corepack

Corepack is a zero-runtime-dependency Node.js script that acts as a bridge between Node.js projects and the package managers they are intended to be used with during development. In practical terms, **Corepack will let you use Yarn and pnpm without having to install them** - just like what currently happens with npm, which is shipped by Node.js by default.

Important: At the moment, Corepack only covers Yarn and pnpm. Given that we have little control on the npm project, we prefer to focus on the Yarn and pnpm use cases. As a result, Corepack doesn't have any effect at all on the way you use npm.

How to Install

Default Installs

Corepack is distributed by default with Node.js 16.9, but is opt-in for the time being. Run `corepack enable` to install the required shims.

Manual Installs

Click here to see how to install Corepack using npm

First uninstall your global Yarn and pnpm binaries (just leave npm). In general, you'd do this by running the following command:

```
npm uninstall -g yarn pnpm

# That should be enough, but if you installed Yarn without going through npm it might
# be more tedious - for example, you might need to run `brew uninstall yarn` as well.
```

Then install Corepack:

```
npm install -g corepack
```

We do acknowledge the irony and overhead of using npm to install Corepack, which is at least part of why the preferred option is to use the Corepack version that is distributed along with Node.js itself.

Usage

Just use your package managers as you usually would. Run `yarn install` in Yarn projects, `pnpm install` in pnpm projects, and `npm` in npm projects. Corepack will catch these calls, and depending on the situation:

- **If the local project is configured for the package manager you're using**, Corepack will silently download and cache the latest compatible version.

- **If the local project is configured for a different package manager**, Corepack will request you to run the command again using the right package manager - thus avoiding corruptions of your install artifacts.
- **If the local project isn't configured for any package manager**, Corepack will assume that you know what you're doing, and will use whatever package manager version has been pinned as "known good release". Check the relevant section for more details.

Known Good Releases

When running Yarn or pnpm within projects that don't list a supported package manager, Corepack will default to a set of Known Good Releases. In a way, you can compare this to Node.js, where each version ships with a specific version of npm.

The Known Good Releases can be updated system-wide using the `--activate` flag from the `corepack prepare` and `corepack hydrate` commands.

Offline Workflow

The utility commands detailed in the next section.

- Either you can use the network while building your container image, in which case you'll simply run `corepack prepare` to make sure that your image includes the Last Known Good release for the specified package manager.
  - If you want to have *all* Last Known Good releases for all package managers, just use the `--all` flag which will do just that.
- Or you're publishing your project to a system where the network is unavailable, in which case you'll preemptively generate a package manager archive from your local computer (using `corepack prepare -o`) before storing it somewhere your container will be able to access (for example within your repository). After that it'll just be a matter of running `corepack hydrate <path/to/corepack.tgz>` to setup the cache.

Utility Commands

`corepack <binary name>[@<version>] [... args]`

This meta-command runs the specified package manager in the local folder. You can use it to force an install to run with a given version, which can be useful when looking for regressions.

Note that those commands still check whether the local project is configured for the given package manager (ie you won't be able to run `corepack yarn install` on a project where the `packageManager` field references `pnpm`).

`corepack enable [... name]`

Option	Description
`--install-directory`	Add the shims to the specified location

This command will detect where Node.js is installed and will create shims next to it for each of the specified package managers (or all of them if the command is called without parameters). Note that the npm shims will not be installed unless explicitly requested, as npm is currently distributed with Node.js through other means.

`corepack disable [... name]`

Option	Description
`--install-directory`	Remove the shims to the specified location

This command will detect where Node.js is installed and will remove the shims from there.

`corepack prepare [... name@version]`

Option	Description
`--all`	Prepare the "Last Known Good" version of all supported package managers
`-o,--output`	Also generate an archive containing the package managers
`--activate`	Also update the "Last Known Good" release

This command will download the given package managers (or the one configured for the local project if no argument is passed in parameter) and store it within the Corepack cache. If the `-o,--output` flag is set (optionally with a path as parameter), an archive will also be generated that can be used by the `corepack hydrate` command.

`corepack hydrate <path/to/corepack.tgz>`

Option	Description
`--activate`	Also update the “Last Known Good” release

This command will retrieve the given package manager from the specified archive and will install it within the Corepack cache, ready to be used without further network interaction.

Environment Variables

- `COREPACK_ROOT` has no functional impact on Corepack itself; it's automatically being set in your environment by Corepack when it shells out to the underlying package managers, so that they can feature-detect its presence (useful for commands like `yarn init`).

Contributing

If you want to build corepack yourself, you can build the project like this:

1. Clone this repository
2. Run `yarn build` (no need for `yarn install`)
3. The `dist/` directory now contains the corepack build and the shims
4. Call `node ./dist/corepack --help` and behold

You can also run the tests with `yarn jest` (still no install needed).

Design

Various tidbits about Corepack's design are explained in more details in DESIGN.md.

License (MIT)