pipenv Documentation

Release 2022.1.9.dev0

Kenneth Reitz

Jan 10, 2022

Contents

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*.

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

- You no longer need to use `pip` and `virtualenv` separately. They work together.

- Managing a `requirements.txt` file can be problematic, so Pipenv uses `Pipfile` and `Pipfile.lock` to separate abstract dependency declarations from the last tested combination.

- Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.

- Strongly encourage the use of the latest versions of dependencies to minimize security risks arising from outdated components.

- Give you insight into your dependency graph (e.g. `$ pipenv graph`).

- Streamline development workflow by loading `.env` files.

You can quickly play with Pipenv right in your browser:

Install Pipenv Today!

If you already have Python and pip, you can easily install Pipenv into your home directory:

```
$ pip install --user pipenv
```

Or, if you're using Fedora 28:

```
$ sudo dnf install pipenv
```

It's possible to install Pipenv with Homebrew on MacOS, or with Linuxbrew on Linux systems. However, **this is now discouraged**, because updates to the brewed Python distribution will break Pipenv, and perhaps all virtual environments managed by it. You'll then need to re-install Pipenv at least. If you want to give it a try despite this warning, use:

```
$ brew install pipenv
```

More detailed installation instructions can be found in the *Installing Pipenv* chapter.

1.1 Pipenv & Virtual Environments

This tutorial walks you through installing and using Python packages.

It will show you how to install and use the necessary tools and make strong recommendations on best practices. Keep in mind that Python is used for a great many different purposes, and precisely how you want to manage your dependencies may change based on how you decide to publish your software. The guidance presented here is most directly applicable to the development and deployment of network services (including web applications), but is also very well suited to managing development and testing environments for any kind of project.

Note: This guide is written for Python 3, however, these instructions should work fine on Python 2.7—if you are still using it, for some reason.

1.1.1 Make sure you've got Python & pip

Before you go any further, make sure you have Python and that it's available from your command line. You can check this by simply running:

```
$ python --version
```

You should get some output like `3.6.2`. If you do not have Python, please install the latest 3.x version from python.org or refer to the Installing Python section of *The Hitchhiker's Guide to Python*.

Note: If you're newcomer and you get an error like this:

```
>>> python
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'python' is not defined
```

It's because this command is intended to be run in a *shell* (also called a *terminal* or *console*). See the Python for Beginners getting started tutorial for an introduction to using your operating system's shell and interacting with Python.

Additionally, you'll need to make sure you have pip available. You can check this by running:

```
$ pip --version
pip 9.0.1
```

If you installed Python from source, with an installer from python.org, via Homebrew or via Linuxbrew you should already have pip. If you're on Linux and installed using your OS package manager, you may have to install pip separately.

If you plan to install Pipenv using Homebrew or Linuxbrew you can skip this step. The Homebrew/Linuxbrew installer takes care of pip for you.

1.1.2 Installing Pipenv

Pipenv is a dependency manager for Python projects. If you're familiar with Node.js's npm or Ruby's bundler, it is similar in spirit to those tools. While pip can install Python packages, Pipenv is recommended as it's a higher-level tool that simplifies dependency management for common use cases.

Isolated Installation of Pipenv with Pipx

Pipx is a tool to help you install and run end-user applications written in Python. It installs applications into an isolated and clean environment on their own. To install pipx, just run:

```
$ pip install --user pipx
```

Once you have `pipx` ready on your system, continue to install Pipenv:

```
$ pipx install pipenv
```

Pragmatic Installation of Pipenv

If you have a working installation of pip, and maintain certain "tool-chain" type Python modules as global utilities in your user environment, pip user installs allow for installation into your home directory. Note that due to interaction between dependencies, you should limit tools installed in this way to basic building blocks for a Python workflow like virtualenv, pipenv, tox, and similar software.

To install:

```
$ pip install --user pipenv
```

Note: This does a user installation to prevent breaking any system-wide packages. If `pipenv` isn't available in your shell after installation, you'll need to add the user base's binary directory to your `PATH`.

On Linux and macOS you can find the user base binary directory by running `python -m site --user-base` and adding `bin` to the end. For example, this will typically print `~/.local` (with ~ expanded to the absolute path to your home directory) so you'll need to add `~/.local/bin` to your `PATH`. You can set your `PATH` permanently by modifying ~/.profile.

On Windows you can find the user base binary directory by running `python -m site --user-site` and replacing `site-packages` with `Scripts`. For example, this could return `C:\Users\Username\AppData\Roaming\Python36\site-packages` so you would need to set your `PATH` to include `C:\Users\Username\AppData\Roaming\Python36\Scripts`. You can set your user `PATH` permanently in the Control Panel. You may need to log out for the `PATH` changes to take effect.

For more information, see the user installs documentation.

To upgrade pipenv at any time:

```
$ pip install --user --upgrade pipenv
```

Crude Installation of Pipenv

If you don't even have pip installed, you can use this crude installation method, which will bootstrap your whole system:

```
$ curl https://raw.githubusercontent.com/pypa/pipenv/master/get-pipenv.py | python
```

Homebrew Installation of Pipenv(Discouraged)

Homebrew is a popular open-source package management system for macOS. For Linux users, Linuxbrew is a Linux port of that.

Installing pipenv via Homebrew or Linuxbrew will keep pipenv and all of its dependencies in an isolated virtual environment so it doesn't interfere with the rest of your Python installation.

Once you have installed Homebrew or Linuxbrew simply run:

```
$ brew install pipenv
```

To upgrade pipenv at any time:

```
$ brew upgrade pipenv
```

Note: Homebrew installation is discouraged because each time the Homebrew Python is upgraded, which Pipenv depends on, users have to re-install Pipenv, and perhaps all virtual environments managed by it.

1.1.3 Installing packages for your project

Pipenv manages dependencies on a per-project basis. To install packages, change into your project's directory (or just an empty directory for this tutorial) and run:

```
$ cd myproject
$ pipenv install requests
```

Pipenv will install the excellent Requests library and create a `Pipfile` for you in your project's directory. The `Pipfile` is used to track which dependencies your project needs in case you need to re-install them, such as when you share your project with others. You should get output similar to this (although the exact paths shown will vary):

```
Creating a Pipfile for this project...
Creating a virtualenv for this project...
Using base prefix '/usr/local/Cellar/python3/3.6.2/Frameworks/Python.framework/
→Versions/3.6'
New python executable in ~/.local/share/virtualenvs/tmp-agwWamBd/bin/python3.6
Also creating executable in ~/.local/share/virtualenvs/tmp-agwWamBd/bin/python
Installing setuptools, pip, wheel...done.

Virtualenv location: ~/.local/share/virtualenvs/tmp-agwWamBd
Installing requests...
Collecting requests
  Using cached requests-2.18.4-py2.py3-none-any.whl
Collecting idna<2.7,>=2.5 (from requests)
  Using cached idna-2.6-py2.py3-none-any.whl
Collecting urllib3<1.23,>=1.21.1 (from requests)
  Using cached urllib3-1.22-py2.py3-none-any.whl
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Using cached chardet-3.0.4-py2.py3-none-any.whl
Collecting certifi>=2017.4.17 (from requests)
  Using cached certifi-2017.7.27.1-py2.py3-none-any.whl
Installing collected packages: idna, urllib3, chardet, certifi, requests
Successfully installed certifi-2017.7.27.1 chardet-3.0.4 idna-2.6 requests-2.18.4␣
→urllib3-1.22

Adding requests to Pipfile's [packages]...
P.S. You have excellent taste!
```

1.1.4 Using installed packages

Now that Requests is installed you can create a simple `main.py` file to use it:

```python
import requests

response = requests.get('https://httpbin.org/ip')

print('Your IP is {0}'.format(response.json()['origin']))
```

Then you can run this script using `pipenv run`:

```
$ pipenv run python main.py
```

You should get output similar to this:

```
Your IP is 8.8.8.8
```

Using `$ pipenv run` ensures that your installed packages are available to your script. It's also possible to spawn a new shell that ensures all commands have access to your installed packages with `$ pipenv shell`.

1.1.5 Virtualenv mapping caveat

- Pipenv automatically maps projects to their specific virtualenvs.

- The virtualenv is stored globally with the name of the project's root directory plus the hash of the full path to the project's root (e.g., `my_project-a3de50`).

- If you change your project's path, you break such a default mapping and pipenv will no longer be able to find and to use the project's virtualenv.

- You might want to set `export PIPENV_VENV_IN_PROJECT=1` in your .bashrc/.zshrc (or any shell configuration file) for creating the virtualenv inside your project's directory, avoiding problems with subsequent path changes.

1.1.6 Next steps

Congratulations, you now know how to install and use Python packages!

1.2 Release and Version History

1.2.1 2022.1.8 (2022-01-08)

Bug Fixes

- Remove the extra parentheses around the venv prompt. #4877

- Fix a bug of installation fails when extra index url is given. #4881

- Fix regression where lockfiles would only include the hashes for releases for the platform generating the lockfile #4885

- Fix the index parsing to reject illegal requirements.txt. #4899

1.2.2 2021.11.23 (2021-11-23)

Bug Fixes

- Update `charset-normalizer` from `2.0.3` to `2.0.7`, this fixes an import error on Python 3.6. #4865

- Fix a bug of deleting a virtualenv that is not managed by Pipenv. #4867

- Fix a bug that source is not added to `Pipfile` when index url is given with `pipenv install`. #4873

1.2.3 2021.11.15 (2021-11-15)

Bug Fixes

- Return an empty dict when `PIPENV_DONT_LOAD_ENV` is set. #4851

- Don't use `sys.executable` when inside an activated venv. #4852

Vendored Libraries

- Drop the vendored `jinja2` dependency as it is not needed any more. #4858
- Update `click` from `8.0.1` to `8.0.3`, to fix a problem with bash completion. #4860
- Drop unused vendor `chardet`. #4862

Improved Documentation

- Fix the documentation to reflect the fact that special characters must be percent-encoded in the URL. #4856

1.2.4 2021.11.9 (2021-11-09)

Features & Improvements

- Replace `click-completion` with `click`'s own completion implementation. #4786

Bug Fixes

- Fix a bug that `pipenv run` doesn't set environment variables correctly. #4831
- Fix a bug that certifi can't be loaded within `notpip`'s vendor library. This makes several objects of `pip` fail to be imported. #4833
- Fix a bug that `3.10.0` can be found be python finder. #4837

Vendored Libraries

- Update `pythonfinder` from `1.2.8` to `1.2.9`. #4837

1.2.5 2021.11.5.post0 (2021-11-05)

Bug Fixes

- Fix a regression that `pipenv shell` fails to start a subshell. #4828
- Fix a regression that `pip_shims` object isn't imported correctly. #4829

1.2.6 2021.11.5 (2021-11-05)

Features & Improvements

- Avoid sharing states but create project objects on demand. So that most integration test cases are able to switch to a in-process execution method. #4757
- Shell-quote `pip` commands when logging. #4760

Bug Fixes

- Ignore empty .venv in rood dir and create project name base virtual environment #4790

Vendored Libraries

- Update vendored dependencies - `attrs` from `20.3.0` to `21.2.0` - `cerberus` from `1.3.2` to `1.3.4` - `certifi` from `2020.11.8` to `2021.5.30` - `chardet` from `3.0.4` to `4.0.0` - `click` from `7.1.2` to `8.0.1` - `distlib` from `0.3.1` to `0.3.2` - `idna` from `2.10` to `3.2` - `importlib-metadata` from `2.0.0` to `4.6.1` - `importlib-resources` from `3.3.0` to `5.2.0` - `jinja2` from `2.11.2` to `3.0.1` - `markupsafe` from `1.1.1` to `2.0.1` - `more-itertools` from `5.0.0` to `8.8.0` - `packaging` from `20.8` to `21.0` - `pep517` from `0.9.1` to `0.11.0` - `pipdeptree` from `1.0.0` to `2.0.0` - `ptyprocess` from `0.6.0` to `0.7.0` - `python-dateutil` from `2.8.1` to `2.8.2` - `python-dotenv` from `0.15.0` to `0.19.0` - `pythonfinder` from `1.2.5` to `1.2.8` - `requests` from `2.25.0` to `2.26.0` - `shellingham` from `1.3.2` to `1.4.0` - `six` from `1.15.0` to `1.16.0` - `tomlkit` from `0.7.0` to `0.7.2` - `urllib3` from `1.26.1` to `1.26.6` - `zipp` from `1.2.0` to `3.5.0`

  Add new vendored dependencies - `charset-normalizer 2.0.3` - `termcolor 1.1.0` - `tomli 1.1.0` - `wheel 0.36.2` #4747

- Drop the dependencies for Python 2.7 compatibility purpose. #4751

- Switch the dependency resolver from `pip-tools` to *pip*.

  Update vendor libraries: - Update `requirementslib` from `1.5.16` to `1.6.1` - Update `pip-shims` from `0.5.6` to `0.6.0` - New vendor `platformdirs 2.4.0` #4759

Improved Documentation

- remove prefixes on install commands for easy copy/pasting #4792

- Officially drop support for Python 2.7 and Python 3.5. #4261

1.2.7 2021.5.29 (2021-05-29)

Bug Fixes

- Fix a bug where passing –skip-lock when PIPFILE has no [SOURCE] section throws the error: "tomlkit.exceptions.NonExistentKey: 'Key "source" does not exist.'" #4141

- Fix bug where environment wouldn't activate in paths containing & and $ symbols #4538

- Fix a bug that `importlib-metadata` from the project's dependencies conflicts with that from `pipenv`'s. #4549

- Fix a bug where `pep508checker.py` did not expect double-digit Python minor versions (e.g. "3.10"). #4602

- Fix bug where environment wouldn't activate in paths containing () and [] symbols #4615

- Fix bug preventing use of pipenv lock –pre #4642

Vendored Libraries

- Update `packaging` from `20.4` to `20.8`. #4591

1.2.8 2020.11.15 (2020-11-15)

Features & Improvements

- Support expanding environment variables in requirement URLs. #3516

- Show warning message when a dependency is skipped in locking due to the mismatch of its markers. #4346

Bug Fixes

- Fix a bug that executable scripts with leading backslash can't be executed via `pipenv run`. #4368
- Fix a bug that VCS dependencies always satisfy even if the ref has changed. #4387
- Restrict the acceptable hash type to SHA256 only. #4517
- Fix the output of `pipenv scripts` under Windows platform. #4523
- Fix a bug that the resolver takes wrong section to validate constraints. #4527

Vendored Libraries

- **Update vendored dependencies:**
    - `colorama` from `0.4.3` to `0.4.4`
    - `python-dotenv` from `0.10.3` to `0.15.0`
    - `first` from `2.0.1` to `2.0.2`
    - `iso8601` from `0.1.12` to `0.1.13`
    - `parse` from `1.15.0` to `1.18.0`
    - `pipdeptree` from `0.13.2` to `1.0.0`
    - `requests` from `2.23.0` to `2.25.0`
    - `idna` from `2.9` to `2.10`
    - `urllib3` from `1.25.9` to `1.26.1`
    - `certifi` from `2020.4.5.1` to `2020.11.8`
    - `requirementslib` from `1.5.15` to `1.5.16`
    - `attrs` from `19.3.0` to `20.3.0`
    - `distlib` from `0.3.0` to `0.3.1`
    - `packaging` from `20.3` to `20.4`
    - `six` from `1.14.0` to `1.15.0`
    - `semver` from `2.9.0` to `2.13.0`
    - `toml` from `0.10.1` to `0.10.2`
    - `cached-property` from `1.5.1` to `1.5.2`
    - `yaspin` from `0.14.3` to `1.2.0`
    - `resolvelib` from `0.3.0` to `0.5.2`
    - `pep517` from `0.8.2` to `0.9.1`
    - `zipp` from `0.6.0` to `1.2.0`
    - `importlib-metadata` from `1.6.0` to `2.0.0`
    - `importlib-resources` from `1.5.0` to `3.3.0` #4533

Improved Documentation

- Fix suggested pyenv setup to avoid using shimmed interpreter [#4534](#)

1.2.9 2020.11.4 (2020-11-04)

Features & Improvements

- Add a new command `pipenv scripts` to display shortcuts from Pipfile. [#3686](#)

- Retrieve package file hash from URL to accelerate the locking process. [#3827](#)

- Add the missing `--system` option to `pipenv sync`. [#4441](#)

- Add a new option pair `--header/--no-header` to `pipenv lock` command, which adds a header to the generated requirements.txt [#4443](#)

Bug Fixes

- Fix a bug that percent encoded characters will be unquoted incorrectly in the file URL. [#4089](#)

- Fix a bug where setting PIPENV_PYTHON to file path breaks environment name [#4225](#)

- Fix a bug that paths are not normalized before comparison. [#4330](#)

- Handle Python major and minor versions correctly in Pipfile creation. [#4379](#)

- Fix a bug that non-wheel file requirements can be resolved successfully. [#4386](#)

- Fix a bug that `pexept.exceptions.TIMEOUT` is not caught correctly because of the wrong import path. [#4424](#)

- Fix a bug that compound TOML table is not parsed correctly. [#4433](#)

- Fix a bug that invalid Python paths from Windows registry break `pipenv install`. [#4436](#)

- Fix a bug that function calls in `setup.py` can't be parsed rightly. [#4446](#)

- Fix a bug that dist-info inside `venv` directory will be mistaken as the editable package's metadata. [#4480](#)

- Make the order of hashes in resolution result stable. [#4513](#)

Vendored Libraries

- Update `tomlkit` from `0.5.11` to `0.7.0`. [#4433](#)

- Update `requirementslib` from `1.5.13` to `1.5.14`. [#4480](#)

Improved Documentation

- Discourage homebrew installation in installation guides. [#4013](#)

1.2.10 2020.8.13 (2020-08-13)

Bug Fixes

- Fixed behaviour of `pipenv uninstall --all-dev`. From now on it does not uninstall regular packages.
  #3722

- Fix a bug that incorrect Python path will be used when `--system` flag is on. #4315

- Fix falsely flagging a Homebrew installed Python as a virtual environment #4316

- Fix a bug that `pipenv uninstall` throws an exception that does not exist. #4321

- Fix a bug that Pipenv can't locate the correct file of special directives in `setup.cfg` of an editable package.
  #4335

- Fix a bug that `setup.py` can't be parsed correctly when the assignment is type-annotated. #4342

- Fix a bug that `pipenv graph` throws an exception that PipenvCmdError(cmd_string, c.out, c.err, return_code). #4388

- Do not copy the whole directory tree of local file package. #4403

- Correctly detect whether Pipenv in run under an activated virtualenv. #4412

Vendored Libraries

- Update `requirementslib` to `1.5.12`. #4385

-   - Update `requirements` to `1.5.13`.

    - Update `pip-shims` to `0.5.3`. #4421

1.2.11 2020.6.2 (2020-06-02)

Features & Improvements

- Pipenv will now detect existing `venv` and `virtualenv` based virtual environments more robustly. #4276

Bug Fixes

- + signs in URL authentication fragments will no longer be incorrectly replaced with space ('' '') characters.
  #4271

- Fixed a regression which caused Pipenv to fail when running under `/`. #4273

- `setup.py` files with `version` variables read from `os.environ` are now able to be parsed successfully.
  #4274

- Fixed a bug which caused Pipenv to fail to install packages in a virtual environment if those packages were
  already present in the system global environment. #4276

- Fix a bug that caused non-specific versions to be pinned in `Pipfile.lock`. #4278

- Corrected a missing exception import and invalid function call invocations in `pipenv.cli.command`. #4286

- Fixed an issue with resolving packages with names defined by function calls in `setup.py`. #4292

- Fixed a regression with installing the current directory, or `.`, inside a `venv` based virtual environment. #4295

- Fixed a bug with the discovery of python paths on Windows which could prevent installation of environments during `pipenv install`. #4296

- Fixed an issue in the `requirementslib` AST parser which prevented parsing of `setup.py` files for dependency metadata. #4298

- Fix a bug where Pipenv doesn't realize the session is interactive #4305

Vendored Libraries

- Updated requirementslib to version `1.5.11`. #4292

- **Updated vendored dependencies:**

    - **pythonfinder**: `1.2.2 => 1.2.4`

    - **requirementslib**: `1.5.9 => 1.5.10` #4302

1.2.12 2020.5.28 (2020-05-28)

Features & Improvements

- `pipenv install` and `pipenv sync` will no longer attempt to install satisfied dependencies during installation. #3057, #3506

- Added support for resolution of direct-url dependencies in `setup.py` files to respect `PEP-508` style URL dependencies. #3148

- Added full support for resolution of all dependency types including direct URLs, zip archives, tarballs, etc.

    - Improved error handling and formatting.

    - Introduced improved cross platform stream wrappers for better `stdout` and `stderr` consistency. #3298

- For consistency with other commands and the `--dev` option description, `pipenv lock --requirements --dev` now emits both default and development dependencies. The new `--dev-only` option requests the previous behaviour (e.g. to generate a `dev-requirements.txt` file). #3316

- Pipenv will now successfully recursively lock VCS sub-dependencies. #3328

- Added support for `--verbose` output to `pipenv run`. #3348

- Pipenv will now discover and resolve the intrinsic dependencies of **all** VCS dependencies, whether they are editable or not, to prevent resolution conflicts. #3368

- Added a new environment variable, `PIPENV_RESOLVE_VCS`, to toggle dependency resolution off for non-editable VCS, file, and URL based dependencies. #3577

- Added the ability for Windows users to enable emojis by setting `PIPENV_HIDE_EMOJIS=0`. #3595

- Allow overriding PIPENV_INSTALL_TIMEOUT environment variable (in seconds). #3652

- Allow overriding PIP_EXISTS_ACTION evironment variable (value is passed to pip install). Possible values here: https://pip.pypa.io/en/stable/reference/pip/#exists-action-option Useful when you need to *PIP_EXISTS_ACTION=i* (ignore existing packages) - great for CI environments, where you need really fast setup. #3738

- Pipenv will no longer forcibly override `PIP_NO_DEPS` on all vcs and file dependencies as resolution happens on these in a pre-lock step. #3763

- Improved verbose logging output during `pipenv lock` will now stream output to the console while maintaining a spinner. #3810

- Added support for automatic python installs via `asdf` and associated `PIPENV_DONT_USE_ASDF` environment variable. #4018

- Pyenv/asdf can now be used whether or not they are available on PATH. Setting PYENV_ROOT/ASDF_DIR in a Pipenv's .env allows Pipenv to install an interpreter without any shell customizations, so long as pyenv/asdf is installed. #4245

- Added `--key` command line parameter for including personal PyUp.io API tokens when running `pipenv check`. #4257

Behavior Changes

- Make conservative checks of known exceptions when subprocess returns output, so user won't see the whole traceback - just the error. #2553

- Do not touch Pipfile early and rely on it so that one can do `pipenv sync` without a Pipfile. #3386

- Re-enable `--help` option for `pipenv run` command. #3844

- Make sure `pipenv lock -r --pypi-mirror {MIRROR_URL}` will respect the pypi-mirror in requirements output. #4199

Bug Fixes

- Raise *PipenvUsageError* when [[source]] does not contain url field. #2373

- Fixed a bug which caused editable package resolution to sometimes fail with an unhelpful setuptools-related error message. #2722

- Fixed an issue which caused errors due to reliance on the system utilities `which` and `where` which may not always exist on some systems. - Fixed a bug which caused periodic failures in python discovery when executables named `python` were not present on the target `$PATH`. #2783

- Dependency resolution now writes hashes for local and remote files to the lockfile. #3053

- Fixed a bug which prevented `pipenv graph` from correctly showing all dependencies when running from within `pipenv shell`. #3071

- Fixed resolution of direct-url dependencies in `setup.py` files to respect `PEP-508` style URL dependencies. #3148

- Fixed a bug which caused failures in warning reporting when running pipenv inside a virtualenv under some circumstances.

  - Fixed a bug with package discovery when running `pipenv clean`. #3298

- Quote command arguments with carets (`^`) on Windows to work around unintended shell escapes. #3307

- Handle alternate names for UTF-8 encoding. #3313

- Abort pipenv before adding the non-exist package to Pipfile. #3318

- Don't normalize the package name user passes in. #3324

- Fix a bug where custom virtualenv can not be activated with pipenv shell #3339

- Fix a bug that `--site-packages` flag is not recognized. #3351

- Fix a bug where pipenv –clear is not working #3353

- Fix unhashable type error during `$ pipenv install --selective-upgrade` #3384

- Dependencies with direct `PEP508` compliant VCS URLs specified in their `install_requires` will now be successfully locked during the resolution process. #3396

- Fixed a keyerror which could occur when locking VCS dependencies in some cases. #3404

- Fixed a bug that `ValidationError` is thrown when some fields are missing in source section. #3427

- Updated the index names in lock file when source name in Pipfile is changed. #3449

- Fixed an issue which caused `pipenv install --help` to show duplicate entries for `--pre`. #3479

- Fix bug causing `[SSL: CERTIFICATE_VERIFY_FAILED]` when Pipfile `[[source]]` has verify_ssl=false and url with custom port. #3502

- Fix `sync --sequential` ignoring `pip install` errors and logs. #3537

- Fix the issue that lock file can't be created when `PIPENV_PIPFILE` is not under working directory. #3584

- Pipenv will no longer inadvertently set `editable=True` on all vcs dependencies. #3647

- The `--keep-outdated` argument to `pipenv install` and `pipenv lock` will now drop specifier constraints when encountering editable dependencies. - In addition, `--keep-outdated` will retain specifiers that would otherwise be dropped from any entries that have not been updated. #3656

- Fixed a bug which sometimes caused pipenv to fail to respect the `--site-packages` flag when passed with `pipenv install`. #3718

- Normalize the package names to lowercase when comparing used and in-Pipfile packages. #3745

- `pipenv update --outdated` will now correctly handle comparisons between pre/post-releases and normal releases. #3766

- Fixed a `KeyError` which could occur when pinning outdated VCS dependencies via `pipenv lock --keep-outdated`. #3768

- Resolved an issue which caused resolution to fail when encountering poorly formatted `python_version` markers in `setup.py` and `setup.cfg` files. #3786

- Fix a bug that installation errors are displayed as a list. #3794

- Update `pythonfinder` to fix a problem that `python.exe` will be mistakenly chosen for virtualenv creation under WSL. #3807

- Fixed several bugs which could prevent editable VCS dependencies from being installed into target environments, even when reporting successful installation. #3809

- `pipenv check --system` should find the correct Python interpreter when `python` does not exist on the system. #3819

- Resolve the symlinks when the path is absolute. #3842

- Pass `--pre` and `--clear` options to `pipenv update --outdated`. #3879

- Fixed a bug which prevented resolution of direct URL dependencies which have PEP508 style direct url VCS sub-dependencies with subdirectories. #3976

- Honor PIPENV_SPINNER environment variable #4045

- Fixed an issue with `pipenv check` failing due to an invalid API key from `pyup.io`. #4188

- Fixed a bug which caused versions from VCS dependencies to be included in `Pipfile.lock` inadvertently. #4217

- Fixed a bug which caused pipenv to search non-existent virtual environments for `pip` when installing using `--system`. #4220

- `Requires-Python` values specifying constraint versions of python starting from `1.x` will now be parsed successfully. #4226

- Fix a bug of `pipenv update --outdated` that can't print output correctly. #4229

- Fixed a bug which caused pipenv to prefer source distributions over wheels from `PyPI` during the dependency resolution phase. Fixed an issue which prevented proper build isolation using `pep517` based builders during dependency resolution. #4231

- Don't fallback to system Python when no matching Python version is found. #4232

Vendored Libraries

- Updated vendored dependencies:
    - **attrs**: `18.2.0` => `19.1.0`
    - **certifi**: `2018.10.15` => `2019.3.9`
    - **cached_property**: `1.4.3` => `1.5.1`
    - **cerberus**: `1.2.0` => `1.3.1`
    - **click-completion**: `0.5.0` => `0.5.1`
    - **colorama**: `0.3.9` => `0.4.1`
    - **distlib**: `0.2.8` => `0.2.9`
    - **idna**: `2.7` => `2.8`
    - **jinja2**: `2.10.0` => `2.10.1`
    - **markupsafe**: `1.0` => `1.1.1`
    - **orderedmultidict**: `(new)` => `1.0`
    - **packaging**: `18.0` => `19.0`
    - **parse**: `1.9.0` => `1.12.0`
    - **pathlib2**: `2.3.2` => `2.3.3`
    - **pep517**: `(new)` => `0.5.0`
    - **pexpect**: `4.6.0` => `4.7.0`
    - **pipdeptree**: `0.13.0` => `0.13.2`
    - **pyparsing**: `2.2.2` => `2.3.1`
    - **python-dotenv**: `0.9.1` => `0.10.2`
    - **pythonfinder**: `1.1.10` => `1.2.1`
    - **pytoml**: `(new)` => `0.1.20`
    - **requests**: `2.20.1` => `2.21.0`
    - **requirementslib**: `1.3.3` => `1.5.0`
    - **scandir**: `1.9.0` => `1.10.0`
    - **shellingham**: `1.2.7` => `1.3.1`
    - **six**: `1.11.0` => `1.12.0`
    - **tomlkit**: `0.5.2` => `0.5.3`

- **urllib3**: `1.24` => `1.25.2`

- **vistir**: `0.3.0` => `0.4.1`

- **yaspin**: `0.14.0` => `0.14.3`

- Removed vendored dependency **cursor**. #3298

- Updated `pip_shims` to support `--outdated` with new pip versions. #3766

- Update vendored dependencies and invocations

  - Update vendored and patched dependencies - Update patches on `piptools`, `pip`, `pip-shims`, "tomlkit`

  - Fix invocations of dependencies - Fix custom "InstallCommand` instantiation - Update "PackageFinder` usage - Fix "Bool` stringify attempts from "tomlkit`

Updated vendored dependencies:

  - **attrs**: `18.2.0` => `19.1.0`

  - **certifi**: `2018.10.15` => `2019.3.9`

  - **cached_property**: `1.4.3` => `1.5.1`

  - **cerberus**: `1.2.0` => `1.3.1`

  - **click**: `7.0.0` => `7.1.1`

  - **click-completion**: `0.5.0` => `0.5.1`

  - **colorama**: `0.3.9` => `0.4.3`

  - **contextlib2**: ` (new)` => `0.6.0.post1`

  - **distlib**: `0.2.8` => `0.2.9`

  - **funcsigs**: ` (new)` => `1.0.2`

  - **importlib_metadata** `1.3.0` => `1.5.1`

  - **importlib-resources**: ` (new)` => `1.4.0`

  - **idna**: `2.7` => `2.9`

  - **jinja2**: `2.10.0` => `2.11.1`

  - **markupsafe**: `1.0` => `1.1.1`

  - **more-itertools**: ` (new)` => `5.0.0`

  - **orderedmultidict**: ` (new)` => `1.0`

  - **packaging**: `18.0` => `19.0`

  - **parse**: `1.9.0` => `1.15.0`

  - **pathlib2**: `2.3.2` => `2.3.3`

  - **pep517**: ` (new)` => `0.5.0`

  - **pexpect**: `4.6.0` => `4.8.0`

  - **pip-shims**: `0.2.0` => `0.5.1`

  - **pipdeptree**: `0.13.0` => `0.13.2`

  - **pyparsing**: `2.2.2` => `2.4.6`

  - **python-dotenv**: `0.9.1` => `0.10.2`

> - **pythonfinder**: `` `1.1.10 `` => `` `1.2.2 ``
>
> - **pytoml**: `` `(new) `` => `` `0.1.20 ``
>
> - **requests**: `` `2.20.1 `` => `` `2.23.0 ``
>
> - **requirementslib**: `` `1.3.3 `` => `` `1.5.4 ``
>
> - **scandir**: `` `1.9.0 `` => `` `1.10.0 ``
>
> - **shellingham**: `` `1.2.7 `` => `` `1.3.2 ``
>
> - **six**: `` `1.11.0 `` => `` `1.14.0 ``
>
> - **tomlkit**: `` `0.5.2 `` => `` `0.5.11 ``
>
> - **urllib3**: `` `1.24 `` => `` `1.25.8 ``
>
> - **vistir**: `` `0.3.0 `` => `` `0.5.0 ``
>
> - **yaspin**: `` `0.14.0 `` => `` `0.14.3 ``
>
> - **zipp**: `` `0.6.0 ``
>
> – Removed vendored dependency **cursor**. #4169

- Add and update vendored dependencies to accommodate `safety` vendoring: - **safety** `(none)` => `1.8.7` - **dparse** `(none)` => `0.5.0` - **pyyaml** `(none)` => `5.3.1` - **urllib3** `1.25.8` => `1.25.9` - **certifi** `2019.11.28` => `2020.4.5.1` - **pyparsing** `2.4.6` => `2.4.7` - **resolvelib** `0.2.2` => `0.3.0` - **importlib-metadata** `1.5.1` => `1.6.0` - **pip-shims** `0.5.1` => `0.5.2` - **requirementslib** `1.5.5` => `1.5.6` #4188

- Updated vendored `pip` => `20.0.2` and `pip-tools` => `5.0.0`. #4215

- Updated vendored dependencies to latest versions for security and bug fixes:

  - **requirementslib** `1.5.8` => `1.5.9`

  - **vistir** `0.5.0` => `0.5.1`

  - **jinja2** `2.11.1` => `2.11.2`

  - **click** `7.1.1` => `7.1.2`

  - **dateutil** `(none)` => `2.8.1`

  - **backports.functools_lru_cache** `1.5.0` => `1.6.1`

  - **enum34** `1.1.6` => `1.1.10`

  - **toml** `0.10.0` => `0.10.1`

  - **importlib_resources** `1.4.0` => `1.5.0` #4226

- Changed attrs import path in vendored dependencies to always import from `pipenv.vendor`. #4267

Improved Documentation

- Added documenation about variable expansion in `Pipfile` entries. #2317

- Consolidate all contributing docs in the rst file #3120

- Update the out-dated manual page. #3246

- Move CLI docs to its own page. #3346

- Replace (non-existant) video on docs index.rst with equivalent gif. #3499

- Clarify wording in Basic Usage example on using double quotes to escape shell redirection #3522

- Ensure docs show navigation on small-screen devices #3527
- Added a link to the TOML Spec under General Recommendations & Version Control to clarify how Pipfiles should be written. #3629
- Updated the documentation with the new `pytest` entrypoint. #3759
- Fix link to GIF in README.md demonstrating Pipenv's usage, and add descriptive alt text. #3911
- Added a line describing potential issues in fancy extension. #3912
- Documental description of how Pipfile works and association with Pipenv. #3913
- Clarify the proper value of `python_version` and `python_full_version`. #3914
- Write description for –deploy extension and few extensions differences. #3915
- More documentation for `.env` files #4100
- Updated documentation to point to working links. #4137
- Replace docs.pipenv.org with pipenv.pypa.io #4167
- Added functionality to check spelling in documentation and cleaned up existing typographical issues. #4209

1.2.13 2018.11.26 (2018-11-26)

Bug Fixes

- Environment variables are expanded correctly before running scripts on POSIX. #3178
- Pipenv will no longer disable user-mode installation when the `--system` flag is passed in. #3222
- Fixed an issue with attempting to render unicode output in non-unicode locales. #3223
- Fixed a bug which could cause failures to occur when parsing python entries from global pyenv version files. #3224
- Fixed an issue which prevented the parsing of named extras sections from certain `setup.py` files. #3230
- Correctly detect the virtualenv location inside an activated virtualenv. #3231
- Fixed a bug which caused spinner frames to be written to standard output during locking operations which could cause redirection pipes to fail. #3239
- Fixed a bug that editable packages can't be uninstalled correctly. #3240
- Corrected an issue with installation timeouts which caused dependency resolution to fail for longer duration resolution steps. #3244
- Adding normal pep 508 compatible markers is now fully functional when using VCS dependencies. #3249
- Updated `requirementslib` and `pythonfinder` for multiple bug fixes. #3254
- Pipenv will now ignore hashes when installing with `--skip-lock`. #3255
- Fixed an issue where pipenv could crash when multiple pipenv processes attempted to create the same directory. #3257
- Fixed an issue which sometimes prevented successful creation of a project Pipfile. #3260
- `pipenv install` will now unset the `PYTHONHOME` environment variable when not combined with `--system`. #3261
- Pipenv will ensure that warnings do not interfere with the resolution process by suppressing warnings' usage of standard output and writing to standard error instead. #3273

- Fixed an issue which prevented variables from the environment, such as `PIPENV_DEV` or `PIPENV_SYSTEM`, from being parsed and implemented correctly. #3278

- Clear pythonfinder cache after Python install. #3287

- Fixed a race condition in hash resolution for dependencies for certain dependencies with missing cache entries or fresh Pipenv installs. #3289

- Pipenv will now respect top-level pins over VCS dependency locks. #3296

Vendored Libraries

- **Update vendored dependencies to resolve resolution output parsing and python finding:**

    - `pythonfinder 1.1.9 -> 1.1.10`

    - `requirementslib 1.3.1 -> 1.3.3`

    - `vistir 0.2.3 -> 0.2.5` #3280

1.2.14 2018.11.14 (2018-11-14)

Features & Improvements

- Improved exceptions and error handling on failures. #1977

- Added persistent settings for all CLI flags via `PIPENV_{FLAG_NAME}` environment variables by enabling `auto_envvar_prefix=PIPENV` in click (implements PEEP-0002). #2200

- Added improved messaging about available but skipped updates due to dependency conflicts when running `pipenv update --outdated`. #2411

- Added environment variable `PIPENV_PYUP_API_KEY` to add ability to override the bundled PyUP.io API key. #2825

- Added additional output to `pipenv update --outdated` to indicate that the operation succeeded and all packages were already up to date. #2828

- Updated `crayons` patch to enable colors on native powershell but swap native blue for magenta. #3020

- Added support for `--bare` to `pipenv clean`, and fixed `pipenv sync --bare` to actually reduce output. #3041

- Added windows-compatible spinner via upgraded `vistir` dependency. #3089

-
    - Added support for python installations managed by `asdf`. #3096

- Improved runtime performance of no-op commands such as `pipenv --venv` by around 2/3. #3158

- Do not show error but success for running `pipenv uninstall --all` in a fresh virtual environment. #3170

- Improved asynchronous installation and error handling via queued subprocess parallelization. #3217

Bug Fixes

- Remote non-PyPI artifacts and local wheels and artifacts will now include their own hashes rather than including hashes from `PyPI`. #2394

- Non-ascii characters will now be handled correctly when parsed by pipenv's `ToML` parsers. #2737

- Updated `pipenv uninstall` to respect the `--skip-lock` argument. #2848

- Fixed a bug which caused uninstallation to sometimes fail to successfully remove packages from `Pipfiles` with comments on preceding or following lines. #2885, #3099

- Pipenv will no longer fail when encountering python versions on Windows that have been uninstalled. #2983

- Fixed unnecessary extras are added when translating markers #3026

- Fixed a virtualenv creation issue which could cause new virtualenvs to inadvertently attempt to read and write to global site packages. #3047

- Fixed an issue with virtualenv path derivation which could cause errors, particularly for users on WSL bash. #3055

- Fixed a bug which caused `Unexpected EOF` errors to be thrown when `pip` was waiting for input from users who had put login credentials in environment variables. #3088

- Fixed a bug in `requirementslib` which prevented successful installation from mercurial repositories. #3090

- Fixed random resource warnings when using pyenv or any other subprocess calls. #3094

-     - Fixed a bug which sometimes prevented cloning and parsing `mercurial` requirements. #3096

- Fixed an issue in `delegator.py` related to subprocess calls when using `PopenSpawn` to stream output, which sometimes threw unexpected `EOF` errors. #3102, #3114, #3117

- Fix the path casing issue that makes *pipenv clean* fail on Windows #3104

- Pipenv will avoid leaving build artifacts in the current working directory. #3106

- Fixed issues with broken subprocess calls leaking resource handles and causing random and sporadic failures. #3109

- Fixed an issue which caused `pipenv clean` to sometimes clean packages from the base `site-packages` folder or fail entirely. #3113

- Updated `pythonfinder` to correct an issue with unnesting of nested paths when searching for python versions. #3121

- Added additional logic for ignoring and replacing non-ascii characters when formatting console output on non-UTF-8 systems. #3131

- Fix virtual environment discovery when `PIPENV_VENV_IN_PROJECT` is set, but the in-project *.venv* is a file. #3134

- Hashes for remote and local non-PyPI artifacts will now be included in `Pipfile.lock` during resolution. #3145

- Fix project path hashing logic in purpose to prevent collisions of virtual environments. #3151

- Fix package installation when the virtual environment path contains parentheses. #3158

- Azure Pipelines YAML files are updated to use the latest syntax and product name. #3164

- Fixed new spinner success message to write only one success message during resolution. #3183

- Pipenv will now correctly respect the `--pre` option when used with `pipenv install`. #3185

- Fix a bug where exception is raised when run pipenv graph in a project without created virtualenv #3201

- When sources are missing names, names will now be derived from the supplied URL. #3216

Vendored Libraries

- Updated `pythonfinder` to correct an issue with unnesting of nested paths when searching for python versions. #3061, #3121

- **Updated vendored dependencies:**

    - `certifi 2018.08.24 => 2018.10.15`

    - `urllib3 1.23 => 1.24`

    - `requests 2.19.1 => 2.20.0`

    - ``shellingham ``1.2.6 => 1.2.7``

    - `tomlkit 0.4.4. => 0.4.6`

    - `vistir 0.1.6 => 0.1.8`

    - `pythonfinder 0.1.2 => 0.1.3`

    - `requirementslib 1.1.9 => 1.1.10`

    - `backports.functools_lru_cache 1.5.0 (new)`

    - `cursor 1.2.0 (new)` #3089

- **Updated vendored dependencies:**

    - `requests 2.19.1 => 2.20.1`

    - `tomlkit 0.4.46 => 0.5.2`

    - `vistir 0.1.6 => 0.2.4`

    - `pythonfinder 1.1.2 => 1.1.8`

    - `requirementslib 1.1.10 => 1.3.0` #3096

- Switch to `tomlkit` for parsing and writing. Drop `prettytoml` and `contoml` from vendors. #3191

- Updated `requirementslib` to aid in resolution of local and remote archives. #3196

Improved Documentation

- Expanded development and testing documentation for contributors to get started. #3074

1.2.15 2018.10.13 (2018-10-13)

Bug Fixes

- Fixed a bug in `pipenv clean` which caused global packages to sometimes be inadvertently targeted for cleanup. #2849

- Fix broken backport imports for vendored vistir. #2950, #2955, #2961

- Fixed a bug with importing local vendored dependencies when running `pipenv graph`. #2952

- Fixed a bug which caused executable discovery to fail when running inside a virtualenv. #2957

- Fix parsing of outline tables. #2971

- Fixed a bug which caused `verify_ssl` to fail to drop through to `pip install` correctly as `trusted-host`. #2979

- Fixed a bug which caused canonicalized package names to fail to resolve against PyPI. #2989

- Enhanced CI detection to detect Azure Devops builds. #2993

- Fixed a bug which prevented installing pinned versions which used redirection symbols from the command line. #2998

- Fixed a bug which prevented installing the local directory in non-editable mode. #3005

Vendored Libraries

- Updated `requirementslib` to version `1.1.9`. #2989

- Upgraded `pythonfinder => 1.1.1` and `vistir => 0.1.7`. #3007

1.2.16 2018.10.9 (2018-10-09)

Features & Improvements

- Added environment variables *PIPENV_VERBOSE* and *PIPENV_QUIET* to control output verbosity without needing to pass options. #2527

- Updated test-PyPI add-on to better support json-API access (forward compatibility). Improved testing process for new contributors. #2568

- Greatly enhanced python discovery functionality:

    - Added pep514 (windows launcher/finder) support for python discovery.

    - Introduced architecture discovery for python installations which support different architectures. #2582

- Added support for `pipenv shell` on msys and cygwin/mingw/git bash for Windows. #2641

- Enhanced resolution of editable and VCS dependencies. #2643

- Deduplicate and refactor CLI to use stateful arguments and object passing. See this issue for reference. #2814

Behavior Changes

- Virtual environment activation for `run` is revised to improve interpolation with other Python discovery tools. #2503

- Improve terminal coloring to display better in Powershell. #2511

- Invoke `virtualenv` directly for virtual environment creation, instead of depending on `pew`. #2518

- `pipenv --help` will now include short help descriptions. #2542

- Add `COMSPEC` to fallback option (along with `SHELL` and `PYENV_SHELL`) if shell detection fails, improving robustness on Windows. #2651

- Fallback to shell mode if *run* fails with Windows error 193 to handle non-executable commands. This should improve usability on Windows, where some users run non-executable files without specifying a command, relying on Windows file association to choose the current command. #2718

Bug Fixes

- Fixed a bug which prevented installation of editable requirements using `ssh://` style URLs #1393

- VCS Refs for locked local editable dependencies will now update appropriately to the latest hash when running `pipenv update`. #1690

- `.tar.gz` and `.zip` artifacts will now have dependencies installed even when they are missing from the Lockfile. #2173

- The command line parser will now handle multiple `-e/--editable` dependencies properly via click's option parser to help mitigate future parsing issues. #2279

- Fixed the ability of pipenv to parse `dependency_links` from `setup.py` when `PIP_PROCESS_DEPENDENCY_LINKS` is enabled. #2434

- Fixed a bug which could cause `-i/--index` arguments to sometimes be incorrectly picked up in packages. This is now handled in the command line parser. #2494

- Fixed non-deterministic resolution issues related to changes to the internal package finder in `pip 10`. #2499, #2529, #2589, #2666, #2767, #2785, #2795, #2801, #2824, #2862, #2879, #2894, #2933

- Fix subshell invocation on Windows for Python 2. #2515

- Fixed a bug which sometimes caused pipenv to throw a `TypeError` or to run into encoding issues when writing a Lockfile on python 2. #2561

- Improve quoting logic for `pipenv run` so it works better with Windows built-in commands. #2563

- Fixed a bug related to parsing VCS requirements with both extras and subdirectory fragments. Corrected an issue in the `requirementslib` parser which led to some markers being discarded rather than evaluated. #2564

- Fixed multiple issues with finding the correct system python locations. #2582

- Catch JSON decoding error to prevent exception when the lock file is of invalid format. #2607

- Fixed a rare bug which could sometimes cause errors when installing packages with custom sources. #2610

- Update requirementslib to fix a bug which could raise an `UnboundLocalError` when parsing malformed VCS URIs. #2617

- Fixed an issue which prevented passing multiple `--ignore` parameters to `pipenv check`. #2632

- Fixed a bug which caused attempted hashing of `ssh://` style URIs which could cause failures during installation of private ssh repositories. - Corrected path conversion issues which caused certain editable VCS paths to be converted to `ssh://` URIs improperly. #2639

- Fixed a bug which caused paths to be formatted incorrectly when using `pipenv shell` in bash for windows. #2641

- Dependency links to private repositories defined via `ssh://` schemes will now install correctly and skip hashing as long as `PIP_PROCESS_DEPENDENCY_LINKS=1`. #2643

- Fixed a bug which sometimes caused pipenv to parse the `trusted_host` argument to pip incorrectly when parsing source URLs which specify `verify_ssl = false`. #2656

- Prevent crashing when a virtual environment in `WORKON_HOME` is faulty. #2676

- Fixed virtualenv creation failure when a .venv file is present in the project root. #2680

- Fixed a bug which could cause the `-e/--editable` argument on a dependency to be accidentally parsed as a dependency itself. #2714

- Correctly pass `verbose` and `debug` flags to the resolver subprocess so it generates appropriate output. This also resolves a bug introduced by the fix to #2527. #2732

- All markers are now included in `pipenv lock --requirements` output. #2748

- Fixed a bug in marker resolution which could cause duplicate and non-deterministic markers. #2760

- Fixed a bug in the dependency resolver which caused regular issues when handling `setup.py` based dependency resolution. #2766

- **Updated vendored dependencies:**

    - `pip-tools` (updated and patched to latest w/ `pip 18.0` compatibility)

    - `pip 10.0.1 => 18.0`

    - `click 6.7 => 7.0`

    - `toml 0.9.4 => 0.10.0`

    - `pyparsing 2.2.0 => 2.2.2`

    - `delegator 0.1.0 => 0.1.1`

    - `attrs 18.1.0 => 18.2.0`

    - `distlib 0.2.7 => 0.2.8`

    - `packaging 17.1.0 => 18.0`

    - `passa 0.2.0 => 0.3.1`

    - `pip_shims 0.1.2 => 0.3.1`

    - `plette 0.1.1 => 0.2.2`

    - `pythonfinder 1.0.2 => 1.1.0`

    - `pytoml 0.1.18 => 0.1.19`

    - `requirementslib 1.1.16 => 1.1.17`

    - `shellingham 1.2.4 => 1.2.6`

    - `tomlkit 0.4.2 => 0.4.4`

    - `vistir 0.1.4 => 0.1.6` #2802,

    #2867, #2880

- Fixed a bug where *pipenv* crashes when the *WORKON_HOME* directory does not exist. #2877

- Fixed pip is not loaded from pipenv's patched one but the system one #2912

- Fixed various bugs related to `pip 18.1` release which prevented locking, installation, and syncing, and dumping to a `requirements.txt` file. #2924

Vendored Libraries

- Pew is no longer vendored. Entry point `pewtwo`, packages `pipenv.pew` and `pipenv.patched.pew` are removed. #2521

- Update `pythonfinder` to major release `1.0.0` for integration. #2582

- Update requirementslib to fix a bug which could raise an `UnboundLocalError` when parsing malformed VCS URIs. #2617

-
    - Vendored new libraries `vistir` and `pip-shims`, `tomlkit`, `modutil`, and `plette`.

- Update vendored libraries: - `scandir` to `1.9.0` - `click-completion` to `0.4.1` - `semver` to `2.8.1` - `shellingham` to `1.2.4` - `pytoml` to `0.1.18` - `certifi` to `2018.8.24` - `ptyprocess` to `0.6.0` - `requirementslib` to `1.1.5` - `pythonfinder` to `1.0.2` - `pipdeptree` to `0.13.0` - `python-dotenv` to `0.9.1` [#2639](#)

- **Updated vendored dependencies:**

    - `pip-tools` (updated and patched to latest w/ `pip 18.0` compatibility)

    - `pip 10.0.1 => 18.0`

    - `click 6.7 => 7.0`

    - `toml 0.9.4 => 0.10.0`

    - `pyparsing 2.2.0 => 2.2.2`

    - `delegator 0.1.0 => 0.1.1`

    - `attrs 18.1.0 => 18.2.0`

    - `distlib 0.2.7 => 0.2.8`

    - `packaging 17.1.0 => 18.0`

    - `passa 0.2.0 => 0.3.1`

    - `pip_shims 0.1.2 => 0.3.1`

    - `plette 0.1.1 => 0.2.2`

    - `pythonfinder 1.0.2 => 1.1.0`

    - `pytoml 0.1.18 => 0.1.19`

    - `requirementslib 1.1.16 => 1.1.17`

    - `shellingham 1.2.4 => 1.2.6`

    - `tomlkit 0.4.2 => 0.4.4`

    - `vistir 0.1.4 => 0.1.6` [#2902](#),

  [#2935](#)

Improved Documentation

- Simplified the test configuration process. [#2568](#)

- Updated documentation to use working fortune cookie add-on. [#2644](#)

- Added additional information about troubleshooting `pipenv shell` by using the the `$PIPENV_SHELL` environment variable. [#2671](#)

- Added a link to `PEP-440` version specifiers in the documentation for additional detail. [#2674](#)

- Added simple example to README.md for installing from git. [#2685](#)

- Stopped recommending *–system* for Docker contexts. [#2762](#)

- Fixed the example url for doing "pipenv install -e some-repository-url#egg=something", it was missing the "egg=" in the fragment identifier. [#2792](#)

- Fixed link to the "be cordial" essay in the contribution documentation. [#2793](#)

- Clarify *pipenv install* documentation [#2844](#)

- Replace reference to uservoice with PEEP-000 [#2909](#)

1.2.17 2018.7.1 (2018-07-01)

Features & Improvements

- All calls to `pipenv shell` are now implemented from the ground up using shellingham, a custom library which was purpose built to handle edge cases and shell detection. #2371

- Added support for python 3.7 via a few small compatibility / bug fixes. #2427, #2434, #2436

- Added new flag `pipenv --support` to replace the diagnostic command `python -m pipenv.help`. #2477, #2478

- Improved import times and CLI run times with minor tweaks. #2485

Bug Fixes

- Fixed an ongoing bug which sometimes resolved incompatible versions into the project Lockfile. #1901

- Fixed a bug which caused errors when creating virtualenvs which contained leading dash characters. #2415

- Fixed a logic error which caused `--deploy --system` to overwrite editable vcs packages in the Pipfile before installing, which caused any installation to fail by default. #2417

- Updated requirementslib to fix an issue with properly quoting markers in VCS requirements. #2419

- Installed new vendored jinja2 templates for `click-completion` which were causing template errors for users with completion enabled. #2422

- Added support for python 3.7 via a few small compatibility / bug fixes. #2427

- Fixed an issue reading package names from `setup.py` files in projects which imported utilities such as `versioneer`. #2433

- Pipenv will now ensure that its internal package names registry files are written with unicode strings. #2450

- Fixed a bug causing requirements input as relative paths to be output as absolute paths or URIs. Fixed a bug affecting normalization of `git+git@host` URLs. #2453

- Pipenv will now always use `pathlib2` for `Path` based filesystem interactions by default on `python<3.5`. #2454

- Fixed a bug which prevented passing proxy PyPI indexes set with `--pypi-mirror` from being passed to pip during virtualenv creation, which could cause the creation to freeze in some cases. #2462

- Using the `python -m pipenv.help` command will now use proper encoding for the host filesystem to avoid encoding issues. #2466

- The new `jinja2` templates for `click_completion` will now be included in pipenv source distributions. #2479

- Resolved a long-standing issue with re-using previously generated `InstallRequirement` objects for resolution which could cause `PKG-INFO` file information to be deleted, raising a `TypeError`. #2480

- Resolved an issue parsing usernames from private PyPI URIs in `Pipfiles` by updating `requirementslib`. #2484

Vendored Libraries

- All calls to `pipenv shell` are now implemented from the ground up using shellingham, a custom library which was purpose built to handle edge cases and shell detection. #2371

- Updated requirementslib to fix an issue with properly quoting markers in VCS requirements. #2419

- Installed new vendored jinja2 templates for `click-completion` which were causing template errors for users with completion enabled. #2422

- Add patch to `prettytoml` to support Python 3.7. #2426

- Patched `prettytoml.AbstractTable._enumerate_items` to handle `StopIteration` errors in preparation of release of python 3.7. #2427

- Fixed an issue reading package names from `setup.py` files in projects which imported utilities such as `versioneer`. #2433

- Updated `requirementslib` to version `1.0.9` #2453

- Unraveled a lot of old, unnecessary patches to `pip-tools` which were causing non-deterministic resolution errors. #2480

- Resolved an issue parsing usernames from private PyPI URIs in `Pipfiles` by updating `requirementslib`. #2484

Improved Documentation

- Added instructions for installing using Fedora's official repositories. #2404

1.2.18 2018.6.25 (2018-06-25)

Features & Improvements

- Pipenv-created virtualenvs will now be associated with a `.project` folder (features can be implemented on top of this later or users may choose to use `pipenv-pipes` to take full advantage of this.) #1861

- Virtualenv names will now appear in prompts for most Windows users. #2167

- Added support for cmder shell paths with spaces. #2168

- Added nested JSON output to the `pipenv graph` command. #2199

- Dropped vendored pip 9 and vendored, patched, and migrated to pip 10. Updated patched piptools version. #2255

- PyPI mirror URLs can now be set to override instances of PyPI URLs by passing the `--pypi-mirror` argument from the command line or setting the `PIPENV_PYPI_MIRROR` environment variable. #2281

- Virtualenv activation lines will now avoid being written to some shell history files. #2287

- Pipenv will now only search for `requirements.txt` files when creating new projects, and during that time only if the user doesn't specify packages to pass in. #2309

- Added support for mounted drives via UNC paths. #2331

- Added support for Windows Subsystem for Linux bash shell detection. #2363

- Pipenv will now generate hashes much more quickly by resolving them in a single pass during locking. #2384

- `pipenv run` will now avoid spawning additional `COMSPEC` instances to run commands in when possible. #2385

- Massive internal improvements to requirements parsing codebase, resolver, and error messaging. #2388

- `pipenv check` now may take multiple of the additional argument `--ignore` which takes a parameter `cve_id` for the purpose of ignoring specific CVEs. #2408

Behavior Changes

- Pipenv will now parse & capitalize `platform_python_implementation` markers .. warning:: This could cause an issue if you have an out of date `Pipfile` which lower-cases the comparison value (e.g. `cpython` instead of `CPython`). #2123

- Pipenv will now only search for `requirements.txt` files when creating new projects, and during that time only if the user doesn't specify packages to pass in. #2309

Bug Fixes

- Massive internal improvements to requirements parsing codebase, resolver, and error messaging. #1962, #2186, #2263, #2312

- Pipenv will now parse & capitalize `platform_python_implementation` markers. #2123

- Fixed a bug with parsing and grouping old-style `setup.py` extras during resolution #2142

- Fixed a bug causing pipenv graph to throw unhelpful exceptions when running against empty or non-existent environments. #2161

- Fixed a bug which caused `--system` to incorrectly abort when users were in a virtualenv. #2181

- Removed vendored `cacert.pem` which could cause issues for some users with custom certificate settings. #2193

- Fixed a regression which led to direct invocations of `virtualenv`, rather than calling it by module. #2198

- Locking will now pin the correct VCS ref during `pipenv update` runs. Running `pipenv update` with a new vcs ref specified in the `Pipfile` will now properly obtain, resolve, and install the specified dependency at the specified ref. #2209

- `pipenv clean` will now correctly ignore comments from `pip freeze` when cleaning the environment. #2262

- Resolution bugs causing packages for incompatible python versions to be locked have been fixed. #2267

- Fixed a bug causing pipenv graph to fail to display sometimes. #2268

- Updated `requirementslib` to fix a bug in Pipfile parsing affecting relative path conversions. #2269

- Windows executable discovery now leverages `os.pathext`. #2298

- Fixed a bug which caused `--deploy --system` to inadvertently create a virtualenv before failing. #2301

- Fixed an issue which led to a failure to unquote special characters in file and wheel paths. #2302

- VCS dependencies are now manually obtained only if they do not match the requested ref. #2304

- Added error handling functionality to properly cope with single-digit `Requires-Python` metadata with no specifiers. #2377

- `pipenv update` will now always run the resolver and lock before ensuring dependencies are in sync with project Lockfile. #2379

- Resolved a bug in our patched resolvers which could cause nondeterministic resolution failures in certain conditions. Running `pipenv install` with no arguments in a project with only a `Pipfile` will now correctly lock first for dependency resolution before installing. #2384

- Patched `python-dotenv` to ensure that environment variables always get encoded to the filesystem encoding. #2386

Improved Documentation

- Update documentation wording to clarify Pipenv's overall role in the packaging ecosystem. #2194

- Added contribution documentation and guidelines. #2205

- Added instructions for supervisord compatibility. #2215

- Fixed broken links to development philosophy and contribution documentation. #2248

Vendored Libraries

- Removed vendored `cacert.pem` which could cause issues for some users with custom certificate settings. #2193

- Dropped vendored pip 9 and vendored, patched, and migrated to pip 10. Updated patched piptools version. #2255

- Updated `requirementslib` to fix a bug in Pipfile parsing affecting relative path conversions. #2269

- Added custom shell detection library `shellingham`, a port of our changes to `pew`. #2363

- Patched `python-dotenv` to ensure that environment variables always get encoded to the filesystem encoding. #2386

- Updated vendored libraries. The following vendored libraries were updated:

    - distlib from version `0.2.6` to `0.2.7`.

    - jinja2 from version `2.9.5` to `2.10`.

    - pathlib2 from version `2.1.0` to `2.3.2`.

    - parse from version `2.8.0` to `2.8.4`.

    - pexpect from version `2.5.2` to `2.6.0`.

    - requests from version `2.18.4` to `2.19.1`.

    - idna from version `2.6` to `2.7`.

    - certifi from version `2018.1.16` to `2018.4.16`.

    - packaging from version `16.8` to `17.1`.

    - six from version `1.10.0` to `1.11.0`.

    - requirementslib from version `0.2.0` to `1.0.1`.

    In addition, scandir was vendored and patched to avoid importing host system binaries when falling back to pathlib2. #2368

CHAPTER 2

User Testimonials

David Gang— *This package manager is really awesome. For the first time I know exactly what my dependencies are which I installed and what the transitive dependencies are. Combined with the fact that installs are deterministic, makes this package manager first class, like cargo.*

Justin Myles Holmes— *Pipenv is finally an abstraction meant to engage the mind instead of merely the filesystem.*

Pipenv Features

- Enables truly *deterministic builds*, while easily specifying *only what you want*.

- Generates and checks file hashes for locked dependencies.

- Automatically install required Pythons, if `pyenv` is available.

- Automatically finds your project home, recursively, by looking for a `Pipfile`.

- Automatically generates a `Pipfile`, if one doesn't exist.

- Automatically creates a virtualenv in a standard location.

- Automatically adds/removes packages to a `Pipfile` when they are installed or uninstalled.

- Automatically loads `.env` files, if they exist.

The main commands are `install`, `uninstall`, and `lock`, which generates a `Pipfile.lock`. These are intended to replace `$ pip install` usage, as well as manual virtualenv management (to activate a virtualenv, run `$ pipenv shell`).

3.1 Basic Concepts

- A virtualenv will automatically be created, when one doesn't exist.

- When no parameters are passed to `install`, all packages `[packages]` specified will be installed.

- To initialize a Python 3 virtual environment, run `$ pipenv --three`.

- To initialize a Python 2 virtual environment, run `$ pipenv --two`.

- Otherwise, whatever virtualenv defaults to will be the default.

3.2 Other Commands

- `graph` will show you a dependency graph of your installed dependencies.

- `shell` will spawn a shell with the virtualenv activated. This shell can be deactivated by using `exit`.

- `run` will run a given command from the virtualenv, with any arguments forwarded (e.g. `$ pipenv run python` or `$ pipenv run pip freeze`).

- `check` checks for security vulnerabilities and asserts that PEP 508 requirements are being met by the current environment.

Further Documentation Guides

4.1 Basic Usage of Pipenv

This document covers some of Pipenv's more basic features.

4.1.1 Example Pipfile & Pipfile.lock

Pipfiles contain information for the dependencies of the project, and supersedes the requirements.txt file used in most Python projects. You should add a Pipfile in the Git repository letting users who clone the repository know the only thing required would be installing Pipenv in the machine and typing `pipenv install`. Pipenv is a reference implementation for using Pipfile. Here is a simple example of a `Pipfile` and the resulting `Pipfile.lock`.

Example Pipfile

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

[packages]
requests = "*"


[dev-packages]
pytest = "*"
```

Example Pipfile.lock

```
{
    "_meta": {
        "hash": {
            "sha256":
→"8d14434df45e0ef884d6c3f6e8048ba72335637a8631cc44792f52fd20b6f97a"
        },
        "host-environment-markers": {
            "implementation_name": "cpython",
            "implementation_version": "3.6.1",
            "os_name": "posix",
            "platform_machine": "x86_64",
            "platform_python_implementation": "CPython",
            "platform_release": "16.7.0",
            "platform_system": "Darwin",
            "platform_version": "Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27␣
→PDT 2017; root:xnu-3789.70.16~2/RELEASE_X86_64",
            "python_full_version": "3.6.1",
            "python_version": "3.6",
            "sys_platform": "darwin"
        },
        "pipfile-spec": 5,
        "requires": {},
        "sources": [
            {
                "name": "pypi",
                "url": "https://pypi.python.org/simple",
                "verify_ssl": true
            }
        ]
    },
    "default": {
        "certifi": {
```

```
            "hashes": [

→"sha256:54a07c09c586b0e4c619f02a5e94e36619da8e2b053e20f594348c0611803704",

→"sha256:40523d2efb60523e113b44602298f0960e900388cf3bb6043f645cf57ea9e3f5"
            ],
            "version": "==2017.7.27.1"
        },
        "chardet": {
            "hashes": [

→"sha256:fc323ffcaeaed0e0a02bf4d117757b98aed530d9ed4531e3e15460124c106691",

→"sha256:84ab92ed1c4d4f16916e05906b6b75a6c0fb5db821cc65e70cbd64a3e2a5eaae"
            ],
            "version": "==3.0.4"
        },
        "idna": {
            "hashes": [

→"sha256:8c7309c718f94b3a625cb648ace320157ad16ff131ae0af362c9f21b80ef6ec4",

→"sha256:2c6a5de3089009e3da7c5dde64a141dbc8551d5b7f6cf4ed7c2568d0cc520a8f"
            ],
            "version": "==2.6"
        },
        "requests": {
            "hashes": [

→"sha256:6a1b267aa90cac58ac3a765d067950e7dbbf75b1da07e895d1f594193a40a38b",

→"sha256:9c443e7324ba5b85070c4a818ade28bfabedf16ea10206da1132edaa6dda237e"
            ],
            "version": "==2.18.4"
        },
        "urllib3": {
            "hashes": [

→"sha256:06330f386d6e4b195fbfc736b297f58c5a892e4440e54d294d7004e3a9bbea1b",

→"sha256:cc44da8e1145637334317feebd728bd869a35285b93cbb4cca2577da7e62db4f"
            ],
            "version": "==1.22"
        }
    },
    "develop": {
        "py": {
            "hashes": [

→"sha256:2ccb79b01769d99115aa600d7eed99f524bf752bba8f041dc1c184853514655a",

→"sha256:0f2d585d22050e90c7d293b6451c83db097df77871974d90efd5a30dc12fcde3"
            ],
            "version": "==1.4.34"
        },
        "pytest": {
            "hashes": [

→"sha256:b84f554f8ddc23add65c411bf112b2d88e2489fd45f753b1cae5936358bdf314",
```

```
→"sha256:f46e49e0340a532764991c498244a60e3a37d7424a532b3ff1a6a7653f1a403a"
        ],
        "version": "==3.2.2"
    }
  }
}
```

4.1.2 General Recommendations & Version Control

- Generally, keep both `Pipfile` and `Pipfile.lock` in version control.

- Do not keep `Pipfile.lock` in version control if multiple versions of Python are being targeted.

- Specify your target Python version in your *Pipfile*'s `[requires]` section. Ideally, you should only have one target Python version, as this is a deployment tool. `python_version` should be in the format `X.Y` (or `X`) and `python_full_version` should be in `X.Y.Z` format.

- `pipenv install` is fully compatible with `pip install` syntax, for which the full documentation can be found here.

- Note that the `Pipfile` uses the TOML Spec.

4.1.3 Example Pipenv Workflow

Clone / create project repository:

```
$ cd myproject
```

Install from Pipfile, if there is one:

```
$ pipenv install
```

Or, add a package to your new project:

```
$ pipenv install <package>
```

This will create a `Pipfile` if one doesn't exist. If one does exist, it will automatically be edited with the new package you provided.

Next, activate the Pipenv shell:

```
$ pipenv shell
$ python --version
```

This will spawn a new shell subprocess, which can be deactivated by using `exit`.

4.1.4 Example Pipenv Upgrade Workflow

- Find out what's changed upstream: `$ pipenv update --outdated`.

- **Upgrade packages, two options:**

    1. Want to upgrade everything? Just do `$ pipenv update`.

    2. Want to upgrade packages one-at-a-time? `$ pipenv update <pkg>` for each outdated package.

4.1.5 Importing from requirements.txt

If you only have a `requirements.txt` file available when running `pipenv install`, pipenv will automatically import the contents of this file and create a `Pipfile` for you.

You can also specify `$ pipenv install -r path/to/requirements.txt` to import a requirements file.

If your requirements file has version numbers pinned, you'll likely want to edit the new `Pipfile` to remove those, and let `pipenv` keep track of pinning. If you want to keep the pinned versions in your `Pipfile.lock` for now, run `pipenv lock --keep-outdated`. Make sure to *upgrade* soon!

4.1.6 Specifying Versions of a Package

You can specify versions of a package using the Semantic Versioning scheme (i.e. `major.minor.micro`).

For example, to install requests you can use:

```
$ pipenv install requests~=1.2
```

Pipenv will install version `1.2` and any minor update, but not `2.0`.

This will update your `Pipfile` to reflect this requirement, automatically.

In general, Pipenv uses the same specifier format as pip. However, note that according to PEP 440 , you can't use versions containing a hyphen or a plus sign.

To make inclusive or exclusive version comparisons you can use:

```
$ pipenv install "requests>=1.4"   # will install a version equal or larger than 1.4.0
$ pipenv install "requests<=2.13"  # will install a version equal or lower than 2.13.0
$ pipenv install "requests>2.19"   # will install 2.19.1 but not 2.19.0
```

Note: The use of double quotes around the package and version specification (i.e. `"requests>2.19"`) is highly recommended to avoid issues with Input and output redirection in Unix-based operating systems.

The use of `~=` is preferred over the `==` identifier as the latter prevents pipenv from updating the packages:

```
$ pipenv install "requests~=2.2"  # locks the major version of the package (this is␣
→equivalent to using >=2.2, ==2.*)
```

To avoid installing a specific version you can use the `!=` identifier.

For an in depth explanation of the valid identifiers and more complex use cases check the relevant section of PEP-440.

4.1.7 Specifying Versions of Python

To create a new virtualenv, using a specific version of Python you have installed (and on your `PATH`), use the `--python VERSION` flag, like so:

Use Python 3:

```
$ pipenv --python 3
```

Use Python3.6:

```
$ pipenv --python 3.6
```

Use Python 2.7.14:

```
$ pipenv --python 2.7.14
```

When given a Python version, like this, Pipenv will automatically scan your system for a Python that matches that given version.

If a `Pipfile` hasn't been created yet, one will be created for you, that looks like this:

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

[dev-packages]

[packages]

[requires]
python_version = "3.6"
```

Note: The inclusion of `[requires] python_version = "3.6"` specifies that your application requires this version of Python, and will be used automatically when running `pipenv install` against this `Pipfile` in the future (e.g. on other machines). If this is not true, feel free to simply remove this section.

If you don't specify a Python version on the command–line, either the `[requires] python_full_version` or `python_version` will be selected automatically, falling back to whatever your system's default `python` installation is, at time of execution.

4.1.8 Editable Dependencies (e.g. `-e .`)

You can tell Pipenv to install a path as editable — often this is useful for the current working directory when working on packages:

```
$ pipenv install --dev -e .

$ cat Pipfile
...
[dev-packages]
"e1839a8" = {path = ".", editable = true}
...
```

Note: All sub-dependencies will get added to the `Pipfile.lock` as well. Sub-dependencies are **not** added to the `Pipfile.lock` if you leave the `-e` option out.

4.1.9 Environment Management with Pipenv

The three primary commands you'll use in managing your pipenv environment are `$ pipenv install`, `$ pipenv uninstall`, and `$ pipenv lock`.

$ pipenv install

`$ pipenv install` is used for installing packages into the pipenv virtual environment and updating your Pipfile.

Along with the basic install command, which takes the form:

```
$ pipenv install [package names]
```

The user can provide these additional parameters:

- `--two` — Performs the installation in a virtualenv using the system `python2` link.
- `--three` — Performs the installation in a virtualenv using the system `python3` link.
- `--python` — Performs the installation in a virtualenv using the provided Python interpreter.

> **Warning:** None of the above commands should be used together. They are also **destructive** and will delete your current virtualenv before replacing it with an appropriately versioned one.

> **Note:** The virtualenv created by Pipenv may be different from what you were expecting. Dangerous characters (i.e. `$ ` ! * @ "` as well as space, line feed, carriage return, and tab) are converted to underscores. Additionally, the full path to the current folder is encoded into a "slug value" and appended to ensure the virtualenv name is unique.

- `--dev` — Install both `develop` and `default` packages from `Pipfile`.
- `--system` — Use the system `pip` command rather than the one from your virtualenv.
- `--deploy` — Make sure the packages are properly locked in Pipfile.lock, and abort if the lock file is out-of-date.
- `--ignore-pipfile` — Ignore the `Pipfile` and install from the `Pipfile.lock`.
- `--skip-lock` — Ignore the `Pipfile.lock` and install from the `Pipfile`. In addition, do not write out a `Pipfile.lock` reflecting changes to the `Pipfile`.

$ pipenv uninstall

`$ pipenv uninstall` supports all of the parameters in *pipenv install*, as well as two additional options, `--all` and `--all-dev`.

- `--all` — This parameter will purge all files from the virtual environment, but leave the Pipfile untouched.
- `--all-dev` — This parameter will remove all of the development packages from the virtual environment, and remove them from the Pipfile.

$ pipenv lock

`$ pipenv lock` is used to create a `Pipfile.lock`, which declares **all** dependencies (and sub-dependencies) of your project, their latest available versions, and the current hashes for the downloaded files. This ensures repeatable, and most importantly *deterministic*, builds.

4.1.10 About Shell Configuration

Shells are typically misconfigured for subshell use, so `$ pipenv shell --fancy` may produce unexpected results. If this is the case, try `$ pipenv shell`, which uses "compatibility mode", and will attempt to spawn a subshell despite misconfiguration.

A proper shell configuration only sets environment variables like `PATH` during a login session, not during every subshell spawn (as they are typically configured to do). In fish, this looks like this:

```
if status --is-login
    set -gx PATH /usr/local/bin $PATH
end
```

You should do this for your shell too, in your `~/.profile` or `~/.bashrc` or wherever appropriate.

Note: The shell launched in interactive mode. This means that if your shell reads its configuration from a specific file for interactive mode (e.g. bash by default looks for a `~/.bashrc` configuration file for interactive mode), then you'll need to modify (or create) this file.

If you experience issues with `$ pipenv shell`, just check the `PIPENV_SHELL` environment variable, which `$ pipenv shell` will use if available. For detail, see *Configuration With Environment Variables*.

4.1.11 A Note about VCS Dependencies

You can install packages with pipenv from git and other version control systems using URLs formatted according to the following rule:

```
<vcs_type>+<scheme>://<location>/<user_or_organization>/<repository>@<branch_or_tag>
→#egg=<package_name>
```

The only optional section is the `@<branch_or_tag>` section. When using git over SSH, you may use the shorthand vcs and scheme alias `git+git@<location>:<user_or_organization>/<repository>@<branch_or_tag>#egg=<package_name>`. Note that this is translated to `git+ssh://git@<location>` when parsed.

Note that it is **strongly recommended** that you install any version-controlled dependencies in editable mode, using `pipenv install -e`, in order to ensure that dependency resolution can be performed with an up to date copy of the repository each time it is performed, and that it includes all known dependencies.

Below is an example usage which installs the git repository located at `https://github.com/requests/requests.git` from tag `v2.20.1` as package name `requests`:

```
$ pipenv install -e git+https://github.com/requests/requests.git@v2.20.1#egg=requests
Creating a Pipfile for this project...
Installing -e git+https://github.com/requests/requests.git@v2.20.1#egg=requests...
[...snipped...]
Adding -e git+https://github.com/requests/requests.git@v2.20.1#egg=requests to Pipfile
→'s [packages]...
[...]

$ cat Pipfile
[packages]
requests = {git = "https://github.com/requests/requests.git", editable = true, ref =
→"v2.20.1"}
```

Valid values for <vcs_type> include `git`, `bzr`, `svn`, and `hg`. Valid values for <scheme> include `http`, `https`, `ssh`, and `file`. In specific cases you also have access to other schemes: `svn` may be combined with `svn` as a scheme, and `bzr` can be combined with `sftp` and `lp`.

You can read more about pip's implementation of VCS support here. For more information about other options available when specifying VCS dependencies, please check the Pipfile spec.

4.1.12 Pipfile.lock Security Features

`Pipfile.lock` takes advantage of some great new security improvements in `pip`. By default, the `Pipfile.lock` will be generated with the sha256 hashes of each downloaded package. This will allow `pip` to guarantee you're installing what you intend to when on a compromised network, or downloading dependencies from an untrusted PyPI endpoint.

We highly recommend approaching deployments with promoting projects from a development environment into production. You can use `pipenv lock` to compile your dependencies on your development environment and deploy the compiled `Pipfile.lock` to all of your production environments for reproducible builds.

Note: If you'd like a `requirements.txt` output of the lockfile, run `$ pipenv lock -r`. This will not include hashes, however. To get a `requirements.txt` you can also use `$ pipenv run pip freeze`.

4.2 Advanced Usage of Pipenv

This document covers some of Pipenv's more glorious and advanced features.

4.2.1 Caveats

- Dependencies of wheels provided in a `Pipfile` will not be captured by `$ pipenv lock`.

- There are some known issues with using private indexes, related to hashing. We're actively working to solve this problem. You may have great luck with this, however.

- Installation is intended to be as deterministic as possible — use the `--sequential` flag to increase this, if experiencing issues.

4.2.2 Specifying Package Indexes

If you'd like a specific package to be installed with a specific package index, you can do the following:

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

[[source]]
url = "http://pypi.home.kennethreitz.org/simple"
verify_ssl = false
name = "home"

[dev-packages]

[packages]
requests = {version="*", index="home"}
maya = {version="*", index="pypi"}
records = "*"
```

Very fancy.

4.2.3 Using a PyPI Mirror

If you would like to override the default PyPI index URLs with the URL for a PyPI mirror, you can use the following:

```
$ pipenv install --pypi-mirror <mirror_url>

$ pipenv update --pypi-mirror <mirror_url>

$ pipenv sync --pypi-mirror <mirror_url>

$ pipenv lock --pypi-mirror <mirror_url>

$ pipenv uninstall --pypi-mirror <mirror_url>
```

Alternatively, you can set the `PIPENV_PYPI_MIRROR` environment variable.

4.2.4 Injecting credentials into Pipfiles via environment variables

Pipenv will expand environment variables (if defined) in your Pipfile. Quite useful if you need to authenticate to a private PyPI:

```
[[source]]
url = "https://$USERNAME:${PASSWORD}@mypypi.example.com/simple"
verify_ssl = true
name = "pypi"
```

Luckily - pipenv will hash your Pipfile *before* expanding environment variables (and, helpfully, will substitute the environment variables again when you install from the lock file - so no need to commit any secrets! Woo!)

If your credentials contain special characters, make sure they are URL-encoded as specified in rfc3986.

Environment variables may be specified as `${MY_ENVAR}` or `$MY_ENVAR`.

On Windows, `%MY_ENVAR%` is supported in addition to `${MY_ENVAR}` or `$MY_ENVAR`.

Environment variables in the URL part of requirement specifiers can also be expanded, where the variable must be in the form of `${VAR_NAME}`. Neither `$VAR_NAME` nor `%VAR_NAME%` is acceptable:

```
[[package]]
requests = {git = "git://${USERNAME}:${PASSWORD}@private.git.com/psf/requests.git",␣
→ref = "2.22.0"}
```

Keep in mind that environment variables are expanded in runtime, leaving the entries in `Pipfile` or `Pipfile.lock` untouched. This is to avoid the accidental leakage of credentials in the source code.

4.2.5 Specifying Basically Anything

If you'd like to specify that a specific package only be installed on certain systems, you can use PEP 508 specifiers to accomplish this.

Here's an example `Pipfile`, which will only install `pywinusb` on Windows systems:

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

[packages]
requests = "*"
pywinusb = {version = "*", sys_platform = "== 'win32'"}
```

Voilà!

Here's a more complex example:

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

[packages]
unittest2 = {version = ">=1.0,<3.0", markers="python_version < '2.7.9' or (python_
→version >= '3.0' and python_version < '3.4')"}
```

Magic. Pure, unadulterated magic.

4.2.6 Using pipenv for Deployments

You may want to use `pipenv` as part of a deployment process.

You can enforce that your `Pipfile.lock` is up to date using the `--deploy` flag:

```
$ pipenv install --deploy
```

This will fail a build if the `Pipfile.lock` is out–of–date, instead of generating a new one.

Or you can install packages exactly as specified in `Pipfile.lock` using the `sync` command:

```
$ pipenv sync
```

Note: `pipenv install --ignore-pipfile` is nearly equivalent to `pipenv sync`, but `pipenv sync` will *never* attempt to re-lock your dependencies as it is considered an atomic operation. `pipenv install` by default does attempt to re-lock unless using the `--deploy` flag.

Deploying System Dependencies

You can tell Pipenv to install a Pipfile's contents into its parent system with the `--system` flag:

```
$ pipenv install --system
```

This is useful for managing the system Python, and deployment infrastructure (e.g. Heroku does this).

4.2.7 Pipenv and Other Python Distributions

To use Pipenv with a third-party Python distribution (e.g. Anaconda), you simply provide the path to the Python binary:

```
$ pipenv install --python=/path/to/python
```

Anaconda uses Conda to manage packages. To reuse Conda–installed Python packages, use the `--site-packages` flag:

```
$ pipenv --python=/path/to/python --site-packages
```

4.2.8 Generating a `requirements.txt`

You can convert a `Pipfile` and `Pipfile.lock` into a `requirements.txt` file very easily, and get all the benefits of extras and other goodies we have included.

Let's take this `Pipfile`:

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

[packages]
requests = {version="*"}

[dev-packages]
pytest = {version="*"}
```

And generate a set of requirements out of it with only the default dependencies:

```
$ pipenv lock -r
chardet==3.0.4
requests==2.18.4
certifi==2017.7.27.1
idna==2.6
urllib3==1.22
```

As with other commands, passing `--dev` will include both the default and development dependencies:

```
$ pipenv lock -r --dev
chardet==3.0.4
requests==2.18.4
certifi==2017.7.27.1
idna==2.6
urllib3==1.22
py==1.4.34
pytest==3.2.3
```

Finally, if you wish to generate a requirements file with only the development requirements you can do that too, using the `--dev-only` flag:

```
$ pipenv lock -r --dev-only
py==1.4.34
pytest==3.2.3
```

The locked requirements are written to stdout, with shell output redirection used to write them to a file:

```
$ pipenv lock -r > requirements.txt
$ pipenv lock -r --dev-only > dev-requirements.txt
$ cat requirements.txt
chardet==3.0.4
requests==2.18.4
certifi==2017.7.27.1
idna==2.6
urllib3==1.22
$ cat dev-requirements.txt
py==1.4.34
pytest==3.2.3
```

4.2.9 Detection of Security Vulnerabilities

Pipenv includes the safety package, and will use it to scan your dependency graph for known security vulnerabilities!

Example:

```
$ cat Pipfile
[packages]
django = "==1.10.1"

$ pipenv check
Checking PEP 508 requirements...
Passed!
Checking installed package safety...

33075: django >=1.10,<1.10.3 resolved (1.10.1 installed)!
Django before 1.8.x before 1.8.16, 1.9.x before 1.9.11, and 1.10.x before 1.10.3,␣
→when settings.DEBUG is True, allow remote attackers to conduct DNS rebinding␣
→attacks by leveraging failure to validate the HTTP Host header against settings.
→ALLOWED_HOSTS.
```

```
33076: django >=1.10,<1.10.3 resolved (1.10.1 installed)!
Django 1.8.x before 1.8.16, 1.9.x before 1.9.11, and 1.10.x before 1.10.3 use a␣
→hardcoded password for a temporary database user created when running tests with an␣
→Oracle database, which makes it easier for remote attackers to obtain access to the␣
→database server by leveraging failure to manually specify a password in the␣
→database settings TEST dictionary.

33300: django >=1.10,<1.10.7 resolved (1.10.1 installed)!
CVE-2017-7233: Open redirect and possible XSS attack via user-supplied numeric␣
→redirect URLs
===============================================================================================

Django relies on user input in some cases  (e.g.
:func:`django.contrib.auth.views.login` and :doc:`i18n </topics/i18n/index>`)
to redirect the user to an "on success" URL. The security check for these
redirects (namely ``django.utils.http.is_safe_url()``) considered some numeric
URLs (e.g. ``http:999999999``) "safe" when they shouldn't be.

Also, if a developer relies on ``is_safe_url()`` to provide safe redirect
targets and puts such a URL into a link, they could suffer from an XSS attack.

CVE-2017-7234: Open redirect vulnerability in ``django.views.static.serve()``
=============================================================================

A maliciously crafted URL to a Django site using the
:func:`~django.views.static.serve` view could redirect to any other domain. The
view no longer does any redirects as they don't provide any known, useful
functionality.

Note, however, that this view has always carried a warning that it is not
hardened for production use and should be used only as a development aid.
```

Note: Each month, *PyUp.io* updates the `safety` database of insecure Python packages and makes it available to the community for free. Pipenv makes an API call to retrieve those results and use them each time you run `pipenv check` to show you vulnerable dependencies.

For more up-to-date vulnerability data, you may also use your own safety API key by setting the environment variable `PIPENV_PYUP_API_KEY`.

4.2.10 Community Integrations

There are a range of community-maintained plugins and extensions available for a range of editors and IDEs, as well as different products which integrate with Pipenv projects:

- Heroku (Cloud Hosting)
- Platform.sh (Cloud Hosting)
- PyUp (Security Notification)
- Emacs (Editor Integration)
- Fish Shell (Automatic `$ pipenv shell!`)
- VS Code (Editor Integration)

- PyCharm (Editor Integration)

Works in progress:

- Sublime Text (Editor Integration)

- Mysterious upcoming Google Cloud product (Cloud Hosting)

4.2.11 Open a Module in Your Editor

Pipenv allows you to open any Python module that is installed (including ones in your codebase), with the `$ pipenv open` command:

```
$ pipenv install -e git+https://github.com/kennethreitz/background.git#egg=background
Installing -e git+https://github.com/kennethreitz/background.git#egg=background...
...
Updated Pipfile.lock!

$ pipenv open background
Opening '/Users/kennethreitz/.local/share/virtualenvs/hmm-mGOawwm_/src/background/
↪background.py' in your EDITOR.
```

This allows you to easily read the code you're consuming, instead of looking it up on GitHub.

> **Note:** The standard `EDITOR` environment variable is used for this. If you're using VS Code, for example, you'll want to `export EDITOR=code` (if you're on macOS you will want to install the command on to your `PATH` first).

4.2.12 Automatic Python Installation

If you have pyenv installed and configured, Pipenv will automatically ask you if you want to install a required version of Python if you don't already have it available.

This is a very fancy feature, and we're very proud of it:

```
$ cat Pipfile
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true

[dev-packages]

[packages]
requests = "*"

[requires]
python_version = "3.6"

$ pipenv install
Warning: Python 3.6 was not found on your system...
Would you like us to install latest CPython 3.6 with pyenv? [Y/n]: y
Installing CPython 3.6.2 with pyenv (this may take a few minutes)...
...
Making Python installation global...
Creating a virtualenv for this project...
Using /Users/kennethreitz/.pyenv/shims/python3 to create virtualenv...
```

```
...
No package provided, installing all dependencies.
...
Installing dependencies from Pipfile.lock...
    5/5 -- 00:00:03
To activate this project's virtualenv, run the following:
 $ pipenv shell
```

Pipenv automatically honors both the `python_full_version` and `python_version` PEP 508 specifiers.

4.2.13 Automatic Loading of `.env`

If a `.env` file is present in your project, `$ pipenv shell` and `$ pipenv run` will automatically load it, for you:

```
$ cat .env
HELLO=WORLD

$ pipenv run python
Loading .env environment variables...
Python 2.7.13 (default, Jul 18 2017, 09:17:00)
[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.environ['HELLO']
'WORLD'
```

Shell like variable expansion is available in `.env` files using *${VARNAME}* syntax.:

```
$ cat .env
CONFIG_PATH=${HOME}/.config/foo

$ pipenv run python
Loading .env environment variables...
Python 3.7.6 (default, Dec 19 2019, 22:52:49)
[GCC 9.2.1 20190827 (Red Hat 9.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.environ['CONFIG_PATH']
'/home/kennethreitz/.config/foo'
```

This is very useful for keeping production credentials out of your codebase. We do not recommend committing `.env` files into source control!

If your `.env` file is located in a different path or has a different name you may set the `PIPENV_DOTENV_LOCATION` environment variable:

```
$ PIPENV_DOTENV_LOCATION=/path/to/.env pipenv shell
```

To prevent pipenv from loading the `.env` file, set the `PIPENV_DONT_LOAD_ENV` environment variable:

```
$ PIPENV_DONT_LOAD_ENV=1 pipenv shell
```

See theskumar/python-dotenv for more information on `.env` files.

4.2.14 Custom Script Shortcuts

Pipenv supports creating custom shortcuts in the (optional) `[scripts]` section of your Pipfile.

You can then run `pipenv run <shortcut name>` in your terminal to run the command in the context of your pipenv virtual environment even if you have not activated the pipenv shell first.

For example, in your Pipfile:

```
[scripts]
printspam = "python -c \"print('I am a silly example, no one would need to do this')\"
↪"
```

And then in your terminal:

```
$ pipenv run printspam
I am a silly example, no one would need to do this
```

Commands that expect arguments will also work. For example:

```
[scripts]
echospam = "echo I am really a very silly example"
```

```
$ pipenv run echospam "indeed"
I am really a very silly example indeed
```

You can then display the names and commands of your shortcuts by running `pipenv scripts` in your terminal.

```
$ pipenv scripts
command   script
echospam  echo I am really a very silly example
```

4.2.15 Configuration With Environment Variables

Pipenv comes with a handful of options that can be enabled via shell environment variables. To activate them, simply create the variable in your shell and pipenv will detect it.

`pipenv.environments.`**`env_to_bool`**(*val*)

> Convert **val** to boolean, returning True if truthy or False if falsey
>
> > **Parameters** **`val`** (`Any`) – The value to convert
> >
> > **Returns** False if Falsey, True if truthy
> >
> > **Return type** bool

`pipenv.environments.`**`get_from_env`**(*arg*, *prefix='PIPENV'*, *check_for_negation=True*)

> Check the environment for a variable, returning its truthy or stringified value
>
> For example, setting `PIPENV_NO_RESOLVE_VCS=1` would mean that `get_from_env("RESOLVE_VCS", prefix="PIPENV")` would return `False`.
>
> > **Parameters**
> >
> > - **`arg`** (`str`) – The name of the variable to look for
> >
> > - **`prefix`** (`str`) – The prefix to attach to the variable, defaults to "PIPENV"
> >
> > - **`check_for_negation`** (`bool`) – Whether to check for `<PREFIX>_NO_<arg>`, defaults to True

> **Returns** The value from the environment if available
>
> **Return type** Optional[Union[str, bool]]

pipenv.environments.**is_in_virtualenv**()
    Check virtualenv membership dynamically

> **Returns** True or false depending on whether we are in a regular virtualenv or not
>
> **Return type** bool

pipenv.environments.**is_using_venv**()
    Check for venv-based virtual environment which sets sys.base_prefix

If you'd like to set these environment variables on a per-project basis, I recommend utilizing the fantastic direnv project, in order to do so.

Also note that pip itself supports environment variables, if you need additional customization.

For example:

```
$ PIP_INSTALL_OPTION="-- -DCMAKE_BUILD_TYPE=Release" pipenv install -e .
```

4.2.16 Custom Virtual Environment Location

Pipenv automatically honors the WORKON_HOME environment variable, if you have it set — so you can tell pipenv to store your virtual environments wherever you want, e.g.:

```
export WORKON_HOME=~/.venvs
```

In addition, you can also have Pipenv stick the virtualenv in project/.venv by setting the PIPENV_VENV_IN_PROJECT environment variable.

4.2.17 Testing Projects

Pipenv is being used in projects like Requests for declaring development dependencies and running the test suite.

We have currently tested deployments with both Travis-CI and tox with success.

Travis CI

An example Travis CI setup can be found in Requests. The project uses a Makefile to define common functions such as its init and tests commands. Here is a stripped down example .travis.yml:

```
language: python
python:
    - "2.6"
    - "2.7"
    - "3.3"
    - "3.4"
    - "3.5"
    - "3.6"
    - "3.7-dev"

# command to install dependencies
install: "make"
```

```
# command to run tests
script:
    - make test
```

and the corresponding Makefile:

```
init:
    pip install pipenv
    pipenv install --dev

test:
    pipenv run pytest tests
```

Tox Automation Project

Alternatively, you can configure a `tox.ini` like the one below for both local and external testing:

```
[tox]
envlist = flake8-py3, py26, py27, py33, py34, py35, py36, pypy

[testenv]
deps = pipenv
commands=
    pipenv install --dev
    pipenv run pytest tests

[testenv:flake8-py3]
basepython = python3.4
commands=
    pipenv install --dev
    pipenv run flake8 --version
    pipenv run flake8 setup.py docs project test
```

Pipenv will automatically use the virtualenv provided by `tox`. If `pipenv install --dev` installs e.g. `pytest`, then installed command `pytest` will be present in given virtualenv and can be called directly by `pytest tests` instead of `pipenv run pytest tests`.

You might also want to add `--ignore-pipfile` to `pipenv install`, as to not accidentally modify the lock-file on each test run. This causes Pipenv to ignore changes to the `Pipfile` and (more importantly) prevents it from adding the current environment to `Pipfile.lock`. This might be important as the current environment (i.e. the virtualenv provisioned by tox) will usually contain the current project (which may or may not be desired) and additional dependencies from `tox`'s `deps` directive. The initial provisioning may alternatively be disabled by adding `skip_install = True` to tox.ini.

This method requires you to be explicit about updating the lock-file, which is probably a good idea in any case.

A 3rd party plugin, tox-pipenv is also available to use Pipenv natively with tox.

4.2.18 Shell Completion

To enable completion in fish, add this to your configuration:

```
eval (env _PIPENV_COMPLETE=fish_source pipenv)
```

Alternatively, with zsh, add this to your configuration:

```
eval "$(_PIPENV_COMPLETE=zsh_source pipenv)"
```

Alternatively, with bash, add this to your configuration:

```
eval "$(_PIPENV_COMPLETE=bash_source pipenv)"
```

Magic shell completions are now enabled!

4.2.19 Working with Platform-Provided Python Components

It's reasonably common for platform specific Python bindings for operating system interfaces to only be available through the system package manager, and hence unavailable for installation into virtual environments with *pip*. In these cases, the virtual environment can be created with access to the system *site-packages* directory:

```
$ pipenv --three --site-packages
```

To ensure that all *pip*-installable components actually are installed into the virtual environment and system packages are only used for interfaces that don't participate in Python-level dependency resolution at all, use the *PIP_IGNORE_INSTALLED* setting:

```
$ PIP_IGNORE_INSTALLED=1 pipenv install --dev
```

4.2.20 Pipfile vs setup.py

There is a subtle but very important distinction to be made between **applications** and **libraries**. This is a very common source of confusion in the Python community.

Libraries provide reusable functionality to other libraries and applications (let's use the umbrella term **projects** here). They are required to work alongside other libraries, all with their own set of sub-dependencies. They define **abstract dependencies**. To avoid version conflicts in sub-dependencies of different libraries within a project, libraries should never ever pin dependency versions. Although they may specify lower or (less frequently) upper bounds, if they rely on some specific feature/fix/bug. Library dependencies are specified via `install_requires` in `setup.py`.

Libraries are ultimately meant to be used in some **application**. Applications are different in that they usually are not depended on by other projects. They are meant to be deployed into some specific environment and only then should the exact versions of all their dependencies and sub-dependencies be made concrete. To make this process easier is currently the main goal of Pipenv.

To summarize:

- For libraries, define **abstract dependencies** via `install_requires` in `setup.py`. The decision of which version exactly to be installed and where to obtain that dependency is not yours to make!

- For applications, define **dependencies and where to get them** in the *Pipfile* and use this file to update the set of **concrete dependencies** in `Pipfile.lock`. This file defines a specific idempotent environment that is known to work for your project. The `Pipfile.lock` is your source of truth. The `Pipfile` is a convenience for you to create that lock-file, in that it allows you to still remain somewhat vague about the exact version of a dependency to be used. Pipenv is there to help you define a working conflict-free set of specific dependency-versions, which would otherwise be a very tedious task.

- Of course, `Pipfile` and Pipenv are still useful for library developers, as they can be used to define a development or test environment.

- And, of course, there are projects for which the distinction between library and application isn't that clear. In that case, use `install_requires` alongside Pipenv and `Pipfile`.

You can also do this:

```
$ pipenv install -e .
```

This will tell Pipenv to lock all your `setup.py`–declared dependencies.

4.2.21 Changing Pipenv's Cache Location

You can force Pipenv to use a different cache location by setting the environment variable `PIPENV_CACHE_DIR` to the location you wish. This is useful in the same situations that you would change `PIP_CACHE_DIR` to a different directory.

4.2.22 Changing Default Python Versions

By default, Pipenv will initialize a project using whatever version of python the python3 is. Besides starting a project with the `--three` or `--two` flags, you can also use `PIPENV_DEFAULT_PYTHON_VERSION` to specify what version to use when starting a project when `--three` or `--two` aren't used.

4.3 Pipenv CLI Reference

4.4 Frequently Encountered Pipenv Problems

Pipenv is constantly being improved by volunteers, but is still a very young project with limited resources, and has some quirks that needs to be dealt with. We need everyone's help (including yours!).

Here are some common questions people have using Pipenv. Please take a look below and see if they resolve your problem.

Note: Make sure you're running the newest Pipenv version first!

4.4.1 Your dependencies could not be resolved

Make sure your dependencies actually *do* resolve. If you're confident they are, you may need to clear your resolver cache. Run the following command:

```
pipenv lock --clear
```

and try again.

If this does not work, try manually deleting the whole cache directory. It is usually one of the following locations:

- `~/Library/Caches/pipenv` (macOS)
- `%LOCALAPPDATA%\pipenv\pipenv\Cache` (Windows)
- `~/.cache/pipenv` (other operating systems)

Pipenv does not install pre-releases (i.e. a version with an alpha/beta/etc. suffix, such as *1.0b1*) by default. You will need to pass the `--pre` flag in your command, or set

```
[pipenv]
allow_prereleases = true
```

in your Pipfile.

4.4.2 No module named <module name>

This is usually a result of mixing Pipenv with system packages. We *strongly* recommend installing Pipenv in an isolated environment. Uninstall all existing Pipenv installations, and see *Installing Pipenv* to choose one of the recommended way to install Pipenv instead.

4.4.3 My pyenv-installed Python is not found

Make sure you have PYENV_ROOT set correctly. Pipenv only supports CPython distributions, with version name like 3.6.4 or similar.

4.4.4 Pipenv does not respect pyenv's global and local Python versions

Pipenv by default uses the Python it is installed against to create the virtualenv. You can set the --python option to $(pyenv which python) to use your current pyenv interpreter. See *Specifying Versions of a Package* for more information.

4.4.5 ValueError: unknown locale: UTF-8

macOS has a bug in its locale detection that prevents us from detecting your shell encoding correctly. This can also be an issue on other systems if the locale variables do not specify an encoding.

The workaround is to set the following two environment variables to a standard localization format:

- LC_ALL
- LANG

For Bash, for example, you can add the following to your ~/.bash_profile:

```
export LC_ALL='en_US.UTF-8'
export LANG='en_US.UTF-8'
```

For Zsh, the file to edit is ~/.zshrc.

Note: You can change both the en_US and UTF-8 part to the language/locale and encoding you use.

4.4.6 /bin/pip: No such file or directory

This may be related to your locale setting. See *ValueError: unknown locale: UTF-8* for a possible solution.

4.4.7 Pipenv does not respect dependencies in setup.py

No, it does not, intentionally. Pipfile and setup.py serve different purposes, and should not consider each other by default. See *Pipfile vs setup.py* for more information.

4.4.8 Using `pipenv run` in Supervisor program

When you configure a supervisor program's `command` with `pipenv run ...`, you need to set locale environment variables properly to make it work.

Add this line under `[supervisord]` section in `/etc/supervisor/supervisord.conf`:

```
[supervisord]
environment=LC_ALL='en_US.UTF-8',LANG='en_US.UTF-8'
```

4.4.9 An exception is raised during `Locking dependencies...`

Run `pipenv lock --clear` and try again. The lock sequence caches results to speed up subsequent runs. The cache may contain faulty results if a bug causes the format to corrupt, even after the bug is fixed. `--clear` flushes the cache, and therefore removes the bad results.

Contribution Guides

5.1 Development Philosophy

Pipenv is an open but opinionated tool, created by an open but opinionated developer.

5.1.1 Management Style

To be updated (as of March 2020).

Kenneth Reitz is the BDFL. He has final say in any decision related to the Pipenv project. Kenneth is responsible for the direction and form of the library, as well as its presentation. In addition to making decisions based on technical merit, he is responsible for making decisions based on the development philosophy of Pipenv.

Dan Ryan, Tzu-ping Chung, and Nate Prewitt are the core contributors. They are responsible for triaging bug reports, reviewing pull requests and ensuring that Kenneth is kept up to speed with developments around the library. The day-to-day managing of the project is done by the core contributors. They are responsible for making judgments about whether or not a feature request is likely to be accepted by Kenneth.

5.1.2 Values

- Simplicity is always better than functionality.

- Listen to everyone, then disregard it.

- The API is all that matters. Everything else is secondary.

- Fit the 90% use-case. Ignore the nay-sayers.

5.2 Contributing to Pipenv

If you're reading this, you're probably interested in contributing to Pipenv. Thank you very much! Open source projects live-and-die based on the support they receive from others, and the fact that you're even considering contributing to the Pipenv project is *very* generous of you.

This document lays out guidelines and advice for contributing to this project. If you're thinking of contributing, please start by reading this document and getting a feel for how contributing to this project works. If you have any questions, feel free to reach out to either Dan Ryan, Tzu-ping Chung, or Nate Prewitt, the primary maintainers.

The guide is split into sections based on the type of contribution you're thinking of making, with a section that covers general guidelines for all contributors.

5.2.1 General Guidelines

Be Cordial

> **Be cordial or be on your way**. —*Kenneth Reitz*

Pipenv has one very important rule governing all forms of contribution, including reporting bugs or requesting features. This golden rule is "be cordial or be on your way".

All contributions are welcome, as long as everyone involved is treated with respect.

Get Early Feedback

If you are contributing, do not feel the need to sit on your contribution until it is perfectly polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback in no way prejudices your chances of getting that contribution accepted, and can save you from putting a lot of work into a contribution that is not suitable for the project.

Contribution Suitability

Our project maintainers have the last word on whether or not a contribution is suitable for Pipenv. All contributions will be considered carefully, but from time to time, contributions will be rejected because they do not suit the current goals or needs of the project.

If your contribution is rejected, don't despair! As long as you followed these guidelines, you will have a much better chance of getting your next contribution accepted.

5.2.2 Questions

The GitHub issue tracker is for *bug reports* and *feature requests*. Please do not use it to ask questions about how to use Pipenv. These questions should instead be directed to Stack Overflow. Make sure that your question is tagged with the `pipenv` tag when asking it on Stack Overflow, to ensure that it is answered promptly and accurately.

5.2.3 Code Contributions

Steps for Submitting Code

When contributing code, you'll want to follow this checklist:

1. Understand our development philosophy.

2. Fork the repository on GitHub.

3. Set up your *Development Setup*

4. Run the tests (*Testing*) to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report by following the guidelines in this document: *Bug Reports*.

5. Write tests that demonstrate your bug or feature. Ensure that they fail.

6. Make your change.

7. Run the entire test suite again, confirming that all tests pass *including the ones you just added*.

8. Send a GitHub Pull Request to the main repository's `master` branch. GitHub Pull Requests are the expected method of code collaboration on this project.

The following sub-sections go into more detail on some of the points above.

Development Setup

To get your development environment setup, run:

```
pip install -e .
pipenv install --dev
```

This will install the repository version of Pipenv and then install the development dependencies. Once that has completed, you can start developing.

The repository version of Pipenv must be installed over other global versions to resolve conflicts with the `pipenv` folder being implicitly added to `sys.path`. See pypa/pipenv#2557 for more details.

Testing

Tests are written in `pytest` style and can be run very simply:

```
pytest
```

This will run all Pipenv tests, which can take awhile. To run a subset of the tests, the standard pytest filters are available, such as:

- provide a directory or file: `pytest tests/unit` or `pytest tests/unit/test_cmdparse.py`

- provide a keyword expression: `pytest -k test_lock_editable_vcs_without_install`

- provide a nodeid: `pytest tests/unit/test_cmdparse.py::test_parse`

- provide a test marker: `pytest -m lock`

Code Review

Contributions will not be merged until they have been code reviewed. You should implement any code review feedback unless you strongly object to it. In the event that you object to the code review feedback, you should make your case clearly and calmly. If, after doing so, the feedback is judged to still apply, you must either apply the feedback or withdraw your contribution.

Package Index

To speed up testing, tests that rely on a package index for locking and installing use a local server that contains vendored packages in the `tests/pypi` directory. Each vendored package should have it's own folder containing the necessary releases. When adding a release for a package, it is easiest to use either the `.tar.gz` or universal wheels (ex: `py2.py3-none`). If a `.tar.gz` or universal wheel is not available, add wheels for all available architectures and platforms.

5.2.4 Documentation Contributions

Documentation improvements are always welcome! The documentation files live in the `docs/` directory of the codebase. They're written in reStructuredText, and use Sphinx to generate the full suite of documentation.

When contributing documentation, please do your best to follow the style of the documentation files. This means a soft-limit of 79 characters wide in your text files and a semi-formal, yet friendly and approachable, prose style.

When presenting Python code, use single-quoted strings (`'hello'` instead of `"hello"`).

5.2.5 Bug Reports

Bug reports are hugely important! They are recorded as GitHub issues. Please be aware of the following things when filing bug reports:

1. Avoid raising duplicate issues. *Please* use the GitHub issue search feature to check whether your bug report or feature request has been mentioned in the past. Duplicate bug reports and feature requests are a huge maintenance burden on the limited resources of the project. If it is clear from your report that you would have struggled to find the original, that's okay, but if searching for a selection of words in your issue title would have found the duplicate then the issue will likely be closed extremely abruptly.

2. When filing bug reports about exceptions or tracebacks, please include the *complete* traceback. Partial tracebacks, or just the exception text, are not helpful. Issues that do not contain complete tracebacks may be closed without warning.

3. Make sure you provide a suitable amount of information to work with. This means you should provide:

   - Guidance on **how to reproduce the issue**. Ideally, this should be a *small* code sample that can be run immediately by the maintainers. Failing that, let us know what you're doing, how often it happens, what environment you're using, etc. Be thorough: it prevents us needing to ask further questions.

   - Tell us **what you expected to happen**. When we run your example code, what are we expecting to happen? What does "success" look like for your code?

   - Tell us **what actually happens**. It's not helpful for you to say "it doesn't work" or "it fails". Tell us *how* it fails: do you get an exception? A hang? The packages installed seem incorrect? How was the actual result different from your expected result?

   - Tell us **what version of Pipenv you're using**, and **how you installed it**. Different versions of Pipenv behave differently and have different bugs, and some distributors of Pipenv ship patches on top of the code we supply.

   If you do not provide all of these things, it will take us much longer to fix your problem. If we ask you to clarify these and you never respond, we will close your issue without fixing it.

5.2.6 Run the tests

Three ways of running the tests are as follows:

1. `make test` (which uses `docker`)

2. `./run-tests.sh` or `run-tests.bat`

3. Using pipenv:

```
$ git clone https://github.com/pypa/pipenv.git
$ cd pipenv
$ git submodule sync && git submodule update --init --recursive
$ pipenv install --dev
$ pipenv run pytest
```

For the last two, it is important that your environment is setup correctly, and this may take some work, for example, on a specific Mac installation, the following steps may be needed:

```
# Make sure the tests can access github
if [ "$SSH_AGENT_PID" = "" ]
then
   eval `ssh-agent`
   ssh-add
fi

# Use unix like utilities, installed with brew,
# e.g. brew install coreutils
for d in /usr/local/opt/*/libexec/gnubin /usr/local/opt/python/libexec/bin
do
  [[ ":$PATH:" != *":$d:"* ]] && PATH="$d:${PATH}"
done

export PATH

# PIP_FIND_LINKS currently breaks test_uninstall.py
unset PIP_FIND_LINKS
```

CHAPTER 6

Indices and tables

- genindex

- modindex

Python Module Index

p
pipenv.environments, 53

E

G

I

P