Managing Packages

- Internal workspaces (tools and tests)
- Supply chain policy

This repository uses npm workspaces to manage WordPress packages and lerna to publish them to npm. This enforces certain steps in the workflow which are described in details in packages documentation.

Add a short entry to the package's `CHANGELOG.md` for changes that affect consumers. Follow Maintaining Changelogs for scope and formatting.

Publishing WordPress packages to npm is automated by synchronizing it with the bi-weekly Gutenberg plugin RC1 release. You can learn more about this process and other ways to publish new versions of npm packages in the Gutenberg Release Process document.

Internal workspaces (tools and tests)

The repository also contains internal workspaces under `tools/` and `test/` for development tooling and test infrastructure. When you need to add a new tool, script, or dependency for repo-level work, create a workspace under `tools/` (or add to an existing one) instead of adding dependencies to the root `package.json`. See the Workspace Development guide for the conversion pattern, CI conventions, and reference examples.

Supply chain policy

`.npmrc` opts in to the npm v12 defaults that refuse git references (`EALLOWGIT`) and tarball URLs (`EALLOWREMOTE`), and extends that to local tarball files (`EALLOWFILE`). Local directories stay at the npm default, which gates nothing here: the `file:` links between workspaces resolve as workspaces rather than directory dependencies.

Install scripts are opt-in: every dependency that ships one is recorded in `allowScripts` in the root `package.json`, and `strict-allow-scripts` fails the install with `ESTRICTALLOWSCRIPTS` on anything missing from that list. Every entry is `false`, so nothing compiles on install. `test/ai-development` installs with `--prefix`, so it carries its own copy of the policy.

`allowScripts` covers dependencies only, so workspace lifecycle scripts such as the `@wordpress/icons` `prepare` still run. npm v12 skips them under `install-strategy=linked` with no error (npm/cli#9982), so moving to npm v12 means invoking them from the root `postinstall` instead.

When an install fails that way, read the script, then edit `allowScripts` by hand and commit the `package.json` change. Use the package name on its own to cover every version, or `name@version` to pin an approval to the version you reviewed:

```
"allowScripts": {
    "some-package": false,
    "another-package@1.2.3": true
}
```

Do not use the `npm install-scripts` subcommands until an npm release ships npm/cli#9941. All of them read the hoisted layout, so under `install-strategy=linked` they misbehave: `ls` reports covered packages as uncovered, `prune` deletes every entry as unused, and `approve` and `deny` write unusable `node_modules/.store` paths instead of the package name.

□□□ #2
□ □□□ □□□ 2026-10-31 18:25:32 UTC
□ □□□ □□□ 2026-10-31 18:25:33 UTC