

Python Virtual Environments

— A gentle introduction for Linux users —

Cedric Bhihe ©
Last update: Jun. 2022

Foreword

This tutorial was inspired by two posts: the first (2019) was a [blog post](#) by Bartek Skorulski¹, the second (2017) was a [post](#) by @Flimm on SO. The material presented goes beyond what is usually served on the Internet, mostly disjoint commentaries, which many times can truly be understood only by people who already are experienced Python virtual environment practitioners. All hand-on examples shown were run on the Debian flavored Linux environment commonly seen by Arch Linux users, and by extension by all Linux users whose distribution runs on the system and service initialization manager²: [systemd](#). CLI snippets generally applies to the Bash shell but can be adapted with minimal changes to most older, POSIX compliant shells. The tutorial is meant to address the down-to-earth needs of both first-time users and users who want to make sense of the current Python virtual environment (VE) landscape for Linux, as of mid 2020.

But really, what is a Python virtual environment? What could motivate the creation of VEs for Python? I answer those two questions superficially in the next paragraph and choose to dedicate your reading time to how one may go about creating and managing them.

If you use Python for anything, either professionally or at home, chances are you already ran into snags related to the need for different versions of the same Python module(s). If you have not yet contended with such predicament, the premise to this tutorial is that sooner or later you will. This tutorial is a step-by-step guide on how to overcome such a difficulty by using Python virtual environments. There are several ways to do so and you may choose the one that appears more practical to you.

At its root the issue for any programmer is that different versions of any Python module normally cannot coexist in a single runtime environment (RTE) and name space. If you did not already tweak your python setup, you may at most have the latest versions of Python2 (v. 2.7.16) and some version of Python3 installed side by side in your environment. But what if you also needed versions (say 3.7.4 *and* 3.10.0 *and* 2.7.1 *and* 2.5.0) for specific projects involving older libraries ? Forcefully installing any older Python version on your platform would normally require you to rebuild older packages on a massive scale, as well as to roll back your system's module versions and many of its applications. It can be done but it would be extraordinarily complicated, not least because it would be very time-consuming and prone to breakage. As for installing several such versions side by side, simply forget it, unless you enjoy [breaking things](#). So how does one remain practical ? By embracing the concept of *Python virtual environments*.

VEs are isolated Python environments, in as many specific directories. In each VE you can install and execute a Python version of your choice, independently of other Python VEs already residing on your platform. The relevant [Arch wiki](#) explains:

A virtual environment is a directory in which some binaries and shell scripts are installed. The binaries include python for executing scripts and pip for installing other modules within the environment. There are also shell scripts (at least one for Bash, ...) to activate the environment. Essentially, a virtual environment mimics a full system install of Python and all of the desired modules without interfering with any system on which the application might run.

-
- 1 Data Scientist at Koa Health, a spin-off of the Telefónica Alpha incubator in Barcelona, Spain.
 - 2 A simplified overview of the entire Linux boot and startup process is: (i) At host's power-up, the BIOS does minimal hardware tests and initialization. It then hands control over to the boot loader. (ii) The boot loader calls the kernel. (iii) The kernel loads an initial RAM disk that loads the system drives and then looks for the root file system. (iv) After kernel set up, *systemd* initialization starts. (v) Finally *systemd* takes over and continues to mount the host's file systems and start services.

Copyright and licensing terms

Copyright © Cedric Bhihe, 2019-2022

under the terms of the [GNU Free Documentation License \(Version 1.3 or later\)](#),

Permission is granted to copy, distribute and/or modify this document with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in Appendix B, entitled "GNU Free Documentation License".

List of abbreviations

Bash	Bourne again shell
CLI	command line interface
GNU	GNU is not Unix
iPython	interactive Python
RAM	random access memory
RTE	runtime environment
SO	stackoverflow.com
VE	virtual environment

Table of Contents

Foreword.....	1
Copyright and licensing terms.....	2
List of abbreviations.....	2
1. Flavors of VEs and ways to set them up.....	4
1-1. 'venv'.....	4
1-1-1. Virtual environment activation.....	5
1-1-2. Caveat: changes in executable paths.....	6
1-1-3. Other limitations.....	7
1-2. 'virtualenv'.....	7
1-2-1. Installation.....	7
1-2-2. Usage.....	7
1-2-3. 'virtualenv' command options.....	8
1-2-4. Activating the virtual environment.....	9
1-2-5. NOT activating the virtual environment.....	9
1-2-6. Deactivating and removing the virtual environment.....	10
1-2-7. 'virtualenvwrapper' (plugin).....	10
1-2-7-1. Features.....	10
1-2-7-2. Setup.....	11
1-2-7-3. Usage.....	11
1-3. 'pyenv'.....	13
1-3-1. Working principle.....	13
1-3-1-1. Scope.....	13
1-3-1-2. Plugins: 'pyenv-virtualenv' and 'pyenv-virtualenvwrapper'.....	14
1-3-2. Installation of 'pyenv' (pyenv-git).....	14
1-3-3. Configuration and update of 'pyenv'.....	14
1-3-4. Uninstalling 'pyenv'.....	16
1-3-5. Installation of 'pyenv-virtualenv'.....	16
2. Using Python VEs.....	17
2-1. 'pyenv' usage.....	17
2-1-1. Selection of the VE's Python version.....	17
2-1-2. Activation of the Python VE.....	18
2-1-3. Removing a Python VE.....	19
2-2. 'pyenv-virtualenv' usage with 'pyenv'.....	19
2-2-1. Creation of Python VEs.....	19
2-2-2. Listing of Python VEs.....	19
2-2-3. Automatic / manual virtual environments activation / deactivation.....	20
2-2-4. Deletion of an existing virtual environments.....	20
2-3. Working with shell-specific Python versions.....	20
2-4. Run a Jupyter notebook in a Python virtual environment.....	21
References.....	22
Appendix A: Work with 'virtualenvwrapper'.....	23
Appendix B: GNU Free Documentation License.....	25

1. Flavors of VEs and ways to set them up

In this tutorial only 2 closely related VE flavors are described: ‘[venv](#)’ and ‘[virtualenv](#)’. In addition I introduce a handy plugin used by many: ‘[virtualenvwrapper](#)’. Finally we will see three ways to go about setting up virtual environments on a Linux OS:

- [venv](#)

The **Venv** module’s native capability is already built-in in Linux system running Python v3.3+. It is very lightweight and you need not install any new package. It allows testing your code against your OS’ Python version, in a way that isolates the installation of any Python packages and their dependencies from those of your system.

- [virtualenv](#)

Virtualenv serves the same purpose as Venv, but has more features ([see a comparison here](#)). The corresponding Arch linux users’ package is named [python-virtualenv](#). It continues to be more popular than Venv. This is because Virtualenv (unlike Venv) supports both Python 2 and 3.

- [pyenv](#)

Pyenv is also used to isolate Python versions. To Virtualenv users (package: ‘[python-virtualenv](#)’), the idea of a Python version local to a directory might seem familiar. A *local* Python created from [pyenv](#) is also a virtual environment. The main difference is that Pyenv actually copies an entire Python installation every time you install a new Python version. In contrast, Virtualenv (not described in this Tutorial) makes use of symbolic links to decrease the size of the virtual environment.

Testing against the *current* Python versions 2 and 3 in a sandbox would only require Virtualenv. However testing against any sand-boxed Python versions (e.g. 2.7, 3.6, 3.7 and 3.9), requires a practical way to switch between them. Meet Pyenv ! And if you were not confused yet, Pyenv boasts *two main plugins*, which extend its command set:

- ***Pyenv-virtualenv*** (under the package name [pyenv-virtualenv](#))

It integrates nicely in Pyenv and makes possible creating and handling native Pyenv’s VEs.

- ***Virtualenvwrapper*** (under the package name [pyenv-virtualenvwrapper](#))

It helps interacting with the plugin Virtualenvwrapper itself.

Only the first of the two plugins, namely Pyenv-virtualenv, is described because I found that it covers all my needs and is lightweight enough that one does not have to remember overwrought command syntax.

1-1. ‘venv’

As already mentioned, ‘[venv](#)’ is a native capability of Python v3.3+. You can run it using the command

```
$ /usr/bin/python3 -m venv
```

(although for some reason some distros separate it out into a separate distro package, such as ‘python3-venv’, e.g. on Ubuntu/Debian).

1-1-1. Virtual environment activation

To set it up along with pip (included by default since Python 3.4+), just issue:

```
$ python -m venv [--system-site-packages] --prompt VENV <dir1> \
[<dir2> [<dir3> [...]]]
```

where:

- `-m venv` will *create* the VE
- `--system-site-packages` optionally gives the VE access to the system site packages
- `--prompt VENV` will modify the prompt by prepending (`VENV`) to it, so you know when you operate in

an activated VE or not.

- `<dir1>`, and optional `<dir2>`, ... are fully qualified target directories, created if they do not exist already. Inside each one of them them, a VE will have its own Python binary, *which matches the version of the Python binary used to create the VE*. This is a crucial restriction. If the Python binary of your calling environment has version 3.7.x, so will the Python binary of your VE under `venv`. Within that restriction, each `venv` VE can have its own set of Python packages installed independently from one another , but all based on the same Python binary version.

The created `pyvenv.cfg` file also includes the `include-system-site-packages` key, set to `true` if `venv` is run with the `--system-site-packages` option, `false` otherwise.

Unless the `--without-pip` option is given, `ensurepip` will be invoked by default in order to bootstrap `pip` into the VE.

If the `--without-pip` option was given, but `pip` needs to be installed *a posteriori*, you can always issue:

```
$ python -m ensurepip -upgrade
```

The `--upgrade` option will ensure that the version of `pip` installed is at least as new as the existing one if any. To know all the available options, issue:

```
$ python -m venv -h
```

To activate and deactivate the Venv Python VE, issue:

```
<pwd> $ source <dir>/bin/activate
(VENV) <dir> $ do your thing here...
(VENV) <dir> $ deactivate
```

where `<dir>` is any previously specified target directory where the Venv VE is to be deployed.

1-1-2. Caveat: changes in executable paths

Quoting for the [Python3 docs](#):

When a VE is active (i.e., the VE's Python interpreter is running), the attributes `sys.prefix` and `sys.exec_prefix` point to the base directory of the VE, whereas `sys.base_prefix` and `sys.base_exec_prefix` point to the non-VE Python installation which was used to create the VE. If a VE is not active, then `sys.prefix` is the same as `sys.base_prefix` and `sys.exec_prefix` is the same as `sys.base_exec_prefix` (they all point to a non-VE Python installation).

When a VE is active, any options that change the installation path will be ignored from all `distutils` configuration files to prevent projects being inadvertently installed outside of the VE.

When working in a command shell, users can make a VE active by running an `activate` script in the VE's directory (the precise filename is shell-dependent). This prepends the VE's directory for executables to the `PATH` environment variable for the running shell. There should be no need in other circumstances to activate a VE—scripts installed into VEs have a “shebang” line which points to the VE's Python interpreter. This means that the script will run with that interpreter regardless of the value of `PATH`.

An immediate consequence of the above mentioned changes in [path variables](#) manifests itself, for example, when running a Jupyter notebook. There are two possibilities to run such a notebook:

a) It is run from the global (not isolated, non virtual) environment:

In such a case its Python v3.x.y kernel will be based on `/usr/bin/python3`. Hence all package imported in the notebook should be installed in that global `/usr/bin/python3` environment prior to importing them in the notebook.

To do so, first install ‘`pip3`’ globally:

```
$ /usr/bin/python3 -m ensurepip --user --upgrade
```

followed by, e.g. for the ‘seaborn’ package:

```
$ /usr/bin/python3 -m pip install seaborn
```

Only then can ‘`seaborn`’ be imported from a Jupyter notebook.

b) it is run explicitly from the activated Python VE

In that case we revert to the usual package installation syntax, where the command ‘`python3`’ refers to the local VE Python.

```
(ENV) $ python3 -m pip install seaborn
```

1-1-3. Other limitations

As already mentioned the functionalities of `'venv'` are a subset of what `'virtualenv'` (described in the next subsection) does. That subset of tools was integrated upstream into the standard libraries under the [venv module](#). Note however that the `'venv'` module does not offer all features of `'virtualenv'`. For instance it is not:

- able to create bootstrap scripts,
- able to create VEs for Python versions other than that of the host's base Python,
- relocatable,
- ...

For all those reasons the [venv module](#) bundled with Python versions 3.3 and later is recommended for projects that no longer need to support Python2 and just require straightforward isolated environments, always based on the host's Python version.

Venv VE isolation and independence is only meant in the sense of installed package *availability*, **not** in the sense of the arbitrariness of installed package *versions*.

1-2. 'virtualenv'

On many Linux distros, Ian Bicking's [virtualenv](#) along with its plugin, [virtualenvwrapper](#), described later in subsection 1.2.7, are used extensively, for both development and deployment.

While [virtualenv](#) is a tool to create VEs, [virtualenvwrapper](#) is a plugin, i.e. a set of extensions for [virtualenv](#). Unlike [venv](#), [virtualenv](#) creates VEs where libraries are not shared with those of other VEs. Optionally they also can be denied access to the globally installed libraries.

1-2-1. Installation

```
$ sudo pacman -Syu python-virtualenv
```

after which *man* pages may be consulted in the usual manner:

```
$ man virtualenv
```

1-2-2. Usage

Quoting largely from the Arch Linux documentation, [virtualenv](#) has one basic command:

```
$ virtualenv [--python=PYTHON_EXE [options]] <dir>
```

where `<dir>` is a fully qualified directory, which is to house the new Python's VE. `<dir>` will also be the new prompt's default prefix, the which can be modified easily with the `--prompt` option.

The newly created VE will be placed in `<dir>`. By default it will not rely on site-packages. It is modifiable via many [options](#) and has the following effects:

- `<dir>/lib/` and `<dir>/include/` are created, containing supporting library files for a new Python's VE. Packages installed in this VE will live under `<dir>/lib/pythonX.X/site-packages/`.
- `<dir>/bin/` is where executable binary files live, among them a new Python binary, whose version depends on the option `--python=PYTHON_EXE`. Thus, running a script with shebang
`#!/<dir>/bin/python`
would run that script under this very virtualenv's python's binary version.

1-2-3. 'virtualenv' command options

You may modify the build of your VE with options.

- Options `--no-pip` and `--no-setuptools`
When used, preclude installation of the two crucial packages: [pip](#) and [setuptools](#). Otherwise those are installed by default, which allows other packages to be further installed in the VE. This associated pip can be run with `<dir>/bin/pip`. Thus the Python version in your new VE is effectively isolated from the Python used to create it.
- Option `-p PYTHON_EXE`, or equivalently `--python=PYTHON_EXE`
With this you may specify the Python interpreter to use to create the new environment, e.g.:
`-p python2.5`
or equivalently
`--python=python2.5`.
In absence of that option, the default interpreter is that with which *virtualenv* was installed. As of 2017, it would most likely be `/usr/bin/python`, symbolic-linked to the system's version of Python 3.x.
- Option `--system-site-packages`
That makes your VE inherit packages from wherever your global site-packages directory is located, e.g.:
`/usr/lib/python2.7/site-packages`.

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 complete isolation from the global system, in particular if you want to prevent a system update to wreak havoc in your carefully isolated VE's packages and dependencies' versions, do **not** use this flag.

If you need to change this option after creating a virtual environment, you can turn it off or on. You do so by respectively adding or deleting the file `no-global-site-packages.txt` to or from the VE directory: `<dir>/lib/python3.7/`

- Option `--prompt MY_PROMPT`
modifies the activated VE's prompt's prefix.
`MY_PROMPT` is usually much shorter and more meaningful than its default `<dir>` value.
- Other slightly less common but handy options include:
`--relocatable`
`--extra-search-dir=/path/to/distributions`

They are briefly described in the man page as well as in the online [virtualenv user guide](#).

1-2-4. Activating the virtual environment

In a newly created VE there will also be an `activate` shell script. The exact `activate` file location may vary with the shell being used (`csh`, `fish`, ...). On POSIX and POSIX-like systems, and in particular under for `bash`, `zsh` and `dash`, the activation script resides in `<dir>/bin/`, so you can run:

```
$ source <dir>/bin/activate
```

For some shells (e.g. the original Bourne Shell), or `source` does not exist, you may need to use the `.'` cmd. Cmd `source` is purely a convenience in that it makes your shell environment change *in-place*. All that activation of the VE does is to change the `PATH` variable, so its first entry becomes the VE's `<dir>/bin/` directory.

The `activate` script will also modify your shell prompt to indicate which environment is currently active. To disable that behaviour, see [VIRTUAL_ENV_DISABLE_PROMPT](#).

1-2-5. NOT activating the virtual environment

If you choose NOT to activate your VE, and instead run a script or the python interpreter directly from the VE's directory, `<dir>/bin/`, e.g. by doing:

```
<dir>/bin/pip or  
<dir>/bin/python-script.py
```

then `sys.path` will automatically be set to use the Python libraries associated with the VE. Bear in mind however, that in that case, unlike what we just saw in the previous section with activation scripts, the environment variables `PATH` and `VIRTUAL_ENV` will *not* be modified. This means that if for example a Python script uses `subprocess` to run another Python script, for instance via the shebang `#!/usr/bin/env python`, the second script *may neither be executed with the same Python binary as the first*, nor see the same libraries. To prevent this from happening your first script will need to modify the environment variables in the same manner as the activation scripts, before the second script is executed.

1-2-6. Deactivating and removing the virtual environment

To undo these changes to PATH (and prompt), and to return to your normal global Python's default environment just run:

```
(<dir>) $ deactivate
```

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

```
(<dir>)$ deactivate  
$ cd .. && \rm -r <dir>
```

1-2-7. 'virtualenvwrapper' (plugin)

This MIT licensed plugin by Doug Hellmann is a set of extensions to virtualenv (see [docs](#)). It gives you commands such as:

```
mkvirtualenv  
mktmpenv  
lsvirtualenv  
lssitepackages  
showvirtualenv  
rmvirtualenv  
allvirtualenv  
mkproject  
workon
```

and more... The last cited one above is also the most used and allow the user to switch between different virtualenv directories and their specific VEs. This tool is especially useful if you usually work with multiple virtualenv directories simultaneously, i.e. when your workflow makes you jump from one to the other.

1-2-7-1. Features

The extensions include wrappers for creating and deleting VEs and otherwise managing your development workflow. It makes it easier to work on more than one project at a time, without introducing conflicts in their dependencies.

1. Organizes all of your VEs in one place.
2. Wrappers for managing your VEs (create, delete, copy).
3. Use a single cmd to switch between VEs.
4. Tab completion for cmds that take a VE as argument.
5. User-configurable hooks for all operations (see [Per-User Customization](#)).
6. Plugin system for more sharable extensions (see [Extending virtualenvwrapper](#)).

1-2-7-2. Setup

virtualenvwrapper should be installed in the same global-site packages area as **virtualenv**, in the absence of any active VE, so the same release is shared by all Python environment on your platform.

```
$ sudo pacman -Syu python-virtualenvwrapper
```

or

```
$ /usr/bin/python -m pip install virtualenvwrapper
```

Include the following two lines in `~/.profile`:

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/Projects
mkdir -p "$WORKON_HOME"
```

All relevant environmental variables are defined in `~/.profile` (`$WORKON_HOME`, `$PROJECT_HOME`, `$VIRTUAL_ENV`, etc.)

Include the following in `~/.bashrc`:

```
#source /usr/bin/virtualenvwrapper.sh      # loading upon shell creation
source /usr/bin/virtualenvwrapper_lazy.sh  # lazy loading
```

The commented out line above may induce slowness every time sub-shells are spawned. Favor lazy invocation per the next line to only load plugin shell functions when they are first needed.

1-2-7-3. Usage

At startup or when spawning a new shell, **virtualenvwrapper.sh** or **virtualenvwrapper_lazy.sh** finds the first Python and **virtualenv** programs on the `$PATH` and remembers them for later use. This eliminates any conflict as the `$PATH` changes, enabling interpreters inside VEs where **virtualenvwrapper** is not installed or even where different versions of **virtualenv** are installed. To make that behavior possible, it is important for the `$PATH` to be set **before** sourcing **virtualenvwrapper.sh** or **virtualenvwrapper_lazy.sh**.

Make VE `env1` inside `${WORKON_HOME}/`

```
$ mkvirtualenv env1      # creates new VE in $WORKON_HOME and switches to it
(env1)$ ls $WORKON_HOME
env1 hook.log
```

The general syntax is:

```
$ mkvirtualenv [-a project_path] [-i package] [-r requirements] [virtualenv options]
ENVNAME
```

Install a package inside `${WORKON_HOME}/env1`

```
(env1)$ pip install django
(env1)$ lssitepackages
Django-1.1.1-py2.6.egg-info      easy-install.pth      setuptools-0.6.10-py2.6.egg
```

```
setuptools-0.6.10-py2.6.egg    pip-0.6.3-py2.6.egg    setuptools.pth
django
```

There is no limitation (beyond disc space and memory) as to the number of VEs that can be created:

```
$ mkvirtualenv env2          # makes and switches to newly created VE
(env2)$ ls $WORKON_HOME
env1 env2 hook.log
```

To switch between VEs:

```
(env2)$ workon env1
(env1)$ pwd
/Users/dhellmann/Envs/env1
(env1)$
```

The workon command includes tab completion for the environment names. It also invokes customization scripts as a VE is activated or deactivated (see [Per-User Customization](#)).

To apply a general command to all already defined VEs under \$WORKON_HOME, for instance to update pip:

```
$ allvirtualenv python -m pip install -U pip
```

postactivate and **postmkvirtualenv**, **postdeactivate**, **predeactivate**, and others are special files sourced before or after a new VE is either activated, created, deactivated or on the verge of being so. They permit the automation of repetitive tasks, such as the installation of commonly used tools, the unaliasing of commands reserved for when a VE is active, etc...

Two short examples are provided:

```
(env1)$ echo 'cd $VIRTUAL_ENV' >> $WORKON_HOME/postactivate
(env1)$ workon env2          # the previous cmd
(env2)$ pwd
/Users/dhellmann/Envs/env2

(env2) $ echo 'pip install sphinx' >> $WORKON_HOME/postmkvirtualenv
(env2)$ mkvirtualenv env3
New python executable in env3/bin/python
Installing setuptools.....
.....
Downloading/unpacking sphinx
.....
Successfully installed docutils Jinja2 Pygments sphinx
(env3)$ which sphinx-build
/Users/dhellmann/Envs/env3/bin/sphinx-build
```

More details on cmd shortcuts are available at <https://virtualenvwrapper.readthedocs.io/en/latest> and https://virtualenvwrapper.readthedocs.io/en/latest/command_ref.html.

1-3. 'pyenv'

1-3-1. Working principle

1-3-1-1. Scope

pyenv was previously known as **pythonbrew** and can be found on Github.

When used in conjunction with **virtualenvwrapper** it is one of the most practical way of installing and dealing with different Python versions and their configurations on a Linux platform. Both the Pyenv-virtualenv Virtualenvwrapper plugins extend Pyenv's set of cmds. Both plugins are available either in [AUR](#) or on Github.

pyenv lets you:

- change the global Python version,
- install multiple Python versions,
- set directory (project)-specific Python versions, and
- generally create / manage multiple Python VEs.

All this is done on *nix-style machines (Linux, Unix and OS X) without depending on Python itself. It also works with regular users privileges. There is no need for any privilege escalation to super-user level with *sudo*.

Once activated, it prefixes the `$PATH` environment variable with `$HOME/.pyenv/shims`, where special files matching the Python commands (*python*, *pip*) are found. Those are not copies of the Python-shipped commands; they are special scripts that decide on the fly which version of Python to run based on the `PYENV_VERSION` environment variable, or the `.python-version` file, or the `$HOME/.pyenv/version` file. **pyenv** also makes the process of downloading and installing multiple Python versions easier, using the cmd:

```
$ pyenv install <python_version_number>
```

In a nutshell **pyenv** works by inserting a *directory of shims* at the beginning of `$PATH`:

```
$ echo $PATH
/home/USER/.pyenv/shims:/usr/local/bin:/usr/bin:/bin
```

provided `$PYENV_ROOT` is equal to `/home/USER/.pyenv`.

Through a process called *rehashing*, **pyenv** maintains shims in that directory to match every Python cmd across all its installed versions. Shims are lightweight executables that simply pass your cmd along to **pyenv**. So with **pyenv** installed, when you run *pip*, your operating system will do the following:

- Search `$PATH` for an executable file named *pip*
- Find the **pyenv** shim named *pip* at the beginning of `$PATH`
- Run the shim named *pip*, which in turn passes the command along to **pyenv**

Setting a local project-specific Python version is possible by writing the version name to a `.python-version` file in the current VE-directory. That *local version* overrides the *global version*, and can be overridden itself by setting the `$PYENV_VERSION` environment variable or with the **pyenv** shell command.

1-3-1-2. Plugins: ‘pyenv-virtualenv’ and ‘pyenv-virtualenvwrapper’

We already mentioned the two Pyenv plugins:

The first is ‘[pyenv-virtualenv](#)’, and only requires [pyenv-git](#) as a dependency. It comes with various features to help [pyenv](#) users manage VEs created by [virtualenv](#) or [\(ana\)conda](#). Because the [activate](#) script of those VEs changes the user’s interactive shell’s `$PATH`, it intercept [pyenv](#)’s shim style cmd execution hooks.

The second plugin is ‘[pyenv-virtualenvwrapper](#)’. It requires dependencies [pyenv](#) as well as [python-virtualenvwrapper](#) which in turn requires [python-virtualenv](#). You may check the PKGBUILD file in the AUR repo for confirmation or for any possible future change in the dependencies.

1-3-2. Installation of ‘pyenv’ (pyenv-git)

Either

```
$ cd /path/to/your/local_AUR_builds
$ git clone https://aur.archlinux.org/pyenv-git.git
$ cd pyenv-git; makepkg -sric
```

or

```
$ git clone https://github.com/pyenv/pyenv.git ~/.pyenv
```

or, if [pyenv](#) is available in your usual package repository, e.g. for Archlinux with [pacman](#):

```
$ sudo pacman -Syu pyenv
```

For Debian-like systems with `apt-get` and `curl`:

```
$ sudo apt-get install -y make build-essential git libssl-dev zlib1g-dev \
    libbz2-dev libreadline-dev libsqlite3-dev wget curl \
    llvm libncurses5-dev libncursesw5-dev xz-utils tk-dev
$ curl -L https://raw.githubusercontent.com/pyenv/pyenv-installer/master/bin/pyenv-
installer | bash
```

To get help, issue the cmd:

```
$ pyenv help
```

1-3-3. Configuration and update of ‘pyenv’

Add the following lines in `~/.profile`:

```
export PYENV_ROOT=${HOME}/.pyenv
export PATH=${PYENV_ROOT}/shims:${PYENV_ROOT}/bin:$PATH" # prepend PATH !!
export WORKON_HOME="${HOME}/.virtualenvs"           # VEs’ local repo
mkdir -p "${WORKON_HOME}"
export PROJECT_HOME="${HOME}/path_to/project_directory" # project directory
```

Add the following lines in `~/.bashrc`:

```
[ -n "$(command -v pyenv)" ] && eval "$(pyenv init -)"
pyenv global "$(/usr/bin/python --version | cut -d ' ' -f2 2>/dev/null)"
/usr/bin/echo "$(pyenv global)" >| "${PYENV_ROOT}/version
source /usr/bin/virtualenvwrapper.sh
```

or `source /usr/bin/virtualenvwrapper_lazy.sh`

Note that when *lazy-sourcing*, tab-completion of arguments to `virtualenvwrapper` commands (such as environment names) is not enabled until after the first command has been run. Also there seems to be a bug that causes the `virtualenvwrapper` plugin cmd `workon` to make your terminal session crash the plugin is invoked lazily.

`pyenv init -` loads extra commands into your shell. Here's what it actually does:

1. **Sets up your shims path.** This is the only requirement for `pyenv` to function properly. You can do this by hand by pre-pending `$(pyenv root)/shims` to `PATH`.
2. **Installs autocompletion.** Sourcing `$(pyenv root)/completions/pyenv.bash` will set that up.
3. **Rehashes shims.** From time to time you'll need to rebuild your shim files. Doing this on `init` makes sure everything is up to date. Instead you can run `pyenv rehash` manually.
4. **Installs the sh dispatcher.** This bit is also optional, but allows `Pyenv` and plugins to change variables in your current shell, making commands such as `pyenv shell` possible. The sh dispatcher doesn't do anything crazy such as overriding `cd` or hacking your shell prompt, but if for some reason you need `pyenv` to be a real script rather than a shell function, you can skip it safely.

To see exactly what happens under the hood, just look at the standard output of `pyenv init -`.

Back to the console, check the list of available Python versions, pick a couple (e.g. `UVW`, `[XYZ]`) and install them:

```
$ pyenv root
/home/$USER/.pyenv
$ cd $PYENV_ROOT/plugins/python-build/../../ && git pull && cd -
$ pyenv install -l
[... list of available python versions]
$ pyenv install UVW [XYZ]      # where XYZ may be "3.7.3" or "stackless-2.7.5", etc.
$ pyenv versions              # check list of Python version(s) installed in $PYENV_ROOT
```

After installing at least one version of Python, you can set it globally with a shell snippet in `~/.bashrc`. If not set `pyenv`'s global python version defaults to that of your OS, denoted as "`system`" in the output of:

```
$ pyenv versions

$ pyenv global XYZ      # where, as noted above, XYZ can be any valid Python version
                        # available to your pyenv setup
```

or, if you want the system-wide Python version to be the one to be available as the `pyenv global` version:

```
$ pyenv global "$(/usr/bin/python --version | cut -d' ' -f2)"
or $ echo "$(/usr/bin/python --version | cut -d' ' -f2 2>/dev/null)" \
    >| "${PYENV_ROOT}/version
```

Caution:

Check that the latest version available on your platform (given to you by `/usr/bin/python -version`) is also listed in the output of `pyenv install --list`. If not, update either the cloned git repo in `~/.pyenv` and choose the closest prior version available:

```
$ cd $(pyenv root)
$ git pull
```

or update your installed distro package 's version.

Back to the console, check that your globally set Python version is correct:

```
$ pyenv global
```

and that it corresponds to the right default executable path, outside any activated VE:

```
$ which python
/home/$USER/$(pyenv root)/shims/python
$ eval $(which python) -version    # should yield the same version as pyenv global
```

1-3-4. Uninstalling 'pyenv'

The simplicity of Pyenv makes it easy to disable it temporarily or to uninstal it.

1. To **disable** Pyenv, that is to prevent it from managing your Python VEs, simply remove

```
eval "$(pyenv init -)"
```

from `~/.bashrc`. This will remove the `$HOME/pyenv/shims` directory from `$PATH`, and future invocations of `python` will execute your platform's Python version, as it did prior to Pyenv installation. Pyenv will still be accessible on the command line, but your Python apps won't be affected by version switching.
2. To completely **uninstall** Pyenv, go through step (1) and remove Pyenv's root directory. This will **delete all Python versions** installed under directory `$(pyenv root)/versions/`:

```
$ rm -rf $(pyenv root)
```

1-3-5. Installation of 'pyenv-virtualenv'

Create the `$(pyenv root)/plugins` directory if it does not already exist and switch to it. Then clone the Github [repo](#) in it.


```
$ mkdir -p $(pyenv root)/plugins
$ cd $(pyenv root)/plugins
$ git clone https://github.com/pyenv/pyenv-virtualenv
$ cd pyenv-virtualenv; makepkg -sr
```

In `~/.bashrc`, add `eval "$(pyenv virtualenv-init -)"` after `eval "$(pyenv init -)"`.

`pyenv virtualenv-init -` will automatically activate and deactivate VEs upon entering a directory, as long as `$PWD` contains the file `.python-version` listing the name of a valid VE as shown in the output of `pyenv virtualenvs`.

2. Using Python VEs

2-1. 'pyenv' usage

2-1-1. Selection of the VE's Python version

When you try to run Python, Python must first decide which one of its versions to run. To do that it first looks for a file named `.python-version` in the current directory. If it doesn't find this file, it looks for the user-level file `$(pyenv root)/version`. The latter contains the global default Python version, the same that obtains with:

```
$ pyenv global          # displays the global default Python version
$ pyenv prefix          # displays the currently selected Python version
```

List all available Python versions available in the `$(pyenv root)/versions` directory with:

```
$ pyenv versions
```

Alternatively the fully-qualified directory `$(pyenv root)/versions` contains all Python versions installed as sub-directories.

To remove old Python versions, you can automate the removal process, with:

```
$ pyenv uninstall <version_to_remove>
```

Alternatively, simply removing the directory of the version is expedient:

```
$ \rm -rf $(pyenv root)/versions/<Python_version>
```

You can find the version's directory to be deleted by issuing:

```
$ pyenv prefix 2.6.8.
$(pyenv root)/versions/2.6.8
```

If any particular version is not present or not installed, a warning message appears. You can add it with:

```
$ pyenv install --list | less  # displays all Python versions available for installation
$ pyenv install --list | sed -En 's/^\s+//; /^[3-9]\.[0-9]+\.[0-9]+$/'
```

where the second command lists all Python 3 versions excluding versions whose major and minor include a letter e.g. alpha and beta versions and non standard build releases.

To install one or more new Python version(s), available in the previous list:

```
$ pyenv install <Python_version1> [<Python_version2>]
```

pyenv install

To install a new Python version (using [python-build](#)), use:

```
$ pyenv install [-f] [-kvp] <version>
$ pyenv install [-f] [-kvp] <definition_file>
$ pyenv install -l|--list
```

install options:

```
-l/--list           List all available versions
-f/--force          Install even if the version appears to be installed already
-s/--skip-existing  Skip the installation if the version appears to be installed already
```

python-build options:

```
-k/--keep           Keep source tree in $PYENV_BUILD_ROOT after installation
(defaults to $PYENV_ROOT/sources)
-v/--verbose        Verbose mode: print compilation status to stdout
-p/--patch          Apply a patch from stdin before building
-g/--debug          Build a debug version
```

pyenv rehash

Installs shims for all Python binaries known to pyenv (i.e., ~/.pyenv/versions/*/bin/*). Run this command after you install a new version of Python, or install a package that provides binaries.

```
$ pyenv rehash
```

pyenv which

Displays the full path to the executable that Pyenv will invoke when you run the given command.

```
$ pyenv which python3.3
/home/$USER/.pyenv/versions/3.3.3/bin/python3.3
```

pyenv whence

Lists all Python versions with the given command installed.

```
$ pyenv whence 2to3
2.6.8
2.7.6
3.3.3
```

2-1-2. Activation of the Python VE

After installing a Python version available to Pyenv, you can create and simultaneously activate a VE based on that Python version in `my_directory`, like so:

```
$ cd /path/to/my_directory
$ pyenv local <your choice of pyenv-installed Python version>
```

This will create the `/path/to/my_directory/.python-version` file containing the Python version for the VE. The VE is limited to the directory that contains the `.python-version` file.

You can specify one version (2.x or 3.x) or multiple versions (2.x and 3.x) as local Python at once.

2-1-3. Removing a Python VE

Running `pyenv local` without a version number will just query its immediate environment and report the local configuration's VE version if any. In case a local VE configuration exist, run:

```
$ cd /path/to/my_directory
$ pyenv local --unset
```

or

```
$ rm -f .python-version
```

to remove that local configuration.

2-2. 'pyenv-virtualenv' usage with 'pyenv'

2-2-1. Creation of Python VEs

To create a VE for the Python version used with Pyenv, run `pyenv virtualenv`, specifying the Python version you want and the name of the VE directory:

```
$ pyenv virtualenv 2.7.10 ve2710
```

creates a VE based on Python 2.7.10 under `$(pyenv root)/versions/ve2710`. If the Python version is not explicitly specified, the VE is created by default, based on the current version.

```
$ pyenv version
3.4.3 (set by /home/yyuu/.pyenv/version)
$ pyenv virtualenv venv34
```

2-2-2. Listing of Python VEs

To list all existing Python (and Conda) VEs created as shown above, with the command `pyenv virtualenvs`, issue:

```
$ pyenv virtualenvs
...
```

Note that there are two entries for each VE; the shorter one is just a symlink.

2-2-3. Automatic / manual virtual environments activation / deactivation

Automatic virtual environment activation is possible upon entering in a directory, if `eval "$(pyenv virtualenv-init -)"` is configured in your shell (`~/.bashrc`), and provided the `$PWD` contain a `.python-version` file that contains the name of a valid virtual environment as shown in the output of:

```
$ pyenv virtualenvs
```

`.python-version` files are used by Pyenv to denote local Python versions and can be created and deleted with the [*pyenv local*](#) command. You can also activate and deactivate a Pyenv VE manually:

```
$ pyenv activate <VE_name>
$ pyenv deactivate
```

2-2-4. Deletion of an existing virtual environments

Removing the subdirectory that correspond to a specific VE in `$(pyenv root)/versions` amounts to picking the subdirectory `$(pyenv root)/versions/{version}` and will delete the VE and all packages installed in it. Another option is to run:

```
$ pyenv uninstall <my_virtual_environment>
```

or

```
$ pyenv virtualenv-delete <my_virtual_environment>
```

2-3. Working with shell-specific Python versions

To set a shell-specific Python version, just set the `PYENV_VERSION` env-var in your shell. This Python version overrides local application-specific versions as well as the global version.

```
$ pyenv shell <Python_version1> [<Python_version2>]
```

where `<Python_version>` should be either “`system`” or any “`XYZ`” string matching a Python version known to (i.e. installed in) ‘`pyenv`’.

To restore to the previously set shell-specific value of `PYENV_VERSION`:

```
$ pyenv shell -
```

To unset the shell-specific python version altogether, issue:

```
$ pyenv shell --unset
```

2-4. Run a Jupyter notebook in a Python virtual environment

On a platform configured with Pyenv, shims will preempt any call to `python` on the CLI by placing a new path to Python, e.g. `/home/<USER>/.pyenv/shims/python3`, before the standard Python3 entries `/bin` and `/usr/bin` in the `PATH` environment variable. Invoking the platform default Python version is always possible by specifying the full path, as in:

```
$ /usr/bin/python
```

However running a Jupyter notebook in a specific Python environment requires the configuration of a new iPython kernel, which must be later invoked from within the notebook.

The procedure to follow is simple:

```
$ cd /path/to/my_directory          # go to the virtual environment specific directory
$ pyenv local 3.7.0                 # specify the virtual environment's python version, e.g. 3.7.0
$ python -m pip install ipykernel    # install ipykernel in that virtual environment
```

Further install any Python package needed in that virtual environment, before creating a new kernel (“`my_kernel`”) based on the virtual environment Python version:

```
$ python -m pip install matplotlib pandas seaborn tensorflow==1.14.0 keras==2.2.5
$ python -m pip list                 # list local environment's packages
$ awk 'NR==2' < $(python -m pip show pandas) # Pandas' installed version in the VE
Version: 1.1.5
$ ipython kernel install --user --name <my_kernel> --display-name \
    "Python3.7 (<my_kernel>)"
```

The new kernel is now available to iPython sessions, such as Jupyter notebooks. You can check that with:

```
$ jupyter kernelspec list
Available kernels:
Python3.7 (<my_kernel>)    /home/USER/.local/share/jupyter/kernels/<my_kernel>
python3                   /home/USER/.pyenv/versions/3.6.0/share/jupyter/kernels/python3
```

Finally create a new iPython session with Jupyter from within the virtual environment specific directory:

```
$ cd /path/to/my_directory
$ jupyter notebook
```

This will cause a new window to open in your default browser. In that window you may select “`my_kernel`” under the **New notebook** field in the top-right region of the browser page.

Within a Jupyter notebook cell, check that the iPython kernel used in the Jupyter notebook points to the right Python version:

```
[1] import sys
    sys.executable

    '/home/USER/.pyenv/versions/3.7.0/bin/python'
```

References

- pyenv: <https://github.com/pyenv/pyenv>
- pyenv's complete cmds' reference: <https://github.com/pyenv/pyenv/blob/master/COMMANDS.md>
- pyenv-virtualenvwrapper: <https://github.com/pyenv/pyenv-virtualenvwrapper>
- virtualenvwrapper: <https://virtualenvwrapper.readthedocs.io/en/latest/>
- https://opencafe.readthedocs.io/en/latest/getting_started/pyenv/

Appendix A: Work with 'virtualenvwrapper'

Prior to creating a Python virtual environment, pick which version of Python you need for that environment:

```
$ pyenv versions
```

To create a new isolated virtual environment based on your systems global default Python version, you may simply start a new project. The minimal requirement is that the project's name be specified – e.g. 'pytf' below:

```
$ mkproject pytf
```

Given *PROJECT_HOME* as defined in *~/.bashrc*, a new empty project directory will be automatically created at *\$PROJECT_HOME/pytf*, and a new virtual environment will be created at *\$WORKON_HOME/pytf* based on the Python version's global default for your system. You are also automatically transferred to the new project's directory.

The complete syntax for the above cmd is:

```
$ mkproject [-f | --force] [-t template] [-p <python_executable>] \
    [+ other virtualenv options] pytf
```

where

- f, --force** Create the virtualenv even if the project directory already exists
- t <template>** Multiple templates can be selected, applied following cli order
- p <python_executable>** as defined by existing Python version installations, e.g.
\$(pyenv root)/versions/2.7.16/bin/python2

The Python executable must be provided with an absolute path or as above.

Instead you can also use, with an already *created* project, *pytf*:

```
$ mkvirtualenv [-a existing_project_dir_path] [-i package_to_install] \
    [-r pip_packages_list_file] [-p <python_executable>] \
    [+ other virtualenv options] /path/to/project/pytf/directory
```

where you may specify:

- a <project_directory_absolute_path>**, when the project directory already exists
- i <package_to_install>**, to specify one or several packages to be install after virtualenv creation, repeating '-i ...' as many time as needed,
- r <pip_packages_list_file>**, a file containing a list of packages to be processed as arguments of `pip -r`` for installation in the new environment,
- {virtualenv options}**, e.g.: **-p <python_executable>** as defined by existing Python version installations, \$(pyenv root)/versions/2.7.16/bin/python

The Python executable must be provided with an absolute path, or following query of available Python flavors and installation in the available versions directory:

```
$ pyenv install -l
[... list of available python versions]
$ pyenv install XYZ      # where XYZ may be "3.6.0" or "stackless-2.7.5" or ...
$ pyenv versions
```

This would create a new Python's installation in the `~/.virtualenvs/pytf/` directory based on Python version 2.7.16. Upon its creation 'pyenv' switches directly to that virtual environment. In other circumstances, manual switching to any already created virtual environment is done by calling:

```
$ workon pytf          # switch to project directory and activate virtualenv
(pytf) $ _
```

Now the prompt should reflect the fact that you are in the project's directory, in an activated virtual environment.

In principle the *workon* cmd should switch your *PWD* to the project's directory automatically, unless the inline option **-n** or **--no-cd** is present.

Package installation may proceed with *pip*:

```
(pytf) $ pip install tensorflow
```

This package will be installed only in the `pytf` environment. You can list all installed packages by calling:

```
(pytf) $ lssitepackages
```

A list of built virtual environments can be obtained, and any deactivated virtual environment can be removed quite simply, by calling:

```
(pytf) $ deactivate

$ lsvirtualenv [-b | -l] or [-h] # to list virtual environments or to get help

$ rmvirtualenv pytf    # must be deactivated first
```

For more details on cmd's syntax, consult the 'virtualenvwrapper' [documentation](#).

Appendix B: GNU Free Documentation License

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of

the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent

pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and

publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these

copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently

reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.