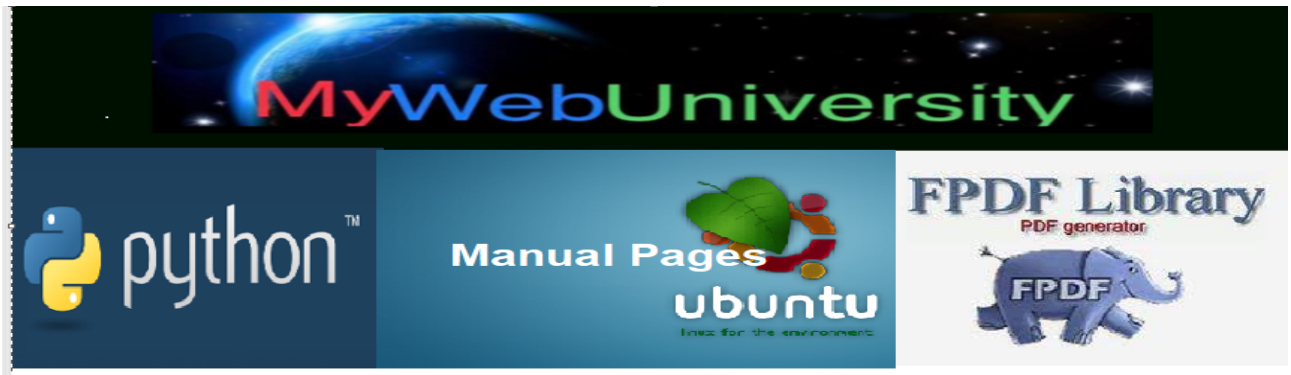

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'pip-install.1'

$ man pip-install.1

PIP3-INSTALL(1) pip PIP3-INSTALL(1)

NAME

pip3-install - description of pip3 install command

DESCRIPTION

Install packages from:

? PyPI (and other indexes) using requirement specifiers.

? VCS project urls.

? Local project directories.

? Local or remote source archives.

pip also supports installing from "requirements files", which provide an easy way to specify

a whole environment to be installed.

USAGE

python -m pip install [options] <requirement specifier> [package-index-options] ...

python -m pip install [options] -r <requirements file> [package-index-options] ...

python -m pip install [options] [-e] <vcs project url> ...

python -m pip install [options] [-e] <local project path> ...

python -m pip install [options] <archive url/path> ...

OPTIONS

-r, --requirement <file>

Install from the given requirements file. This option can be used multiple times.

(environment variable: PIP_REQUIREMENT)

-c, --constraint <file>

Constrain versions using the given constraints file. This option can be used multiple

times.

(environment variable: PIP_CONSTRAINT)

--no-deps

Don't install package dependencies.

(environment variable: PIP_NO_DEPS, PIP_NO_DEPENDENCIES)

--pre Include pre-release and development versions. By default, pip only finds stable ver?

sions.

(environment variable: PIP_PRE)

-e, --editable <path/url>

Install a project in editable mode (i.e. setuptools "develop mode") from a local

project path or a VCS url.

(environment variable: PIP_EDITABLE)

--dry-run

Don't actually install anything, just print what would be. Can be used in combination

with --ignore-installed to 'resolve' the requirements.

(environment variable: PIP_DRY_RUN)

-t, --target <dir>

Install packages into <dir>. By default this will not replace existing files/folders

in <dir>. Use --upgrade to replace existing packages in <dir> with new versions.

(environment variable: PIP_TARGET)

--platform <platform>

Only use wheels compatible with <platform>. Defaults to the platform of the running

system. Use this option multiple times to specify multiple platforms supported by the

target interpreter.

(environment variable: PIP_PLATFORM)

--python-version <python_version>

The Python interpreter version to use for wheel and "Requires-Python" compatibility

checks. Defaults to a version derived from the running interpreter. The version can

be specified using up to three dot-separated integers (e.g. "3" for 3.0.0, "3.7" for

3.7.0, or "3.7.3"). A major-minor version can also be given as a string without dots

(e.g. "37" for 3.7.0).

(environment variable: PIP_PYTHON_VERSION)

--implementation <implementation>

Only use wheels compatible with Python implementation <implementation>, e.g. 'pp', 'jy', 'cp', or 'ip'. If not specified, then the current interpreter implementation is used. Use 'py' to force implementation-agnostic wheels. (environment variable: PIP_IMPLEMENTATION)

--abi <abi>

Only use wheels compatible with Python abi <abi>, e.g. 'pypy_41'. If not specified, then the current interpreter abi tag is used. Use this option multiple times to spec? ify multiple abis supported by the target interpreter. Generally you will need to specify --implementation, --platform, and --python-version when using this option. (environment variable: PIP_ABI)

--user Install to the Python user install directory for your platform. Typically ~/.local/, or %APPDATA%Python on Windows. (See the Python documentation for site.USER_BASE for full details.) (environment variable: PIP_USER)

--root <dir>

Install everything relative to this alternate root directory. (environment variable: PIP_ROOT)

--prefix <dir>

Installation prefix where lib, bin and other top-level folders are placed. Note that the resulting installation may contain scripts and other resources which reference the Python interpreter of pip, and not that of --prefix. See also the --python option if the intention is to install packages into another (possibly pip-free) environment. (environment variable: PIP_PREFIX)

--src <dir>

Directory to check out editable projects into. The default in a virtualenv is "<venv path>/src". The default for global installs is "<current dir>/src". (environment variable: PIP_SRC, PIP_SOURCE, PIP_SOURCE_DIR, PIP_SOURCE_DIRECTORY)

-U, --upgrade

Upgrade all specified packages to the newest available version. The handling of de? pendencies depends on the upgrade-strategy used. (environment variable: PIP_UPGRADE)

--upgrade-strategy <upgrade_strategy>

Determines how dependency upgrading should be handled [default: only-if-needed]. "ea?

ger" - dependencies are upgraded regardless of whether the currently installed ver? sion satisfies the requirements of the upgraded package(s). "only-if-needed" - are upgraded only when they do not satisfy the requirements of the upgraded package(s). (environment variable: PIP_UPGRADE_STRATEGY)

--force-reinstall

Reinstall all packages even if they are already up-to-date.

(environment variable: PIP_FORCE_REINSTALL)

-I, --ignore-installed

Ignore the installed packages, overwriting them. This can break your system if the existing package is of a different version or was installed with a different package manager!

(environment variable: PIP_IGNORE_INSTALLED)

--ignore-requires-python

Ignore the Requires-Python information.

(environment variable: PIP_IGNORE_REQUIRES_PYTHON)

--no-build-isolation

Disable isolation when building a modern source distribution. Build dependencies specified by PEP 518 must be already installed if this option is used.

(environment variable: PIP_NO_BUILD_ISOLATION)

--use-pep517

Use PEP 517 for building source distributions (use --no-use-pep517 to force legacy behaviour).

(environment variable: PIP_USE_PEP517)

--check-build-dependencies

Check the build dependencies when PEP517 is used.

(environment variable: PIP_CHECK_BUILD_DEPENDENCIES)

--break-system-packages

Allow pip to modify an EXTERNALLY-MANAGED Python installation

(environment variable: PIP_BREAK_SYSTEM_PACKAGES)

-C, --config-settings <settings>

Configuration settings to be passed to the PEP 517 build backend. Settings take the form KEY=VALUE. Use multiple --config-settings options to pass multiple keys to the backend.

(environment variable: PIP_CONFIG_SETTINGS)

--global-option <options>

Extra global options to be supplied to the setup.py call before the install or bdist_wheel command.

(environment variable: PIP_GLOBAL_OPTION)

--compile

Compile Python source files to bytecode

(environment variable: PIP_COMPILE)

--no-compile

Do not compile Python source files to bytecode

(environment variable: PIP_NO_COMPILE)

--no-warn-script-location

Do not warn when installing scripts outside PATH

(environment variable: PIP_NO_WARN_SCRIPT_LOCATION)

--no-warn-conflicts

Do not warn about broken dependencies

(environment variable: PIP_NO_WARN_CONFLICTS)

--no-binary <format_control>

Do not use binary packages. Can be supplied multiple times, and each time adds to the existing value. Accepts either ":all:" to disable all binary packages, ":none:" to empty the set (notice the colons), or one or more package names with commas between them (no colons). Note that some packages are tricky to compile and may fail to in? stall when this option is used on them.

(environment variable: PIP_NO_BINARY)

--only-binary <format_control>

Do not use source packages. Can be supplied multiple times, and each time adds to the existing value. Accepts either ":all:" to disable all source packages, ":none:" to empty the set, or one or more package names with commas between them. Packages with? out binary distributions will fail to install when this option is used on them.

(environment variable: PIP_ONLY_BINARY)

--prefer-binary

Prefer binary packages over source packages, even if the source packages are newer.

(environment variable: PIP_PREFER_BINARY)

--require-hashes

Require a hash to check each requirement against, for repeatable installs. This op?

tion is implied when any package in a requirements file has a --hash option.

(environment variable: PIP_REQUIRE_HASHES)

--progress-bar <progress_bar>

Specify whether the progress bar should be used [on, off, raw] (default: on)

(environment variable: PIP_PROGRESS_BAR)

--root-user-action <root_user_action>

Action if pip is run as a root user [warn, ignore] (default: warn)

(environment variable: PIP_ROOT_USER_ACTION)

--report <file>

Generate a JSON file describing what pip did to install the provided requirements.

Can be used in combination with --dry-run and --ignore-installed to 'resolve' the re?

quirements. When - is used as file name it writes to stdout. When writing to stdout,

please combine with the --quiet option to avoid mixing pip logging output with JSON

output.

(environment variable: PIP_REPORT)

--group <[path:]group>

Install a named dependency-group from a "pyproject.toml" file. If a path is given,

the name of the file must be "pyproject.toml". Defaults to using "pyproject.toml" in

the current directory.

(environment variable: PIP_GROUP)

--no-clean

Don't clean up build directories.

(environment variable: PIP_NO_CLEAN)

AUTHOR

pip developers

COPYRIGHT