



Red Hat

Red Hat Enterprise Linux 10

Installing and using dynamic programming languages

Installing and using Python and PHP in Red Hat Enterprise Linux 10

Red Hat Enterprise Linux 10 Installing and using dynamic programming languages

Installing and using Python and PHP in Red Hat Enterprise Linux 10

Legal Notice

Copyright © Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Install and use Python 3. Install the PHP scripting language, use PHP with the Apache HTTP Server or the nginx web server, and run a PHP script from a command-line interface.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	3
CHAPTER 1. INSTALLING AND USING PYTHON	4
1.1. PYTHON VERSIONS	4
1.2. INSTALLING PYTHON 3	4
1.3. INSTALLING ADDITIONAL PYTHON 3 PACKAGES	4
1.4. INSTALLING ADDITIONAL PYTHON 3 TOOLS FOR DEVELOPERS	5
1.5. USING PYTHON	5
1.6. ADDITIONAL RESOURCES	6
CHAPTER 2. PACKAGING PYTHON 3 RPMS	7
2.1. A SPEC FILE DESCRIPTION FOR AN EXAMPLE PYTHON PACKAGE	7
2.2. COMMON MACROS FOR PYTHON 3 RPMS	9
2.3. USING AUTOMATICALLY GENERATED DEPENDENCIES FOR PYTHON RPMS	10
CHAPTER 3. INSTALLING TCL/TK	12
3.1. INSTALLING TCL	12
3.2. INSTALLING TK	12
CHAPTER 4. USING THE PHP SCRIPTING LANGUAGE	14
4.1. INSTALLING THE PHP SCRIPTING LANGUAGE	14
4.2. USING THE PHP SCRIPTING LANGUAGE WITH A WEB SERVER	14
4.2.1. Using PHP with the Apache HTTP Server	14
4.2.2. Using PHP with the nginx web server	16
4.3. RUNNING A PHP SCRIPT USING THE COMMAND LINE	17

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INSTALLING AND USING PYTHON

Python is a high-level programming language that supports multiple programming paradigms, such as object-oriented, imperative, functional, and procedural paradigms. Python has dynamic semantics and can be used for general-purpose programming.

With Red Hat Enterprise Linux, many packages that are installed on the system, such as packages providing system tools, tools for data analysis, or web applications, are written in Python. To use these packages, you must have the **python*** packages installed.

1.1. PYTHON VERSIONS

Python 3.12 is the default Python implementation in RHEL 10. Python 3.12 is distributed in a non-modular **python3** RPM package in the BaseOS repository and is usually installed by default. Python 3.12 will be supported for the whole life cycle of RHEL 10.

Additional versions of Python 3 will be distributed as non-modular RPM packages with a shorter life cycle through the AppStream repository in minor RHEL 10 releases. You will be able to install these additional Python 3 versions in parallel with Python 3.12.

The unversioned **python** command points to the default Python 3.12 version.

For details about the length of support, see [Red Hat Enterprise Linux Life Cycle](#) and [Red Hat Enterprise Linux Application Streams Life Cycle](#).

1.2. INSTALLING PYTHON 3

The default Python implementation is usually installed by default. To install it manually, use the following procedure.

Procedure

- To install Python 3.12, enter:

```
# dnf install python3
```

Verification

- Verify the Python version installed on your system:

```
$ python3 --version
```

1.3. INSTALLING ADDITIONAL PYTHON 3 PACKAGES

Packages prefixed with **python3-** contain add-on modules for the default Python 3.12 version.

Procedure

- To install, for example, the **Requests** module for Python 3.12, enter:

```
# dnf install python3-requests
```

- To install the **pip** package installer from Python 3.12, enter:

```
# dnf install python3-pip
```

Additional resources

- [Upstream documentation about Python add-on modules](#)

1.4. INSTALLING ADDITIONAL PYTHON 3 TOOLS FOR DEVELOPERS

Additional Python tools for developers are distributed through the CodeReady Linux Builder (CRB) repository.

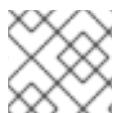


IMPORTANT

The content in the CodeReady Linux Builder repository is unsupported by Red Hat.

The CRB repository contains, for example, the following packages:

- **python3-pytest**
- **python3-idle**
- **python3-debug**
- **python3-cython**



NOTE

Not all upstream Python-related packages are available in RHEL.

Procedure

1. Enable the CodeReady Linux Builder repository:

```
# subscription-manager repos --enable codeready-builder-for-rhel-10-x86_64-rpms
```

2. Install, for example, the **python3-cython** package:

```
# dnf install python3-cython
```

Additional resources

- [How to enable and make use of content within CodeReady Linux Builder](#)

1.5. USING PYTHON

The following procedure contains examples of running the Python interpreter or Python-related commands.

Prerequisites

- Python is installed.
- If you want to download and install third-party applications, install the **python3-pip** package.



WARNING

Installing Python packages with **pip** as the root user places files in system locations. This can override RHEL libraries and might cause system instability or conflicts with supported packages. Red Hat does not support software installed by using **pip** at the system level. To avoid these issues, use **pip** within a virtual environment or install packages as a non-root user with the **--user** option.

Procedure

- To run the Python 3.12 interpreter or related commands, use, for example, the following commands:

```
$ python3  
$ python3 -m venv --help  
$ python3 -m pip install <package>  
$ pip3 install <package>
```

1.6. ADDITIONAL RESOURCES

- [Packaging Python 3 RPMs](#)
- [Modifying interpreter directives in Python scripts](#)

CHAPTER 2. PACKAGING PYTHON 3 RPMS

You can install Python packages on your system by using the **DNF** package manager. **DNF** uses the RPM package format, which offers more downstream control over the software.

Packaging a Python project into an RPM package provides the following advantages compared to native Python packages:

- Dependencies on Python and non-Python packages are possible and strictly enforced by the **DNF** package manager.
- You can cryptographically sign the packages. With cryptographic signing, you can verify, integrate, and test contents of RPM packages with the rest of the operating system.
- You can execute tests during the build process.

The packaging format of native Python packages is defined by [Python Packaging Authority \(PyPA\) Specifications](#). Historically, most Python projects used the **distutils** or **setuptools** utilities for packaging and defined package information in the **setup.py** file. However, possibilities of creating native Python packages have evolved over time:

- To package Python software that uses the **setup.py** file, follow this document.
- To package more modern packages with **pyproject.toml** files, see the **README** file in [pyproject-rpm-macros](#). Note that **pyproject-rpm-macros** is included in the CodeReady Linux Builder (CRB) repository, which contains unsupported packages, and it can change over time to support newer Python packaging standards.

2.1. A SPEC FILE DESCRIPTION FOR AN EXAMPLE PYTHON PACKAGE

An RPM **spec** file for Python projects has some specifics compared to non-Python RPM **spec** files.

Note that it is recommended for any RPM package name of a Python library to include the **python3-** prefix.

See the notes about Python RPM **spec** files specifics in the following example of the **python3-pello** package.

An example SPEC file for the pello program written in Python

```
%global python3_pkgversion 3  
Name:      python-pello  
Version:   1.0.2  
Release:   1%{?dist}  
Summary:   Example Python library  
  
License:    MIT  
URL:       https://github.com/fedora-python/Pello  
Source:     %{url}/archive/v%{version}/Pello-%{version}.tar.gz  
  
BuildArch:  noarch  
BuildRequires: python%{python3_pkgversion}-devel
```

1

2

3

```

# Build dependencies need to be specified manually
BuildRequires: python%{python3_pkgversion}-setuptools

# Test dependencies need to be specified manually
# Runtime dependencies need to be BuildRequired manually to run tests during build
BuildRequires: python%{python3_pkgversion}-pytest >= 3

%global _description %{expand:
Pello is an example package with an executable that prints Hello World! on the command line.}

%description %_description

%package -n python%{python3_pkgversion}-pello
Summary:    %{summary} 4

%description -n python%{python3_pkgversion}-pello %_description

%prep
%autosetup -p1 -n Pello-%{version}

%build
# The macro only supports projects with setup.py
%py3_build 5

%install
# The macro only supports projects with setup.py
%py3_install

%check
%pytest 6

# Note that there is no %%files section for python-pello
%files -n python%{python3_pkgversion}-pello
%doc README.md
%license LICENSE.txt
%{_bindir}/pello_greeting

# The library files needed to be listed manually
%{python3_sitelib}/pello/

# The metadata files needed to be listed manually
%{python3_sitelib}/Pello-*.*.egg-info/

```

- 1 By defining the **python3_pkgversion** macro, you set which Python version this package will be built for. To build for the default Python version **3.12**, remove the line.
- 2 When packaging a Python project into RPM, always add the **python-** prefix to the original name of the project. The project name here is **Pello** and, therefore, the name of the Source RPM (SRPM) is **python-pello**.

- 3 **BuildRequires** specifies what packages are required to build and test this package. In **BuildRequires**, always include items providing tools necessary for building Python packages:
- 4 When choosing a name for the binary RPM (the package that users will be able to install), add a versioned Python prefix. Use the **python3-** prefix for the default Python 3.12. You can use the **%{python3_pkgversion}** macro, which evaluates to **3** for the default Python version **3.12** unless you set it to an explicit version, for example, when a later version of Python is available (see footnote 1).
- 5 The **%py3_build** and **%py3_install** macros run the **setup.py build** and **setup.py install** commands, respectively, with additional arguments to specify installation locations, the interpreter to use, and other details.



NOTE

Using the **setup.py build** and **setup.py install** commands from the **setuptools** package is deprecated and will be removed in the future major RHEL release. You can use [pyproject-rpm-macros](#) instead.

- 6 The **%check** section runs the tests of the packaged project. The exact command depends on the project itself, but you can use the **%pytest** macro to run the **pytest** command in an RPM-friendly way.

2.2. COMMON MACROS FOR PYTHON 3 RPMS

In a Python RPM **spec** file, always use the macros for Python 3 RPMs rather than hardcoding their values.

You can redefine which Python 3 version is used in these macros by defining the **python3_pkgversion** macro on top of your **spec** file. For more information, see [A spec file description for an example Python package](#). If you define the **python3_pkgversion** macro, the values of the macros described in the following table will reflect the specified Python 3 version.

Table 2.1. Macros for Python 3 RPMs

Macro	Normal Definition	Description
<code>%{python3_pkgversion}</code>	3	The Python version that is used by all other macros. Can be redefined to any future Python versions that will be added.
<code>%{python3}</code>	/usr/bin/python3	The Python 3 interpreter.
<code>%{python3_version}</code>	3.12	The major.minor version of the Python 3 interpreter.
<code>%{python3_sitelib}</code>	/usr/lib/python3.12/site-packages	The location where pure-Python modules are installed.
<code>%{python3_sitearch}</code>	/usr/lib64/python3.12/site-packages	The location where modules containing architecture-specific extension modules are installed.

Macro	Normal Definition	Description
%py3_build		Expands to the setup.py build command with arguments suitable for an RPM package.
%py3_install		Expands to the setup.py install command with arguments suitable for an RPM package.
%{py3_shebang_flags}	sP	The default set of flags for the Python interpreter directives macro, %py3_shebang_fix .
%py3_shebang_fix		Changes Python interpreter directives to #! %{python3} , preserves any existing flags (if found), and adds flags defined in the %{py3_shebang_flags} macro.

Additional resources

- [Python macros in upstream documentation](#)

2.3. USING AUTOMATICALLY GENERATED DEPENDENCIES FOR PYTHON RPMS

You can automatically generate dependencies for Python RPMs by using upstream-provided metadata.

Prerequisites

- A **spec** file for the RPM exists. For more information, see [A spec file description for an example Python package](#).

Procedure

1. Include one of the following directories in the resulting RPM:

- **.dist-info**

- **.egg-info**

The RPM build process automatically generates virtual **pythonX.Ydist** provides from these directories, for example:

python3.12dist(pello)

The Python dependency generator then reads the upstream metadata and generates runtime requirements for each RPM package using the generated **pythonX.Ydist** virtual provides. Example of a generated requirements tag:

Requires: python3.12dist(requests)

2. Inspect the generated **Requires**.

3. To remove some of the generated **Requires**, modify the upstream-provided metadata in the **%prep** section of the **spec** file.
4. To disable the automatic requirements generator, include the **%{?python_disable_dependency_generator}** macro above the main package's **%description** declaration.

Additional resources

- [Automatically generated dependencies](#)

CHAPTER 3. INSTALLING TCL/TK

Tcl is a dynamic programming language, while **Tk** is a graphical user interface (GUI) toolkit. They provide a powerful and easy-to-use platform for developing cross-platform applications with graphical interfaces. As a dynamic programming language, 'Tcl' provides simple and flexible syntax for writing scripts. The **tcl** package provides the interpreter for this language and the C library. You can use **Tk** as GUI toolkit that provides a set of tools and widgets for creating graphical interfaces. You can use various user interface elements such as buttons, menus, dialog boxes, text boxes, and canvas for drawing graphics. **Tk** is the GUI for many dynamic programming languages.

For more information about Tcl/Tk, see the [Tcl/Tk manual](#) or [Tcl/Tk documentation web page](#).

3.1. INSTALLING TCL

The default **Tcl** implementation is usually installed by default. To install it manually, use the following procedure.

Procedure

- To install **Tcl**, use:

```
# dnf install tcl
```

Verification

- To verify the Tcl version installed on your system, run the interpreter **tclsh**:

```
$ tclsh
```

- In the interpreter run this command:

```
% info patchlevel  
8.6
```

- You can exit the interpreter interface by pressing **Ctrl+C**

3.2. INSTALLING TK

The default **Tk** implementation is usually installed by default. To install it manually, use the following procedure.

Procedure

- To install **Tk**, use:

```
# dnf install tk
```

Verification

- To verify the **Tk** version installed on your system, run the window shell **wish**. You need to be running a graphical display:

\$ wish

- In the shell run this command:

```
% puts $tk_version  
8.6
```

- You can exit the interpreter interface by pressing **Ctrl+C**

CHAPTER 4. USING THE PHP SCRIPTING LANGUAGE

Hypertext Preprocessor (PHP) is a general-purpose scripting language mainly used for server-side scripting. You can use PHP to run the PHP code by using a web server.

4.1. INSTALLING THE PHP SCRIPTING LANGUAGE

In RHEL 10, PHP is available in PHP 8.3 as the **php** RPM package.

Procedure

- To install PHP 8.3, enter:

```
# dnf install php
```

Additional resources

- [Managing software with the DNF tool](#)

4.2. USING THE PHP SCRIPTING LANGUAGE WITH A WEB SERVER

4.2.1. Using PHP with the Apache HTTP Server

In Red Hat Enterprise Linux 10, the **Apache HTTP Server** allows you to run PHP as a **FastCGI** process server. **FastCGI Process Manager** (FPM) is an alternative PHP **FastCGI** daemon that allows a website to manage high loads. PHP uses **FastCGI Process Manager** by default in RHEL 10.

Prerequisites

- The PHP scripting language is installed on your system.

Procedure

- Configure the **Apache HTTP Server** and **php-fpm** to run PHP scripts.

- a. Install the **httpd** package:

```
# dnf install httpd
```

- b. Start the **Apache HTTP Server**:

```
# systemctl start httpd
```

Or, if the **Apache HTTP Server** is already running on your system, restart the **httpd** service after installing PHP:

```
# systemctl restart httpd
```

- c. Start the **php-fpm** service:

```
# systemctl start php-fpm
```

- d. Optional: Enable both services to start at boot time:

```
# systemctl enable php-fpm httpd
```

- e. To obtain information about your PHP settings, create the **index.php** file with the following content in the **/var/www/html/** directory:

```
# echo '<?php phpinfo(); ?>' > /var/www/html/index.php
```

- f. To run the **index.php** file, point the browser to:

```
http://<hostname>/
```

- g. Optional: Adjust configuration if you have specific requirements:

- **/etc/httpd/conf/httpd.conf** - generic **httpd** configuration
- **/etc/httpd/conf.d/php.conf** - PHP-specific configuration for **httpd**
- **/usr/lib/systemd/system/httpd.service.d/php-fpm.conf** - by default, the **php-fpm** service is started with **httpd**
- **/etc/php-fpm.conf** - FPM main configuration
- **/etc/php-fpm.d/www.conf** - default **www** pool configuration

- Run the "Hello, World!" PHP script using the Apache HTTP Server.

- a. Create a **hello** directory for your project in the **/var/www/html/** directory:

```
# mkdir hello
```

- b. Create a **hello.php** file in the **/var/www/html/hello/** directory with the following content:

```
# <!DOCTYPE html>
<html>
<head>
<title>Hello, World! Page</title>
</head>
<body>
<?php
    echo 'Hello, World!';
?>
</body>
</html>
```

- c. Start the **Apache HTTP Server**:

```
# systemctl start httpd
```

- d. To run the **hello.php** file, point the browser to:

```
http://<hostname>/hello/hello.php
```

As a result, a web page with the "Hello, World!" text is displayed.

See **httpd(8)**, **httpd.conf(5)**, and **php-fpm(8)** man pages for more information.

Additional resources

- [Setting up the Apache HTTP web server](#)

4.2.2. Using PHP with the nginx web server

You can run PHP code through the **nginx** web server.

Prerequisites

- The PHP scripting language is installed on your system.

Procedure

- Configure the **nginx** web server and **php-fpm** to run PHP scripts.

- a. Install the **nginx** package:

```
# dnf install nginx
```

- b. Start the **nginx** server:

```
# systemctl start nginx
```

Or, if the **nginx** server is already running on your system, restart the **nginx** service after installing PHP:

```
# systemctl restart nginx
```

- c. Start the **php-fpm** service:

```
# systemctl start php-fpm
```

- d. Optional: Enable both services to start at boot time:

```
# systemctl enable php-fpm nginx
```

- e. To obtain information about your PHP settings, create the **index.php** file with the following content in the **/usr/share/nginx/html/** directory:

```
# echo '<?php phpinfo(); ?>' > /usr/share/nginx/html/index.php
```

- f. To run the **index.php** file, point the browser to:

```
http://<hostname>/
```

- g. Optional: Adjust configuration if you have specific requirements:

- **/etc/nginx/nginx.conf** – **nginx** main configuration
- **/etc/nginx/conf.d/php-fpm.conf** – FPM configuration for **nginx**
- **/etc/php-fpm.conf** – FPM main configuration
- **/etc/php-fpm.d/www.conf** – default **www** pool configuration
- Run the "Hello, World!" PHP script using the **nginx** server.

- a. Create a **hello** directory for your project in the **/usr/share/nginx/html/** directory:

```
# mkdir hello
```

- b. Create a **hello.php** file in the **/usr/share/nginx/html/hello/** directory with the following content:

```
# <!DOCTYPE html>
<html>
<head>
<title>Hello, World! Page</title>
</head>
<body>
<?php
    echo 'Hello, World!';
?>
</body>
</html>
```

- c. Start the **nginx** server:

```
# systemctl start nginx
```

- d. To run the **hello.php** file, point the browser to:

```
http://<hostname>/hello/hello.php
```

As a result, a web page with the "Hello, World!" text is displayed.

See **nginx(8)** and **php-fpm(8)** man pages for more information.

Additional resources

- [Setting up and configuring NGINX](#)

4.3. RUNNING A PHP SCRIPT USING THE COMMAND LINE

A PHP script is usually run using a web server, but also can be run using the command line.

Prerequisites

- The PHP scripting language is installed on your system.

Procedure

1. Open a text editor and create a PHP file named **hello.php** with the following content:

```
<?php  
    echo 'Hello, World!';  
?>
```

2. Open a terminal and navigate to the directory containing **hello.php**.
3. Run the PHP script from the command line:

```
$ php hello.php  
Hello, World!
```

See **php(1)** man pages for more information.