virtualfish Documentation

Release 1.0.6.dev14+g17071a4.d20170908

Adam Brenecki and contributors

Sep 08, 2017

Contents

A Fish Shell wrapper for Ian Bicking's virtualenv, somewhat loosely based on Doug Hellman's virtualenvwrapper for Bourne-compatible shells.

CHAPTER 1

Contents

Installation and Setup

Installing

1. Make sure you're running Fish 2.x. If you're running an Ubuntu LTS release that has an older Fish version, install Fish 2.x via the fish-shell/release-2 PPA.
2. Install virtualfish by running `pip install virtualfish`.
3. Add the following to your `~/.config/fish/config.fish`:

   ```
   eval (python -m virtualfish)
   ```

   If you want to use virtualfish with *plugins*, list the names of the plugins as arguments to the virtualfish loader:

   ```
   eval (python -m virtualfish compat_aliases)
   ```

   Note: If your `config.fish` modifies your `$PATH`, you should ensure that you load virtualfish *after* those modifications.
4. Customize your `fish_prompt`

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 virtualenv to your prompt is to type `funced fish_prompt`, add the following line in 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.

Usage

Commands

- `vf new [<options>] <envname>` - Create a virtualenv. Note that `<envname>` *must be last*.
- `vf ls` - List the available virtualenvs.
- `vf activate <envname>` - Activate a virtualenv. (Note: Doesn't use the `activate.fish` script provided by virtualenv.)
- `vf deactivate` - Deactivate the current virtualenv.
- `vf rm <envname>` - Delete a virtualenv.
- `vf tmp [<options>]` - Create a temporary virtualenv with a randomly generated name that will be removed when it is deactivated.
- `vf cd` - Change directory to currently-activated virtualenv.
- `vf cdpackages` - Change directory to the currently-activated virtualenv's site-packages.
- `vf addpath` - Add a directory to this virtualenv's `sys.path`.
- `vf all <command>` - Run a command in all virtualenvs sequentially.
- `vf connect` - Connect the current working directory with the currently active virtualenv. 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're used to virtualenvwrapper's commands (`workon`, etc.), you may wish to enable the *Virtualenvwrapper Compatibility Aliases (compat_aliases)* plugin.

Using Different Pythons

By default, the environments you create with `virtualenv` (and, by extension, virtualfish) use the same Python version that `virtualenv` was installed under, which will usually be whatever your default system Python is.

If you want to use something different in a particular virtualenv, just pass in the `--python PYTHON_EXE` (`-p` for brevity) argument to `vf new`, where `PYTHON_EXE` is any Python executable, for example:

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

Configuration Variables

All of these must be set before `virtual.fish` is sourced in your `~/.config/fish/config.fish`.

- `VIRTUALFISH_HOME` (default: `~/.virtualenvs`) - where all your virtualenvs are kept.

- `VIRTUALFISH_DEFAULT_PYTHON` - The default Python interpreter to use when creating a new virtualenv; the value should be a valid argument to `virtualenv`'s `--python` flag.

Plugins

Virtualfish comes in-built with a number of plugins.

You can use them by passing their names in as arguments to the virtualfish loader in your `config.fish`, e.g.:

```
eval (python -m virtualfish auto_activation global_requirements)
```

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`

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 the file `.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 modifications to your `$PATH` in your `config.fish` happen before virtualfish is loaded.

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.

Global Requirements (`global_requirements`)

Keeps a global `requirements.txt` file that is applied to every existing and new virtualenv.

Commands

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

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.

Commands

- `vf project <name>` - Create a new project and matching virtual environment with the specified name. This name **must** be the last parameter (i.e., after `-p python3` or any other arguments destined for the `virtualenv` command). If `VIRTUALFISH_COMPAT_ALIASES` is set, `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 `VIRTUALFISH_COMPAT_ALIASES` is set, `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 `VIRTUALFISH_COMPAT_ALIASES` is set, `cdproject` is aliased to this command.

Configuration Variables

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

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  # A following comment like this is NOT okay
```

Commands

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

Extending Virtualfish

Variables

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

Events

virtualfish emits Fish events instead of using hook scripts. To hook in to 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>`

Frequently Asked Questions

Why isn't Virtualfish written in Python?

Mostly, for the same reasons virtualenvwrapper is.

Does Virtualfish work with Python 3? What about PyPy?

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

Why does Virtualfish use Virtualenv and not pyvenv?

pyvenv may be the new shiny, but it can only be run from Python 3 and can only create Python 3 environments. In contrast, virtualenv fully supports Python 2 and 3, as discussed above. So, we can't use pyvenv on its own.

It's been suggested that we could use both, but that would add complexity for no real benefit. If pyvenv added new, broadly useful features not in virtualenv, or if virtualenv stopped working on Python 3, or if Python 2 went out of widespread use, this might change, but for now virtualenv is the best choice.

Why doesn't Virtualfish use activate.fish?

Virtualfish uses its own internal virtualenv activation code instead of the `activate.fish` file that comes with every virtualenv 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.

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 file a pull request.

Prompts

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

CHAPTER 2

Contributors

Sorted by date of first commit:

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