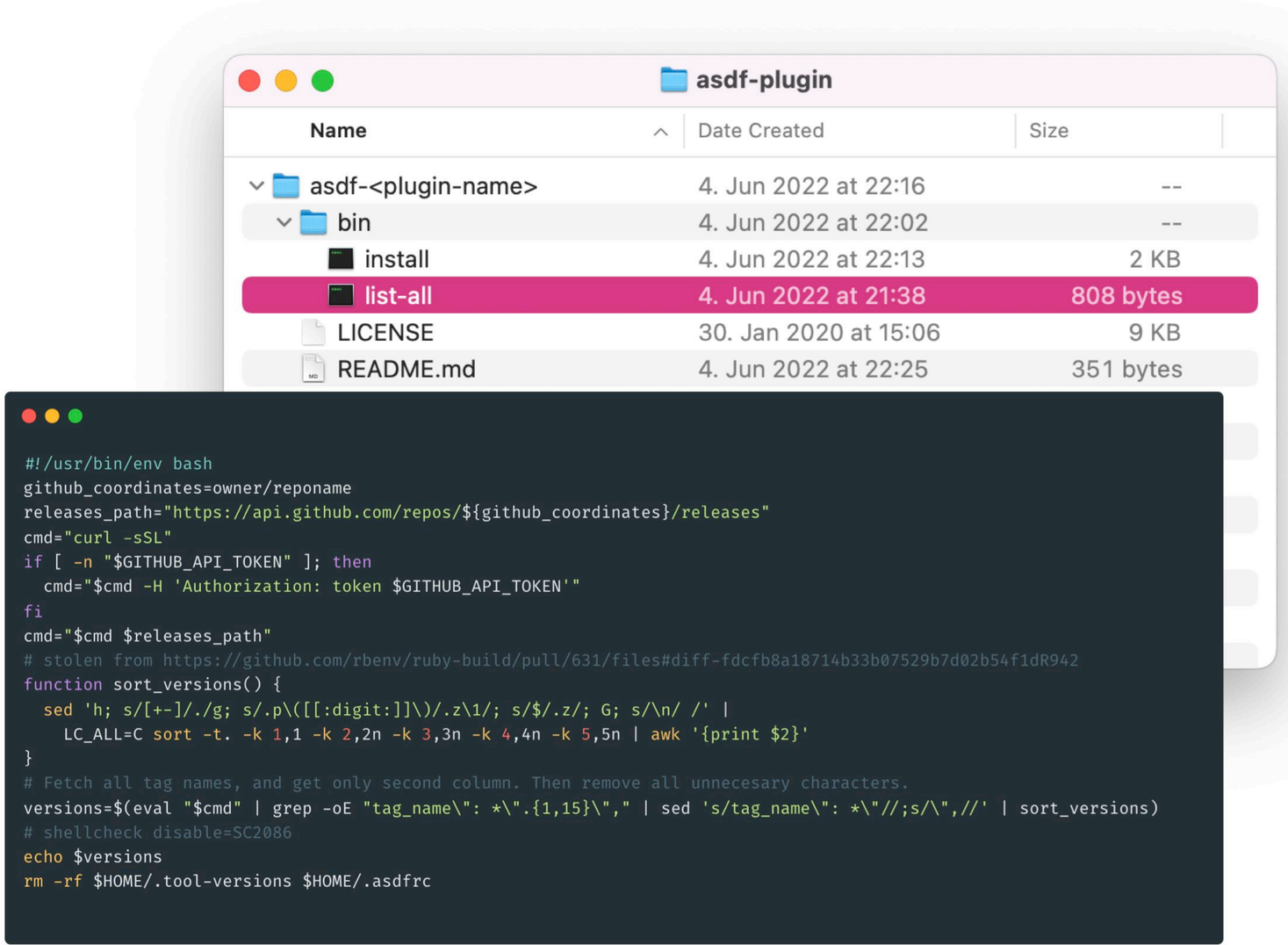
Plugin ecosystem

Easy to create own plugins

At least two shell scripts:

→ list-all

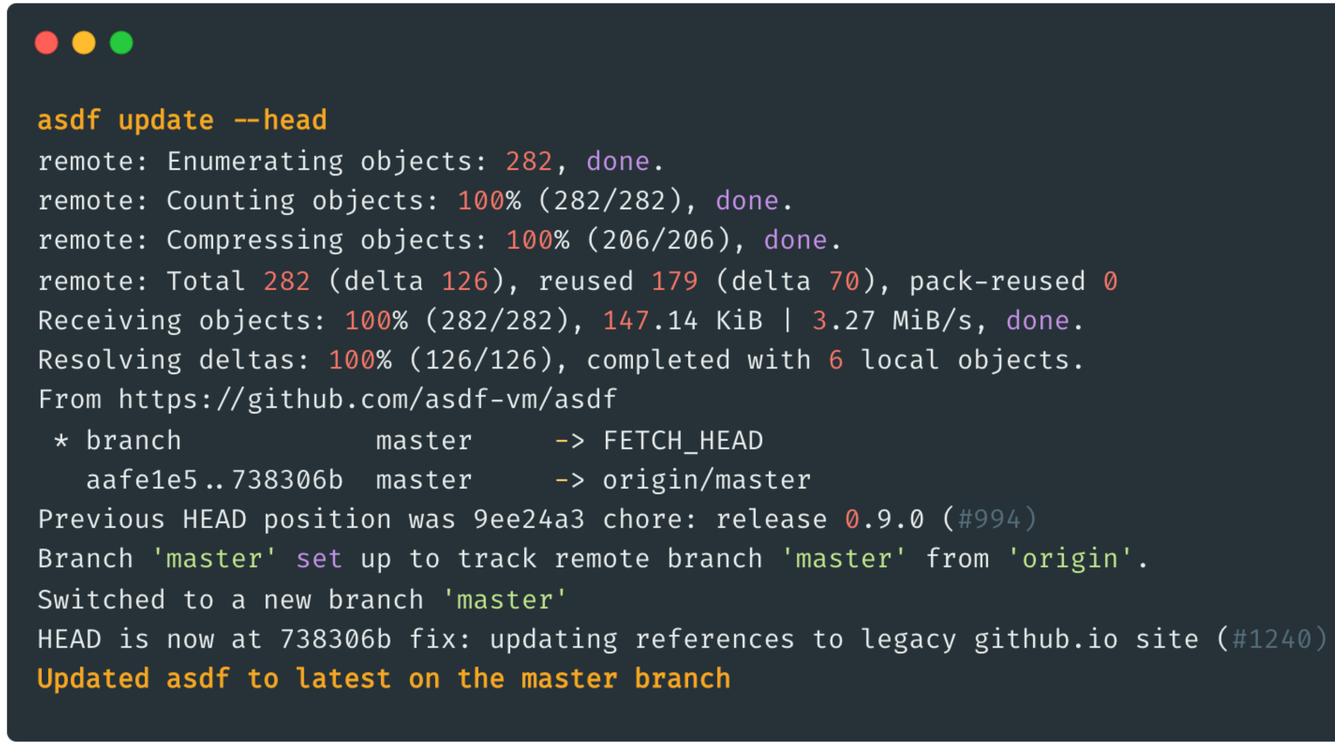
→ install



Update asdf

built in command

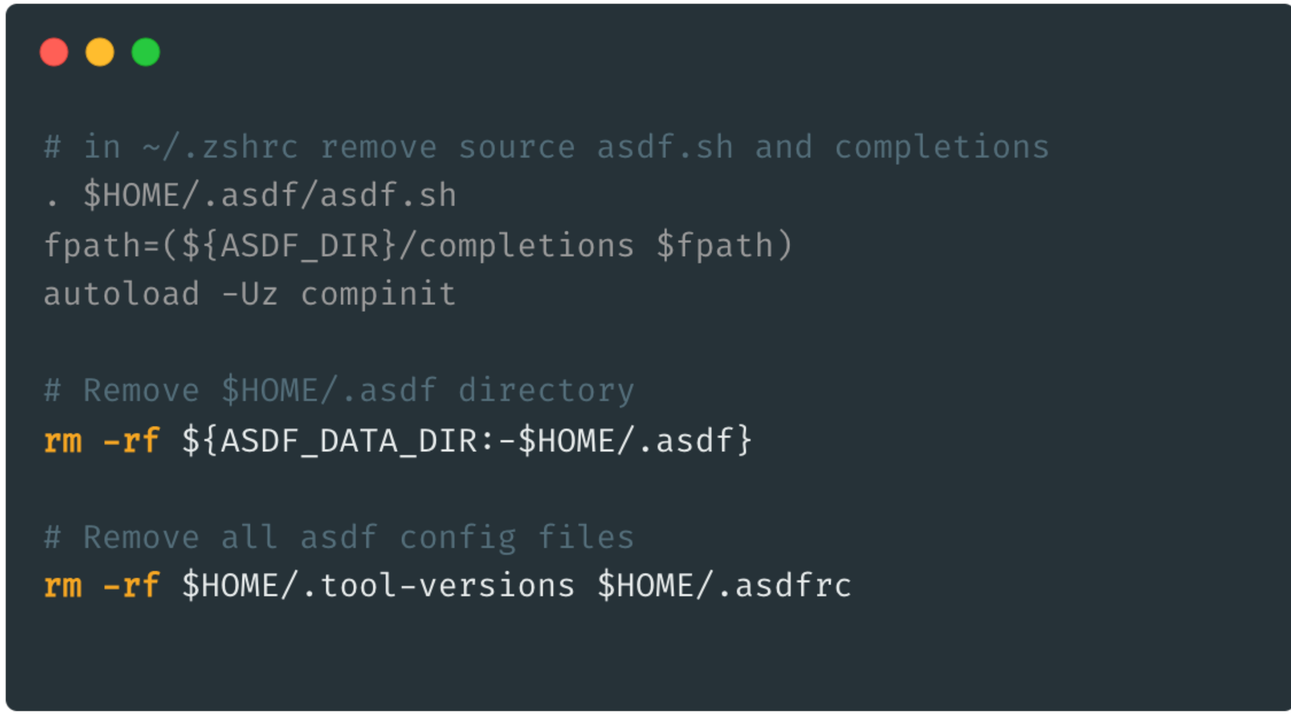
```
asdf update —head
```



Uninstall

Simple removal task

```
rm -rf $HOME/.asdf
rm -rf $HOME/.tool-versions
```



Asdf man page

Manage Plugins

```
asdf -h

MANAGE PLUGINS
asdf plugin add <name> [<git-url>]      Add a plugin from the plugin repo
asdf plugin list [--urls] [--refs]      List installed plugins.
asdf plugin list all                    List plugins registered on asdf-plugins
asdf plugin remove <name>               Remove plugin and package versions
asdf plugin update <name> [<git-ref>]   Update a plugin to latest commit on
                                        default branch or a particular git-ref
asdf plugin update --all                Update all plugins to latest commit on
```

Asdf man page

Manage Packages

```
asdf -h

MANAGE PACKAGES
asdf install                          Install all the package versions listed in the .tool-versions file
asdf install <name>                   Install one tool at the version specified in the .tool-versions file
asdf install <name> <version>         Install a specific version of a package
asdf install <name> latest[:<version>] Install the latest stable version of a package
asdf uninstall <name> <version>       Remove a specific version of a package
asdf current                          Display current version set or being used for all packages
asdf current <name>                   Display current version set or being used for package
asdf where <name> [<version>]         Display install path for an installed or current version
asdf which <command>                  Display the path to an executable
asdf local <name> <version>           Set the package local version
asdf local <name> latest[:<version>]  Set the package local version to the latest provided version
asdf global <name> <version>          Set the package global version
asdf global <name> latest[:<version>] Set the package global version to the latest provided version
asdf shell <name> <version>           Set the package version to `ASDF_${LANG}_VERSION` in the current shell
asdf latest <name> [<version>]        Show latest stable version of a package
asdf latest --all                     Show latest stable version of all the packages and if they are installed
asdf list <name> [version]            List installed versions of a package and optionally filter the versions
asdf list all <name> [<version>]      List all versions of a package and optionally filter the returned version
asdf help <name> [<version>]          Output documentation for plugin and tool
```

Asdf man page

Manage Utils

```
asdf -h

UTILS
asdf exec <command> [args ... ]          Executes the command shim for current version
asdf env <command> [util]                Runs util (default: `env`) inside the environment used for command shim
execution.
asdf info                                Print OS, Shell and ASDF debug information.
asdf reshim <name> <version>             Recreate shims for version of a package
asdf shim-versions <command>             List the plugins and versions that provide a command
asdf update                              Update asdf to the latest stable release
asdf update --head                       Update asdf to the latest on the master branch
```

asdf - brief summary

A simple tool version manager

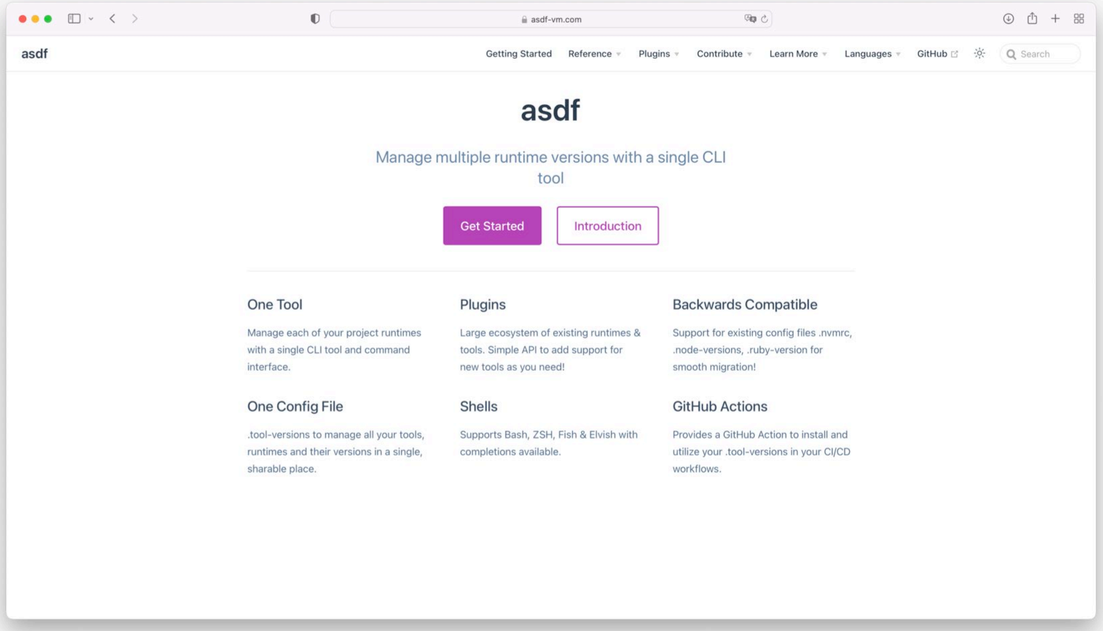
Super simple approach

Linux, macOS support

User-specific only

Plugin ecosystem

Global/local .tool-versions file



Install and switch runtime versions as you like!

Caveats?

Sufficient safeguarding

Supply chain control/hygene

Pin to commits in source repository (Plugins, etc.)

Auditing $HOME/.asdf/ directory

→ File integrity monitoring (Osquery)

→ Collect hashes, SPDX (SBOMs)

Links

Learn more

Project: https://asdf-vm.com

Plugins repo: https://github.com/asdf-vm/asdf-plugins

GitHub Actions: https://github.com/asdf-vm/actions

→ MDO:YVR: https://blog.macadmin.me/posts/asdf-on-macos/

→ macOS: https://www.wiserfirst.com/blog/how-to-use-asdf-on-macos/

→ Linux/ WSL: https://www.joshfinnie.com/blog/setting_up_wsl_with_asdf/

Thank you

Slack: @headmin(henry)
Twitter: @head_min


Zentral: https://zentral.io