Concepts 概念

Tools 工具

Tools are Python packages that provide command-line interfaces.

工具是提供命令行界面的 Python 包。

> **Note 注意**
>
> See the tools guide for an introduction to working with the tools interface — this document discusses details of tool management.
>
> 有关使用工具界面的介绍，请参阅工具指南 — 本文档讨论了工具管理的详细信息。

The `uv tool` interface
`uv 工具`界面

uv includes a dedicated interface for interacting with tools. Tools can be invoked without installation using `uv tool run`, in which case their dependencies are installed in a temporary virtual environment isolated from the current project.

UV 包括一个用于与工具交互的专用界面。无需使用 `uv tool run` 进行安装即可调用工具，在这种情况下，它们的依赖项将安装在与当前项目隔离的临时虚拟环境中。

Because it is very common to run tools without installing them, a `uvx` alias is provided for `uv tool run` — the two commands are exactly equivalent. For brevity, the documentation will mostly refer to `uvx` instead of `uv tool run`.

因为在不安装工具的情况下运行工具是很常见的，所以 为 `uv tool run` — 这两个命令完全等价。为简洁起见，文档将主要引用 `uvx` 而不是 `uv 工具运行` 。

Tools can also be installed with `uv tool install`, in which case their executables are available on the `PATH` — an isolated virtual environment is still used, but it is not removed when the command completes.

工具也可以使用 `uv tool install` 安装，在这种情况下，它们的可执行文件是 在 `PATH` 上可用 — 仍使用隔离的虚拟环境，但在命令完成时不会将其删除。

Execution vs installation
执行与安装

In most cases, executing a tool with `uvx` is more appropriate than installing the tool. Installing the tool is useful if you need the tool to be available to other programs on your system, e.g., if some script you do not control requires the tool, or if you are in a Docker image and want to make the tool available to users.

在大多数情况下，使用 `uvx` 执行工具比安装工具更合适。如果您需要该工具可供系统上的其他程序使用，例如，如果您无法控制的某些脚本需要该工具，或者如果您处于 Docker 映像中并希望使该工具可供用户使用，则安装该工具非常有用。

Tool environments 工具环境

When running a tool with `uvx`, a virtual environment is stored in the uv cache directory and is treated as disposable, i.e., if you run `uv cache clean` the environment will be deleted. The environment is only cached to reduce the overhead of repeated invocations. If the environment is removed, a new one will be created automatically.

当使用 `uvx` 运行工具时，虚拟环境存储在 uv 缓存目录中，并被视为一次性的，即如果您运行 `uv 缓存清理` ，则环境将被删除。缓存环境只是为了减少重复调用的开销。如果删除环境，将自动创建一个新环境。

When installing a tool with `uv tool install`, a virtual environment is created in the uv tools directory. The environment will not be removed unless the tool is uninstalled. If the environment is manually deleted, the tool will fail to run.

使用 `uv tool install` 安装工具时，会在 uv tools 目录中创建一个虚拟环境。除非卸载该工具，否则不会删除环境。如果手动删除环境，则该工具将无法运行。

Tool versions 工具版本

Unless a specific version is requested, `uv tool install` will install the latest available of the requested tool. `uvx` will use the latest available version of the requested tool *on the first invocation*. After that, `uvx` will use the cached version of the tool unless a different version is requested, the cache is pruned, or the cache is refreshed.

除非请求特定版本，否则 `uv tool install` 将安装所请求工具的最新可用版本。`UVX` 将在*第一次调用时*使用所请求工具的最新可用版本。之后，`uvx` 将使用该工具的缓存版本，除非请求不同的版本、修剪缓存或刷新缓存。

For example, to run a specific version of Ruff:

例如，要运行特定版本的 Ruff，请执行以下作：

```
$ uvx ruff@0.6.0 --version
ruff 0.6.0
```

A subsequent invocation of `uvx` will use the latest, not the cached, version.

随后调用 `uvx` 将使用最新版本，而不是缓存版本。

```
$ uvx ruff --version
ruff 0.6.2
```

But, if a new version of Ruff was released, it would not be used unless the cache was refreshed.

但是，如果发布了新版本的 Ruff，除非刷新缓存，否则不会使用它。

To request the latest version of Ruff and refresh the cache, use the `@latest` suffix:

要请求最新版本的 Ruff 并刷新缓存，请使用 `@latest` 后缀：

```
$ uvx ruff@latest --version
0.6.2
```

Once a tool is installed with `uv tool install`, `uvx` will use the installed version by default.

使用 `uv tool install` 安装工具后，`uvx` 将默认使用已安装的版本。

For example, after installing an older version of Ruff:

例如，安装旧版本的 Ruff 后：

```
$ uv tool install ruff==0.5.0
```

The version of `ruff` and `uvx ruff` is the same:

`ruff` 和 `uvx ruff` 的版本是相同的：

```
$ ruff --version
ruff 0.5.0
$ uvx ruff --version
ruff 0.5.0
```

However, you can ignore the installed version by requesting the latest version explicitly, e.g.:

但是，您可以通过显式请求最新版本来忽略已安装的版本，例如：

```
$ uvx ruff@latest --version
0.6.2
```

Or, by using the `--isolated` flag, which will avoid refreshing the cache but ignore the installed version:

或者，通过使用 `--isorated` 标志，这将避免刷新缓存，但忽略已安装的版本：

```
$ uvx --isolated ruff --version
0.6.2
```

`uv tool install` will also respect the `{package}@{version}` and `{package}@latest` specifiers, as in:

`uv tool install` 还将遵循 `{package}@{version}` 和 `{package}@latest` 说明符，如：

```
$ uv tool install ruff@latest
$ uv tool install ruff@0.6.0
```

Tools directory 工具目录

By default, the uv tools directory is named `tools` and is in the uv application state directory, e.g., `~/.local/share/uv/tools`. The location may be customized with the `UV_TOOL_DIR` environment variable.

默认情况下，uv tools 目录名为 `tools`，位于 uv 应用程序状态目录中，例如 `~/.local/share/uv/tools`。可以使用 `UV_TOOL_DIR` 环境变量自定义位置。

To display the path to the tool installation directory:

要显示工具安装目录的路径，请执行以下作：

```
$ uv tool dir
```

Tool environments are placed in a directory with the same name as the tool package, e.g., `.../tools/<name>`.

工具环境放置在与工具包同名的目录中，例如 `.../tools/<name>`。

> **Important 重要**
>
> Tool environments are *not* intended to be mutated directly. It is strongly recommended never to mutate a tool environment manually, e.g., with a `pip` operation.
>
> 工具环境*不打算*直接更改。强烈建议永远不要手动更改工具环境，例如，使用 `pip` 作。

Upgrading tools 升级工具

Tool environments may be upgraded via `uv tool upgrade`, or re-created entirely via subsequent `uv tool install` operations.

工具环境可以通过 `uv 工具升级` 进行升级，也可以通过后续完全重新创建 `uv 工具安装` 作。

To upgrade all packages in a tool environment

升级工具环境中的所有包

```
$ uv tool upgrade black
```

To upgrade a single package in a tool environment:

要升级工具环境中的单个包：

```
$ uv tool upgrade black --upgrade-package click
```

Tool upgrades will respect the version constraints provided when installing the tool. For example, `uv tool install black >=23,<24` followed by `uv tool upgrade black` will upgrade Black to the latest version in the range `>=23,<24`.

工具升级将遵循安装工具时提供的版本约束。例如 `uv tool install black >=23,<24` 随后是 `uv 工具升级` 黑色将 Black 升级到 `>=23, <24` 范围内的最新版本。

To instead replace the version constraints, reinstall the tool with `uv tool install`:

要替换版本约束，请使用 `uv tool install` 重新安装该工具：

```
$ uv tool install black>=24
```

Similarly, tool upgrades will retain the settings provided when installing the tool. For example, `uv tool install black --prerelease allow` followed by `uv tool upgrade black` will retain the `--prerelease allow` setting.

同样，工具升级将保留安装工具时提供的设置。例如 `uv tool install black --prerelease allow` 随后的 `UV 工具升级` 黑色将保留 `--prerelease allow` 设置。

> **Note 注意**
>
> Tool upgrades will reinstall the tool executables, even if they have not changed.
>
> 工具升级将重新安装工具可执行文件，即使它们没有更改。

To reinstall packages during upgrade, use the `--reinstall` and `--reinstall-package` options.

要在升级期间重新安装包，请使用 `--reinstall` 和 `--reinstall-package` 选项。

To reinstall all packages in a tool environment

在工具环境中重新安装所有包

```
$ uv tool upgrade black --reinstall
```

To reinstall a single package in a tool environment:

要在工具环境中重新安装单个包：

```
$ uv tool upgrade black --reinstall-package click
```

Including additional dependencies
包括其他依赖项

Additional packages can be included during tool execution:

在工具执行过程中可以包含其他包：

```
$ uvx --with <extra-package> <tool>
```

And, during tool installation:

并且，在工具安装过程中：

```
$ uv tool install --with <extra-package> <tool-package>
```

The `--with` option can be provided multiple times to include additional packages.

可以多次提供 `--with` 选项以包含其他包。

The `--with` option supports package specifications, so a specific version can be requested:

`--with` 选项支持软件包规范，因此可以请求特定版本：

```
$ uvx --with <extra-package>==<version> <tool-package>
```

The `-w` shorthand can be used in place of the `--with` option:

可以使用 `-w` 简写来代替 `--with` 选项：

```
$ uvx -w <extra-package> <tool-package>
```

If the requested version conflicts with the requirements of the tool package, package resolution will fail and the command will error.

如果请求的版本与工具包的要求冲突，则包解析将失败，命令将出错。

Python versions Python 版本

Each tool environment is linked to a specific Python version. This uses the same Python version discovery logic as other virtual environments created by uv, but will ignore non-global Python version requests like `.python-version` files and the `requires-python` value from a `pyproject.toml`.

每个工具环境都链接到特定的 Python 版本。这使用相同的 Python 版本 发现逻辑与 uv 创建的其他虚拟环境一样，但将忽略非全局 Python 版本请求，例如 `.python-version` 文件和来自 `pyproject.toml` 的 `requires-python` 值。

The `--python` option can be used to request a specific version. See the Python version documentation for more details.

`--python` 选项可用于请求特定版本。请参阅 Python 版本文档了解更多详情。

If the Python version used by a tool is *uninstalled*, the tool environment will be broken and the tool may be unusable.

如果卸*载*工具使用的 Python 版本，工具环境将被破坏，工具可能无法使用。

Tool executables 工具可执行文件

Tool executables include all console entry points, script entry points, and binary scripts provided by a Python package. Tool executables are symlinked into the `bin` directory on Unix and copied on Windows.

工具可执行文件包括 Python 包提供的所有控制台入口点、脚本入口点和二进制脚本。工具可执行文件在 Unix 上符号链接到 `bin` 目录中，并在 Windows 上复制。

The `bin` directory
`bin` 目录

Executables are installed into the user `bin` directory following the XDG standard, e.g., `~/.local/bin`. Unlike other directory schemes in uv, the XDG standard is used on *all platforms* notably including Windows and macOS — there is no clear alternative location to place executables on these platforms. The installation directory is determined from the first available environment variable:

可执行文件按照 XDG 标准安装到用户 `bin` 目录中，例如 `~/.local/bin` 中。与 uv 中的其他目录方案不同，XDG 标准适用于*所有平台* 特别是包括 Windows 和 macOS——没有明确的替代位置来放置可执行文件 这些平台。安装目录是从第一个可用环境确定的 变量：

- `$UV_TOOL_BIN_DIR`
- `$XDG_BIN_HOME`
- `$XDG_DATA_HOME/../bin`
- `$HOME/.local/bin`

Executables provided by dependencies of tool packages are not installed.

不安装工具包的依赖项提供的可执行文件。

The `PATH` 路径

The `bin` directory must be in the `PATH` variable for tool executables to be available from the shell. If it is not in the `PATH`, a warning will be displayed. The `uv tool update-shell` command can be used to add the `bin` directory to the `PATH` in common shell configuration files.

`bin` 目录必须位于 `PATH` 变量中，以便工具可执行文件才能从 shell 中使用。如果它不在 `PATH` 中，则会显示警告。`uv tool update-shell` 命令可用于将 `bin` 目录添加到常见 shell 配置文件中的 `PATH` 中。

Overwriting executables 覆盖可执行文件

Installation of tools will not overwrite executables in the `bin` directory that were not previously installed by uv. For example, if `pipx` has been used to install a tool, `uv tool install` will fail. The `--force` flag can be used to override this behavior.

安装工具不会覆盖 `bin` 目录中以前未由 uv 安装的可执行文件。例如，如果已使用 `pipx` 安装工具，则 `uv 工具安装` 将失败。`--force` 标志可用于覆盖此行为。

Relationship to `uv run`
与 `紫外线运行` 的关系

The invocation `uv tool run <name>` (or `uvx <name>`) is nearly equivalent to:

调用 `uv 工具运行 <name>` （或 `uvx <name>`） 几乎等同于：

```
$ uv run --no-project --with <name> -- <name>
```

However, there are a couple notable differences when using uv's tool interface:

但是，使用 uv 的工具界面时有几个显着差异：

- The `--with` option is not needed — the required package is inferred from the command name.

  不需要 `--with` 选项 — 所需的包是从命令名称推断出来的。

- The temporary environment is cached in a dedicated location.

  临时环境缓存在专用位置。

- The `--no-project` flag is not needed — tools are always run isolated from the project.

  不需要 `--no-project` 标志 — 工具始终与项目隔离运行。

- If a tool is already installed, `uv tool run` will use the installed version but `uv run` will not.

  如果已安装工具，则 `uv tool run` 将使用已安装的版本，但 `uv run` 不会。

If the tool should not be isolated from the project, e.g., when running `pytest` or `mypy`, then `uv run` should be used instead of `uv tool run`.

如果该工具不应与项目隔离，例如，在运行 `pytest` 或 `mypy` 时，则 应使用 `UV 运行（UV Run）` 而不是 `UV 工具运行（UV Tool Run）`。