VirtualFish Documentation

Release 2.5.4

Leigh Brenecki, Justin Mayer, and contributors

Sep 07, 2021

Contents

VirtualFish is a Python virtual environment manager for the Fish shell.

CHAPTER 1

Contents

1.1 Installation and Setup

1.1.1 Installing

1. Make sure you are running Fish 3.1+. If you are running an Ubuntu LTS release that has an older Fish version, install Fish via the Fish 3.x release series PPA.
2. The easiest way to install VirtualFish is by running: `python -m pip install --user virtualfish`. If you're using Pipx, it is better to use: `pipx install virtualfish`.
3. Install the VirtualFish loader by running:

   ```
   vf install
   ```

   If you want to use VirtualFish with *plugins*, list the names of the plugins as arguments to the install command:

   ```
   vf install compat_aliases projects environment
   ```

   Note: After performing the above step, you will be prompted to run `exec fish` in order to make these changes active in your current shell session.
4. Customize your `fish_prompt`

1.1.2 Customizing Your `fish_prompt`

VirtualFish doesn't attempt to mess with your prompt. Since Fish's prompt is a function, it is both much less straightforward to change it automatically, and much more convenient to simply customize it manually to your liking.

The easiest way to add the active virtual environment's name to your prompt is to type `funced fish_prompt` and add the following line somewhere:

```
if set -q VIRTUAL_ENV
    echo -n -s (set_color -b blue white) "(" (basename "$VIRTUAL_ENV") ")" (set_color␣
↪normal) " "
end
```

Then, type `funcsave fish_prompt` to save your new prompt to disk.

1.1.3 Un-installing

To un-install VirtualFish, run:

```
vf uninstall
python -m pip uninstall virtualfish
```

1.2 Usage

1.2.1 Commands

- `vf new [<options>] <envname>` - Create a virtual environment.
- `vf ls [--details]` - List the available virtual environments.
- `vf activate <envname>` - Activate a virtual environment. (Note: Doesn't use the `activate.fish` script provided by Virtualenv.)
- `vf deactivate` - Deactivate the current virtual environment.
- `vf upgrade [<options>] [<envname(s)>]` - Upgrade virtual environment(s).
- `vf rm <envname>` - Delete a virtual environment.
- `vf tmp [<options>]` - Create a temporary virtual environment with a randomly generated name that will be removed when it is deactivated.
- `vf cd` - Change directory to currently-activated virtual environment.
- `vf cdpackages` - Change directory to currently-active virtual environment's site-packages.
- `vf globalpackages` - Toggle system site packages.
- `vf addpath` - Add a directory to this virtual environment's `sys.path`.
- `vf all <command>` - Run a command in all virtual environments sequentially.
- `vf connect [<envname>]` - Connect the current working directory with the currently active (or specified) virtual environment. This requires the *auto-activation plugin* to be enabled in order to have any effect besides creating a `.venv` file in the current directory.

If you are accustomed to virtualenvwrapper commands (`workon`, etc.), you may wish to enable the *Virtualenvwrapper Compatibility Aliases (compat_aliases)* plugin.

1.2.2 Using Different Pythons

By default, the environments you create with VirtualFish will use the same Python version that was originally used to Pip-install VirtualFish, which will usually be your system's default Python interpreter.

If you want to create a new virtual environment with a different Python interpreter, add the `--python PYTHON_EXE` (`-p` for brevity) flag to `vf new`, where `PYTHON_EXE` is any Python executable. For example:

```
vf new -p /usr/bin/python3 my_python3_env
```

Specifying the full path to the Python executable avoids ambiguity and is thus the most reliable option, but if the target Python executable is on your `PATH`, you can save a few keystrokes and pass the bare executable instead:

```
vf new -p pypy my_pypy_env
```

Sometimes there may be Python interpreters on your system that are not on your `PATH`, with full filesystem paths that are long and thus hard to remember and type. VirtualFish makes dealing with these easier by automatically detecting and using Python interpreters in a few known situations, in the following order:

1. asdf Python plugin is installed and has built the specified Python version.
2. Pyenv is installed and has built the specified Python version.
3. Pythonz is installed and has built the specified Python version.
4. Homebrew keg-only versioned Python executable (e.g., 3.8) found at: `/usr/local/opt/python@3.8/bin/python3.8`

For asdf, Pyenv, and Pythonz , in addition to passing option flags such as `-p python3.8` or `-p python3.9.0a4`, you can even get away with specifying just the version numbers, such as `-p 3.8` or `-p 3.9.0a4`.

1.2.3 Upgrading Virtual Environments

Virtual environments contain links to Python interpreters that can become outdated over time. In addition, sometimes the underlying Python interpreter can be removed by Python upgrades, putting the virtual environment into an unusable state. Thankfully, VirtualFish includes a mechanism for upgrading outdated/broken environments.

To understand which environments might be outdated/broken, run:

```
vf ls --details
```

You can maintain a list of target Python versions via a line such as the following in a `~/.tool-versions` file:

```
python 3.9.7 3.8.12 3.7.11 3.6.14
```

Environment Python versions that match one of those versions will be shown as up-to-date (green). If target Python versions are not specified in that file, VirtualFish compares environment Python versions to the current default Python version, as specified by the `VIRTUALFISH_DEFAULT_PYTHON` variable (see below), if defined. To perform a minor (point-release) upgrade to the currently-active virtual environment, run:

```
vf upgrade
```

Minor point-release upgrades will modify in-place the virtual environment's Python version number and symlinks. (While this should work correctly in the majority of cases, there is the possibility that future changes to virtual environment structure will interfere with this in-place upgrade.)

For major version upgrades, say from Python 3.8.x to 3.9.x, you must instead re-build the environment via:

```
vf upgrade --rebuild
```

Re-building an environment will record its current package versions, remove the old environment, create a new environment with the same name, and re-install the list of recorded package versions.

If VirtualFish determines that a virtual environment is in a broken state, it will re-build that environment, even if `--rebuild` is omitted.

To upgrade to a specific Python interpreter or version, use the `--python` option:

```
vf upgrade --rebuild --python /usr/local/bin/python3.8
```

Virtual environments need not be active in order to upgrade them. To upgrade one or more virtual environments, specify their names:

```
vf upgrade project1 project2
```

Upgrades can also be applied to all environments. To re-build all existing environments:

```
vf upgrade --rebuild --all
```

1.2.4 Configuration Variables

The `vf install [...]` installation step writes the VirtualFish loader to a file at `$XDG_CONFIG_HOME/fish/conf.d/virtualfish-loader.fish`, which on most systems defaults to: `~/.config/fish/conf.d/virtualfish-loader.fish`

You can edit this file to, for example, change the plugin loading order. You can also add the following optional variables at the top, so that they are set before `virtual.fish` is sourced.

- `VIRTUALFISH_HOME` (default: `~/.virtualenvs`) - where all your virtual environments are kept.
- `VIRTUALFISH_DEFAULT_PYTHON` - The default Python interpreter to use when creating a new virtual environment; the value should be a valid argument to the Virtualenv `--python` flag.

Regardless of the changes that you make, you must run `exec fish` afterward if you want those changes to take effect for the current shell session.

1.3 Plugins

VirtualFish comes with a number of built-in plugins.

You can use them by passing their names as arguments to the `vf install` command when installing for the first time. For example, the following will activate the `compat_aliases`, `projects`, and `environment` plugins:

```
vf install compat_aliases projects environment
```

To add or remove plugins after installation, use the `vf addplugins` and `vf rmplugins` commands. For example, the following will activate the `auto_activation` and `projects` plugins, and the subsequent command will remove the `projects` plugin:

```
vf addplugins auto_activation projects
vf rmplugins projects
```

1.3.1 Virtualenvwrapper Compatibility Aliases (`compat_aliases`)

This plugin provides some global commands to make VirtualFish behave more like Doug Hellman's virtualenvwrapper.

Commands

- `workon <envname>` = `vf activate <envname>`
- `deactivate` = `vf deactivate`
- `mkvirtualenv [<options>] <envname>` = `vf new [<options>] <envname>`
- `mktmpenv [<options>]` = `vf tmp [<options>]`
- `rmvirtualenv` = `vf rm <envname>`
- `lsvirtualenv` = `vf ls`
- `cdvirtualenv` = `vf cd`
- `cdsitepackages` = `vf cdpackages`
- `add2virtualenv` = `vf addpath`
- `allvirtualenv` = `vf all`
- `setvirtualenvproject` = `vf connect`

1.3.2 Auto-activation (`auto_activation`)

With this plugin enabled, VirtualFish can automatically activate a virtualenv when you are in a certain directory. To configure it to do so, change to the directory, activate the desired virtualenv, and run `vf connect`.

This will save the name of the virtualenv to a file named `.venv`. VirtualFish will then look for this file every time you `cd` into the directory (or `pushd`, or anything else that modifies `$PWD`).

Note: When this plugin is enabled, ensure any environment variables that affect VirtualFish are set as noted in *Upgrading Virtual Environments* and not in `config.fish`. Files in `~/.config/fish/conf.d/` (including VirtualFish) are sourced *before* `config.fish`, and thus variables set in `config.fish` may not be available to VirtualFish.

Commands

- `vf connect` - Connect the current virtualenv to the current directory, so that it is activated automatically as soon as you enter it (and deactivated as soon as you leave).

Configuration Variables

- `VIRTUALFISH_ACTIVATION_FILE` (default: `.venv`) - the name of the file VirtualFish will use for the auto-activation feature. Earlier versions of VirtualFish used `.vfenv`.

State Variables

- `VF_AUTO_ACTIVATED` - If the currently-activated virtualenv was activated automatically, set to the directory that triggered the activation. Otherwise unset.

1.3.3 Global Requirements (`global_requirements`)

Keeps a global `requirements.txt` file that is applied to every existing and new virtual environment. This behavior can be disabled for a given session by setting the `VIRTUALFISH_GLOBAL_REQUIREMENTS` environment variable to "0". To disable on a per-invocation basis, prefix commands with the same variable:

```
VIRTUALFISH_GLOBAL_REQUIREMENTS="0" vf tmp
```

Commands

- `vf requirements` - Edit the global requirements file in your `$EDITOR`. Applies the requirements to all virtualenvs on exit.

1.3.4 Projects (`projects`)

This plugin adds project management capabilities, including automatic directory switching upon virtual environment activation. Typically a project directory contains files — such as source code managed by a version control system — that are often stored separately from the virtual environment.

The following example will create a new project, with a matching virtual environment, both named `YourProject`:

```
vf project YourProject
```

The above command performs the following tasks:

1. creates new empty project directory in `PROJECT_HOME` (if there is no existing `YourProject` directory within) and changes the current working directory to it
2. creates new virtual environment named `YourProject` and activates it

To work on an existing project, use the `vf workon <name>` command to activate the specified virtual environment and change the current working directory to the project of the same name. For cases in which the project name differs from the target virtualenv name, you can manually specify which virtualenv should be activated for a given project by creating a `.venv` file inside the project root containing the name of the corresponding virtualenv.

If you use sub-folders, have projects located outside of `PROJECT_HOME`, or utilize a project organization strategy that does not lend itself to storing all your projects in the root of a single directory, you may navigate to your project and associate the current working directory with the currently-activated virtual environment via the following example steps:

```
vf activate YourVirtualenv
cd /path/to/your/project
echo $PWD > $VIRTUAL_ENV/.project
```

In the future, you may then run `vf workon YourVirtualenv` to simultaneously activate `YourVirtualenv` and switch to the `/path/to/your/project` directory.

Note: `.project` files are restored when calling `vf upgrade --rebuild`. If you are using both the *Compatibility Aliases* and *Projects* plugins, `workon` will alias `vf workon` instead of `vf activate`. If you are using both the *Auto-activation* and *Projects* plugins, the project's virtual environment will be deactivated automatically when you leave the project's directory.

Commands

- `vf project <virtualenv-options> <name>` - Create a new project and matching virtual environment with the specified name and Virtualenv options, including the ability to specify a Python interpreter via `--python`. If the `compat_aliases` plugin is enabled, `mkproject` is aliased to this command.
- `vf workon <name>` - Search for a project and/or virtualenv matching the specified name. If found, this activates the appropriate virtualenv and switches to the respective project directory. If the `compat_aliases` plugin is enabled, `workon` is aliased to this command.
- `vf lsprojects` - List projects available in `$PROJECT_HOME` (see below)
- `vf cdproject` - Search for a project matching the name of the currently activated virtualenv. If found, this switches to the respective project directory. If the `compat_aliases` plugin is enabled, `cdproject` is aliased to this command.

Configuration Variables

- `PROJECT_HOME` (default: `~/projects/`) - Where to create new projects and where to look for existing projects.

1.3.5 Environment Variables (`environment`)

This plugin provides the ability to automatically set environment variables when a virtual environment is activated. The environment variables are stored in a `.env` file by default. This can be configured by setting `VIRTUALFISH_ENVIRONMENT_FILE` to the desired file name. When using the *Projects (projects)* plugin, the env file is stored in the project directory unless it is manually created in the `$VIRTUAL_ENV` directory. If the projects plugin isn't being used, the file is stored in the `$VIRTUAL_ENV` directory.

When the virtualenv is activated, the values in the env file will be added to the environment. If a variable with that name already exists, that value is stored in `__VF_ENVIRONMENT_OLD_VALUE_$key`.

When the virtual environment is deactivated, if there was a pre-existing value it is returned to the environment. Otherwise, the variable is erased.

The format of the env file is one key-value set per line separated by an =. Empty lines are ignored, as are any lines that start with #. See the following:

```
# This is a valid comment and declaration
FOO=bar

# The empty line above is valid
BAR=baz  # Inline comments like this one are NOT okay
```

Commands

- `vf environment` - Open the environment file for the active virtual environment in `$VISUAL`/`$EDITOR`, or `vi` if neither variable is set.

1.3.6 Update Python (`update_python`)

Note: The functionality provided by this plugin has been superseded by the `vf upgrade` command. This plugin has therefore been deprecated and will likely be removed in the future.

This plugin adds commands to change the Python interpreter of the current virtual environment.

Commands

- `vf update_python [<python_exe>]` - Remove the current virtual environment and create a new one with `<python_exe>` (defaults to `VIRTUALFISH_DEFAULT_PYTHON` if it is set, or the first executable named `python` in your `PATH`), and then re-install the same versions of all packages with Pip.
- `vf fix_python [<python_exe>]` - Test the current virtual environment's Python executable. If it doesn't work, update it with `vf update_python [<python_exe>]`. This may be useful when one of your system's Python executables is updated, which may break some of your virtual environments. In that case, you probably just need to run: `vf all vf fix_python`

Configuration Variables

- `VIRTUALFISH_DEFAULT_PYTHON` (default: `python`) - The Python interpreter to use if not specified as an argument to the above commands.

1.4 Extending VirtualFish

1.4.1 Variables

Virtualenv currently provides one global variable to allow you to inspect its state. (Keep in mind that more are provided by VirtualFish plugins.)

- `VIRTUAL_ENV` - Path to the currently active virtualenv.
  - Tips: use `basename` to get the virtualenv's name, or `set -q` to see whether a virtualenv is active at all.

1.4.2 Events

VirtualFish emits Fish events instead of using hook scripts. To hook into events that VirtualFish emits, write a function like this:

```
function myfunc --on-event virtualenv_did_activate
    echo "The virtualenv" (basename $VIRTUAL_ENV) "was activated"
end
```

You can save your function by putting it in `.config/fish/config.fish`, or put it anywhere Fish will see it before it needs to run. (Note: saving it with `funcsave` won't work.)

Some events are emitted twice: once normally and once with the name of the virtualenv as part of the event name. This is to make it easier to listen for events relevant to one specific virtualenv, for example:

```
function myfunc --on-event virtualenv_did_activate:my_site_env
    set -gx DJANGO_SETTINGS_MODULE mysite.settings
end
```

The full list of events is:

- virtualenv_did_setup_plugins
- virtualenv_will_activate
- virtualenv_will_activate:<env name>
- virtualenv_did_activate
- virtualenv_did_activate:<env name>
- virtualenv_will_deactivate
- virtualenv_will_deactivate:<env name>
- virtualenv_did_deactivate
- virtualenv_did_deactivate:<env name>
- virtualenv_will_create
- virtualenv_did_create
- virtualenv_did_create:<env name>
- virtualenv_will_upgrade
- virtualenv_will_upgrade:<env name>
- virtualenv_did_upgrade
- virtualenv_did_upgrade:<env name>
- virtualenv_will_remove
- virtualenv_will_remove:<env name>
- virtualenv_did_remove
- virtualenv_did_remove:<env name>
- virtualenv_did_connect
- virtualenv_did_connect:<env name>

1.5 Frequently Asked Questions

1.5.1 How do I ensure new environments always have the latest version of Pip?

You may see warnings from Pip about a newer available version, even on fresh environments you have just created. To ensure Pip is automatically updated upon environment creation, enable the *Global Requirements* plugin and add Pip via:

```
vf addplugins global_requirements
echo "pip" >> $VIRTUALFISH_HOME/global_requirements.txt
```

1.5.2 Why isn't VirtualFish written in Python?

Mostly, for the same reasons Virtualenvwrapper isn't.

1.5.3 Does VirtualFish work with Python 3? What about PyPy?

Yes! In fact, you can create Python 3 virtual environments even if your system Python is Python 2, or vice versa, using the `--python` argument (see the *Usage* section for full details).

1.5.4 Why does VirtualFish use Virtualenv and not Python's built-in `venv` module?

Virtualenv can create both Python 2 and Python 3 virtual environments, whereas Python's built-in `venv` module can only create Python 3 virtual environments. That said, since Python 2 is no longer officially supported by the Python Software Foundation, Python 2 support is a very minor consideration when deciding which tool to use. The main reason VirtualFish uses Virtualenv is due to its **much** faster speed. We have seen Virtualenv create environments in **one-fifth** the amount of time that the `venv` module takes to perform the same task.

1.5.5 Why doesn't VirtualFish use activate.fish?

VirtualFish uses its own internal virtual environment activation code instead of the `activate.fish` file that Virtualenv generates for two main reasons. One is that when VirtualFish was originally written, `activate.fish` didn't actually work. The second reason, which is still valid today, is that `activate.fish` tries to modify your `fish_prompt` function.

Because `fish_prompt` is a function and not a variable like in most other shells, modifying it programmatically is not trivial, and the way that Virtualenv accomplishes it is more than a little hacky. The benefit of it being a function is that the syntax for customising it is much less terse and cryptic than, say, `PS1` on Bash. This is why VirtualFish doesn't attempt to modify your prompt, and instead tells you how to do it yourself.

1.6 See Also

This page is for other projects that integrate with VirtualFish, such as third-party plugins, prompts, and so on. If you know of (or maintain!) such a project, and it's not on this list, please submit a pull request.

1.6.1 Prompts

- Both `bobthefish` and `scorphish` themes for Oh My Fish! support VirtualFish.

CHAPTER 2

Contributors

Sorted by date of first commit:

- Leigh Brenecki
- Justin Mayer
- David Reid
- Alex Gaynor
- Álvaro Lázaro Gallego
- Jan Kasiak
- David Adam
- Robson Roberto Souza Peixoto
- Casey Chance
- fenekku
- Trung Ly
- George Christou