Менеджеры пакетов и управление зависимостями в Python

1. Экосистема упаковки Python (кратко)

- PyPI — основной публичный индекс; TestPyPI — песочница.
- Форматы: wheel (*.whl) — собранный артефакт; sdist — исходники (требует сборки).
- pyproject.toml — единая точка описания сборки/метаданных (PEP 517/518).

2. pip: основы с пояснениями

Команды установки/удаления/информации:

Команда	Пояснение
python -m pip install --upgrade pip	Обновить сам pip до последней версии.
python -m pip install <package>	Установить пакет из PyPI (или другого индекса).
python -m pip uninstall <package>	Удалить установленный пакет.
python -m pip show <package>	Показать сведения: версия, зависимости, путь установки.
python -m pip list --outdated	Список пакетов, для которых доступны обновления.

Работа с индексами и прокси:

Команда	Пояснение
python -m pip install <pkg> --index-url https://pypi.org/simple	Задать основной индекс (заменить стандартный).
python -m pip install <pkg> --extra-index-url https://repo/simple	Добавить дополнительный индекс поверх основного.
python -m pip install <pkg> --trusted-host repo.local	Доверять хосту без проверки сертификата (осторожно!).
python -m pip install <pkg> --proxy http://user:pass@proxy:3128	Скачать пакет через указанный HTTP-прокси.

Спецификаторы версий (PEP 440):

Спецификатор	Что означает

==1.2.3	Точная фиксированная версия (пиннинг).
~=1.4	Совместимо с 1.4.*, т.е. >=1.4,<2.0 (минимум минорной ветки).
>=1.0,<2	Диапазон версий (интервал).
===1.2.3	Строгое сравнение без нормализации (редко применяется).
; python_version<'3.10'	Маркер окружения: условная зависимость по версии Python.

3. Виртуальные окружения и воспроизводимость

Создание и активация окружения:

Команда	Пояснение
python -m venv .venv	Создать виртуальное окружение в папке .venv.
.venv\Scripts\activate (Windows)	Активировать окружение в текущей сессии терминала.
source .venv/bin/activate (macOS/Linux)	Активировать окружение в bash/zsh.
deactivate	Деактивировать окружение и вернуться к системному Python.

Фиксация и синхронизация зависимостей:

Команда	Пояснение
python -m pip freeze > requirements.txt	Сохранить текущие версии пакетов для воспроизводимости (быстрый способ).
python -m pip install -r requirements.txt	Установить пакеты ровно по файлу requirements.txt.
pip-compile pyproject.toml -o requirements.txt	Сгенерировать «правильный» список зависимостей с пиннингом (pip-tools).

pip-sync requirements.txt	Привести окружение к точному состоянию из файла (удалит лишние пакеты).

4. Продвинутые сценарии pip

Editable install, extras и офлайн-установка:

Команда	Пояснение
python -m pip install -e .	Установка текущего проекта «в разработке» (из исходников).
python -m pip install requests[socks]	Установить пакет с дополнительными зависимостями (extras).
python -m pip download -r requirements.txt -d wheelhouse/	Предварительно скачать колёса для офлайн-установки.
python -m pip install --no-index --find-links=wheelhouse/ -r requirements.txt	Установить пакеты офлайн из локальной директории с колёсами.

Безопасность и аудит:

Команда	Пояснение
pip-compile --generate-hashes -o requirements.txt	Сгенерировать requirements с SHA-хешами для каждой версии.
python -m pip install --require-hashes -r requirements.txt	Разрешить установку только с совпадающими хешами (защита от подмены).
python -m pip install pip-audit && pip-audit	Проверить окружение на известные уязвимости пакетов.

5. Работа с терминалом и системой контроля версий GIT

В разных операционных системах процесс работы с GitHub через командную строку остаётся весьма схожим благодаря Git, который является универсальным инструментом для всех платформ. Однако есть некоторые нюансы использования Git в Windows и Raspberry Pi OS, которые стоит рассмотреть более подробно.

Команды Git в Windows

В Windows для работы с Git часто используют Git Bash, терминал, который входит в стандартный пакет установки Git для Windows. Git Bash эмулирует bash-оболочку, характерную для Linux-систем, что делает работу с Git в Windows удобной и привычной для тех, кто знаком с Linux или macOS.

Основные команды Git в Git Bash для Windows:

git clone [URL] Клонирование удалённого репозитория по указанному URL на ваш компьютер.
git status Показывает состояние рабочего каталога и индекса: изменённые, новые или удалённые файлы.
git add [file] / git add . Добавление файлов в индекс для коммита.
- git add file.txt — добавит один файл.
- git add . — добавит все изменённые файлы.
git commit -m "комментарий" Создание коммита с сообщением.
git remote add origin [URL] Привязка локального репозитория к удалённому.
git push Отправка изменений в удалённый репозиторий. При первом пуше: git push -u origin main указываем нужную ветку удаленного репозитория
git pull Получение и слияние изменений из удалённого репозитория. Это необходимо делать каждый раз когда вы осуществляете совместную работу над одним репозиторием
git branch Просмотр всех веток.
- git branch new_branch — создать новую ветку.
git checkout [branch] Переключение на другую ветку.
- git checkout -b new_branch — создание и переход в новую ветку.
git merge [branch] Слияние указанной ветки с текущей.
git log Просмотр истории коммитов.
git diff Показать различия между рабочими файлами и индексом.
git reset [file] Убрать файл из индекса (отмена git add).
git rm [file] Удаление файла из репозитория и рабочего каталога.
git stash Временное сохранение изменений без коммита.
- git stash pop — вернуть изменения.