Get started with Magic

Magic is a package manager and virtual environment manager for any language, including Python and Mojo. It builds upon the conda and PyPI packaging ecosystems, which provide access to thousands of packages for Python and other languages, while also adding functionality for MAX and Mojo.

The `magic` CLI allows you to instantly launch code examples and create new projects that are fully contained and reproducible across systems. All the package dependencies and environment settings are magically managed for you.

This page provides an introduction to basic `magic` commands. For a deep-dive into more features, see the Magic tutorial.

> **Note**
>
> Magic is built upon pixi, so you'll see this name appear below.

Install Magic

You can install Magic on macOS and Ubuntu with this command:

```
$ curl -ssL https://magic.modular.com | bash
```

Then run the `source` command that's printed in your terminal.

To see the available commands, print the help:

```
$ magic --help
```

Enable auto-completion

To enable auto-completion for `magic`, run the following commands:

Bash ZSH fish

```
$ BASHRC=$( [ -f "$HOME/.bash_profile" ] && echo "$HOME/.bash_profile" || echo
"$HOME/.bashrc" )
echo 'eval "$(magic completion --shell bash)"' >> "$BASHRC"
source "$BASHRC"
```

Update Magic

You can update with the `self-update` command:

```
$ magic self-update
```

Uninstall Magic

To remove Magic, delete the binary:

```
$ rm ~/.modular/bin/magic
```

To remove packages installed for your projects, delete the corresponding project directories.

Create a project

You can create a project with its own package dependencies and virtual environment using the `magic init` command.

By default, this creates a configuration file called `pixi.toml`, but you can also create a `pyproject.toml` or `mojoproject.toml` file for enhanced Python and Mojo project features, respectively.

Create a Python project

Here's how to create a new Python project and install MAX:

1. Create a Python project with the `magic init` command:

   ```
   $ magic init my-project
   ```

   This creates a `my-project` directory and a `pixi.toml` file that defines the project dependencies and more. (If you omit the directory name, it creates the `pixi.toml` file in the current directory.)

2. Enter the project and use `magic add` to set the Python version:

```
$ cd my-project
```

```
$ magic add "python==3.11"
```

3. Use `magic run` to execute code inside the virtual environment:

```
$ magic run python3 --version
```

   Or, activate the environment shell with `magic shell`:

```
$ magic shell
```

```
$ python3 --version
```

   Then use `exit` to deactivate the shell:

```
$ exit
```

   Always exit the shell before changing projects.

4. To install Python packages for your project, use `magic add`:

```
$ magic add max
```

You can run commands such as `magic add` anywhere inside a Magic project directory, whether or not you've activated the shell.

For more information about using Magic for your Python projects, read using Pixi for Python (just replace each `pixi` command with `magic`).

> **Caution**
>
> `magic` also supports `pyproject.toml` files created with `magic init --format pyproject` but it currently fails when you run `magic shell`. We suggest you instead use `pixi.toml` as shown above for now.

Create a Mojo project

A Mojo project is defined with either a `pixi.toml` or `mojoproject.toml` file. Currently, these files behave the same, but the `mojoproject.toml` file allows us to add support for Mojo-specific features in the future.

1. Create a Mojo project with `magic init`:

   ```
   $ magic init my-mojo-project --format mojoproject
   ```

   This creates the `my-mojo-project` directory and creates a `mojoproject.toml` file inside, which defines the project dependencies and more. If you omit the path name, Magic creates the confg file in the current directory.

   By default, `mojoproject.toml` includes `max` as a dependency because `max` is the package that installs Mojo.

2. Enter the project and use `magic run` to execute code inside the environment:

   ```
   $ cd my-mojo-project
   ```

   ```
   $ magic run mojo --version
   ```

   Or, activate the environment shell with `magic shell`:

   ```
   $ magic shell
   ```

   ```
   $ mojo --version
   ```

   Then use `exit` to deactivate the shell:

   ```
   $ exit
   ```

   Always exit the shell before changing projects.

3. If you want to use Python with Mojo, specify the Python version and Python packages with `magic add`. For example:

   ```
   $ magic add "python==3.9"
   ```

You can run commands such as `magic add` anywhere inside a Magic project directory, whether or not you've activated the shell.

Convert a conda project to Magic

If you have an existing conda project, you can convert the `environment.yml` configuration file to Magic with this command:

```
$ magic init --import environment.yml
```

> **Caution**
>
> You might encounter issues if you invoke `magic` within a `conda` virtual environment. It's best if you don't mix the two tools.

Manage packages

You can add Python packages to your project by running `magic add` inside your project directory (every project has its own package versions). For example:

```
$ magic add max "numpy<2.0"
```

This adds the latest version of MAX and the latest version of NumPy that's less than `2.0`.

By default, Magic looks for packages in the `conda-forge` repository (as specified by your config file `channels`).

If you prefer to use PyPI for your packages, add the `--pypi` flag:

```
$ magic add --pypi "numpy<2.0"
```

However, you should avoid mixing conda and PyPi packages in the same project.

Add channels

If a package appears unavailable from conda via `magic add`, you might need to add a new conda channel. For example, PyTorch is not available in `conda-forge`, so you must edit your `pixi.toml` or `mojoproject.toml` config file to add the `"pytorch"` channel:

```
channels = ["pytorch", "conda-forge", "https://conda.modular.com/max"]
```

Then you can install the latest PyTorch as a conda package:

```
$ magic add pytorch
```

> **Note**
>
> You can also use `magic project channel add` to add a channel to your config file, but it adds the channel at the end of the list, which means it is the lowest channel priority.

Update a package

To update a package, use `magic update` within your project path:

```
$ magic update max
```

If there's a new version of MAX, this updates the config file and installs it.

Define a package version

Even after you've added a package, you can run `magic add` again with a new version specifier. For example:

```
$ magic add "max~=24.5"
```

This ensures that the project uses MAX `24.5` but it also allows "compatible" versions such as `24.5.1`.

For more information, read about Pixi dependency tables.

Change the Python version

You can also use `magic add` to change your Python version with a version specifier. For example:

```
$ magic add "python==3.9"
```

The next time you run a `magic` command, it updates Python with the appropriate version:

```
$ magic run python3 --version
Python 3.9.0
```

The `magic.lock` file

Although the project configuration file (`pixi.toml`, `pyproject.toml`, or `mojoproject.toml`) defines your project dependencies, it doesn't define the project's transitive dependencies, which are the dependencies of your project dependencies. Nor does the configuration file always specify the exact package version that is actually installed (such as when you specify a version merely as less-than `<` or greater-than `>`).

The transitive dependencies and actual installed versions are instead specified in the `magic.lock` file, which is automatically generated—you should not edit this file by hand.

This file is crucial to ensure that you can reliably reproduce your environment across different machines. You can learn more about it from the Pixi lock file docs.

Known issues

- You might encounter issues if you invoke `magic` within a `conda` or `venv` virtual environment. It's best if you don't mix Magic with other virtual environment tools.
- If you also have `pixi` installed, it generally should work with projects you created using `magic`, but you might see some issues so we advise you only use `magic` for MAX and Mojo projects.
- Linux aarch64 (ARM64) does not work with projects using PyTorch 2.2.2.
- The MAX Replit pipeline currently doesn't work with the max-conda package.

More reading

You can learn more about the available commands by printing the help:

```
$ magic -h
```

Or see all the Magic commands here.

If you have more questions, see the Magic FAQ.

And because Magic is built upon `pixi`, you can also learn more from the pixi documentation (just replace each `pixi` command with `magic`). However, there are several differences between `magic` and `pixi`. For example, `magic` does not support `exec`, `auth`, and `upload` commands, and probably others to come.

Get started with MAX

Run a PyTorch model on MAX Engine and run Llama 3 built entirely in Mojo.

Get started with Mojo

Get started in Mojo by writing "Hello world" using the Mojo REPL and compiler.

A step-by-step guide to Magic

Learn how to get started and get the most out of the Magic.

Was this page helpful?