Corepack

> *Stability: 1 - Experimental*

Corepack is an experimental tool to help with managing versions of your package managers. It exposes binary proxies for each supported package manager that, when called, will identify whatever package manager is configured for the current project, transparently install it if needed, and finally run it without requiring explicit user interactions.

This feature simplifies two core workflows:

- It eases new contributor onboarding, since they won't have to follow system-specific installation processes anymore just to have the package manager you want them to.
- It allows you to ensure that everyone in your team will use exactly the package manager version you intend them to, without them having to manually synchronize it each time you need to make an update.

Workflows

Enabling the feature

Due to its experimental status, Corepack currently needs to be explicitly enabled to have any effect. To do that simply run `corepack enable`, which will set up the symlinks in your environment, next to the `node` binary (and overwrite the existing symlinks if necessary).

From this point forward, any call to the supported binaries will work without further setup. Should you experience a problem, just run `corepack disable` to remove the proxies from your system (and consider opening up an issue on the Corepack repository to let us know).

Configuring a package

The Corepack proxies will find the closest `package.json` file in your current directory hierarchy to extract its `"packageManager"` property.

If the value corresponds to a supported package manager, Corepack will make sure that all calls to the relevant binaries are run against the requested version, downloading it on demand if needed, and aborting if it cannot be successfully retrieved.

Upgrading the global versions

When running outside of an existing project (for example when running `yarn init`), Corepack will by default use predefined versions roughly corresponding to the latest stable releases from each tool. Those versions can be easily overriden by running the `corepack prepare` command along with the package manager version you wish to set:

```
corepack prepare yarn@x.y.z --activate
```

Offline workflow

Many production environments don't have network access. Since Corepack usually downloads the package manager releases straight from their registries, it can conflict with such environments. To avoid that happening, call the `corepack prepare` command while you still have network access (typically at the same time you're preparing your deploy image). This will ensure that the required package managers are available even without network access.

The `prepare` command has various flags, consult the detailed Corepack documentation for more information on the matter.

Supported package managers

The following binaries are provided through Corepack:

Package manager	Binary names
Yarn	`yarn`, `yarnpkg`
pnpm	`pnpm`, `pnpx`

Common questions

How does Corepack currently interact with npm?

While Corepack could easily support npm like any other package manager, its shims aren't currently enabled by default. This has a few consequences:

- It's always possible to run a `npm` command within a project configured to be used with another package manager, since Corepack cannot intercept it.
- While `npm` is a valid option in the `"packageManager"` property, the lack of shim will cause the global npm to be used.

Running `npm install -g yarn` doesn't work

npm prevents accidentally overriding the Corepack binaries when doing a global install. To avoid this problem, consider one of the following options:

- Don't run this command anymore; Corepack will provide the package manager binaries anyway and will ensure that the requested versions are always available, so installing the package managers explicitly isn't needed anymore.
- Add the `--force` flag to `npm install`; this will tell npm that it's fine to override binaries, but you'll erase the Corepack ones in the process (should that happen, run `corepack enable` again to add them back).