User Guide

Usage

Virtualenv has one basic command:

```
$ virtualenv ENV
```

Where `ENV` is a directory to place the new virtual environment. It has a number of usual effects (modifiable by many Options):

- `ENV/lib/` and `ENV/include/` are created, containing supporting library files for a new virtualenv python. Packages installed in this environment will live under `ENV/lib/pythonX.X/site-packages/`.
- `ENV/bin` is created, where executables live - noticeably a new **python**. Thus running a script with `#! /path/to/ENV/bin/python` would run that script under this virtualenv's python.
- The crucial packages pip and setuptools are installed, which allow other packages to be easily installed to the environment. This associated pip can be run from `ENV/bin/pip`.

The python in your new virtualenv is effectively isolated from the python that was used to create it.

activate script

In a newly created virtualenv there will also be a **activate** shell script. For Windows systems, activation scripts are provided for the Command Prompt and Powershell.

On Posix systems, this resides in `/ENV/bin/`, so you can run:

```
$ source bin/activate
```

For some shells (e.g. the original Bourne Shell) you may need to use the **.** command, when **source** does not exist. There are also separate activate files for some other shells, like csh and fish. `bin/activate` should work for bash/zsh/dash.

This will change your `$PATH` so its first entry is the virtualenv's `bin/` directory. (You have to use `source` because it changes your shell environment in-place.) This is all it does; it's purely a convenience. If you directly run a script or the python interpreter from the virtualenv's `bin/` directory (e.g. `path/to/ENV/bin/pip` or `/path/to/ENV/bin/python-script.py`) there's no need for activation.

The `activate` script will also modify your shell prompt to indicate which environment is currently active. To disable this behaviour, see `VIRTUAL_ENV_DISABLE_PROMPT`.

To undo these changes to your path (and prompt), just run:

```
$ deactivate
```

On Windows, the equivalent *activate* script is in the `Scripts` folder:

```
> \path\to\env\Scripts\activate
```

And type `deactivate` to undo the changes.

Based on your active shell (CMD.exe or Powershell.exe), Windows will use either activate.bat or activate.ps1 (as appropriate) to activate the virtual environment. If using Powershell, see the notes about code signing below.

Note

If using Powershell, the `activate` script is subject to the execution policies on the system. By default on Windows 7, the system's excution policy is set to `Restricted`, meaning no scripts like the `activate` script are allowed to be executed. But that can't stop us from changing that slightly to allow it to be executed.

In order to use the script, you can relax your system's execution policy to `AllSigned`, meaning all scripts on the system must be digitally signed to be executed. Since the virtualenv activation script is signed by one of the authors (Jannis Leidel) this level of the execution policy suffices. As an administrator run:

```
PS C:\> Set-ExecutionPolicy AllSigned
```

Then you'll be asked to trust the signer, when executing the script. You will be prompted with the following:

```
PS C:\> virtualenv .\foo
New python executable in C:\foo\Scripts\python.exe
Installing setuptools.............done.
Installing pip...................done.
PS C:\> .\foo\scripts\activate

Do you want to run software from this untrusted publisher?
File C:\foo\scripts\activate.ps1 is published by E=jannis@leidel.info,
CN=Jannis Leidel, L=Berlin, S=Berlin, C=DE, Description=581796-Gh7xfJxkxQSIO4E0
and is not trusted on your system. Only run scripts from trusted publishers.
[V] Never run  [D] Do not run  [R] Run once  [A] Always run  [?] Help
(default is "D"):A
(foo) PS C:\>
```

If you select `[A] Always Run`, the certificate will be added to the Trusted Publishers of your user account, and will be trusted in this user's context henceforth. If you select `[R] Run Once`, the script will be run, but you will be prompted on a subsequent invocation. Advanced users can add the signer's certificate to the Trusted Publishers of the Computer account to apply to all users (though this technique is out of scope of this document).

Alternatively, you may relax the system execution policy to allow running of local scripts without verifying the code signature using the following:

```
PS C:\> Set-ExecutionPolicy RemoteSigned
```

Since the `activate.ps1` script is generated locally for each virtualenv, it is not considered a remote script and can then be executed.

Removing an Environment

Removing a virtual environment is simply done by deactivating it and deleting the environment folder with all its contents:

```
(ENV)$ deactivate
$ rm -r /path/to/ENV
```

The `--system-site-packages` Option

If you build with `virtualenv --system-site-packages ENV`, your virtual environment will inherit packages from `/usr/lib/python2.7/site-packages` (or wherever your global site-packages directory is).

This can be used if you have control over the global site-packages directory, and you want to depend on the packages there. If you want isolation from the global system, do not use this flag.

Windows Notes

Some paths within the virtualenv are slightly different on Windows: scripts and executables on Windows go in `ENV\Scripts\` instead of `ENV/bin/` and libraries go in `ENV\Lib\` rather than `ENV/lib/`.

To create a virtualenv under a path with spaces in it on Windows, you'll need the win32api library installed.

Using Virtualenv without `bin/python`

Sometimes you can't or don't want to use the Python interpreter created by the virtualenv. For instance, in a mod_python or mod_wsgi environment, there is only one interpreter.

Luckily, it's easy. You must use the custom Python interpreter to *install* libraries. But to *use* libraries, you just have to be sure the path is correct. A script is available to correct the path. You can setup the environment like:

```
activate_this = '/path/to/env/bin/activate_this.py'
execfile(activate_this, dict(__file__=activate_this))
```

This will change `sys.path` and even change `sys.prefix`, but also allow you to use an existing interpreter. Items in your environment will show up first on `sys.path`, before global items. However, global items will always be accessible (as if the `--system-site-packages` flag had been used in creating the environment, whether it was or not). Also, this cannot undo the activation of other environments, or modules that have been imported. You shouldn't try to, for instance, activate an environment before a web request; you should activate *one* environment as early as possible, and not do it again in that process.

Making Environments Relocatable

Note: this option is somewhat experimental, and there are probably caveats that have not yet been identified.

> **Warning**
>
> The `--relocatable` option currently has a number of issues, and is not guaranteed to work in all circumstances. It is possible that the option will be deprecated in a future version of `virtualenv`.

Normally environments are tied to a specific path. That means that you cannot move an environment around or copy it to another computer. You can fix up an environment to make it relocatable with the command:

```
$ virtualenv --relocatable ENV
```

This will make some of the files created by setuptools use relative paths, and will change all the scripts to use `activate_this.py` instead of using the location of the Python interpreter to select the environment.

Note: scripts which have been made relocatable will only work if the virtualenv is activated, specifically the python executable from the virtualenv must be the first one on the system PATH. Also note that the activate scripts are not currently made relocatable by `virtualenv --relocatable`.

Note: you must run this after you've installed *any* packages into the environment. If you make an environment relocatable, then install a new package, you must run `virtualenv --relocatable` again.

Also, this **does not make your packages cross-platform**. You can move the directory around, but it can only be used on other similar computers. Some known environmental differences that can cause incompatibilities: a different version of Python, when one platform uses UCS2 for its internal unicode representation and another uses UCS4 (a compile-time option), obvious platform changes like Windows vs. Linux, or Intel vs. ARM, and if you have libraries that bind to C libraries on the system, if those C libraries are located somewhere different (either different versions, or a different filesystem layout).

If you use this flag to create an environment, currently, the `--system-site-packages` option will be implied.

The `--extra-search-dir` option

This option allows you to provide your own versions of setuptools and/or pip to use instead of the embedded versions that come with virtualenv.

To use this feature, pass one or more `--extra-search-dir` options to virtualenv like this:

```
$ virtualenv --extra-search-dir=/path/to/distributions ENV
```

The `/path/to/distributions` path should point to a directory that contains setuptools and/or pip wheels.

virtualenv will look for wheels in the specified directories, but will use pip's standard algorithm for selecting the wheel to install, which looks for the latest compatible wheel.

As well as the extra directories, the search order includes:

1. The `virtualenv_support` directory relative to virtualenv.py
2. The directory where virtualenv.py is located.
3. The current directory.