yarn

By Lucía Blanco, Pablo Fernández, Ángel Olmedo and Raúl Pérez

What is yarn ?

(A brief introduction)

How does it work?

RESOLUTION

Petitions to the registry and check dependencies in the directory.

FLETCHING

Check if dependency is already in cache’s global directory.

LINKING

Copy cache to node_modules in local directory.

Stakeholders

- Software Architects
- Development Teams
- Project Managers
- Companies (StackShare, Docplanner...)

Basically, **anyone** who works with dependencies!

Quality Goals

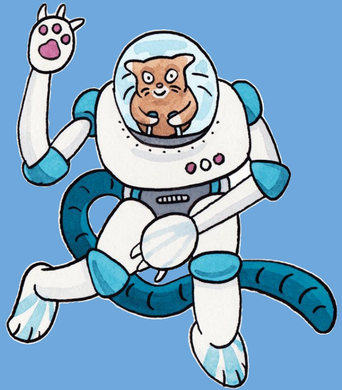
Performance

- Offline caché
- Multithreading
- Zero Installs
- Network Performance

Reliability

- Detailed but concise lockfile format
- The same dependencies will be installed in the same exact way on any machine, regardless of installation order
- Network resilience

Security



- Checksums to verify the integrity of every installed package before its code is executed.

Usability

- Simple commands
- Per-Project installation
- Plug’n’Play feature

Constraints

Prior Art

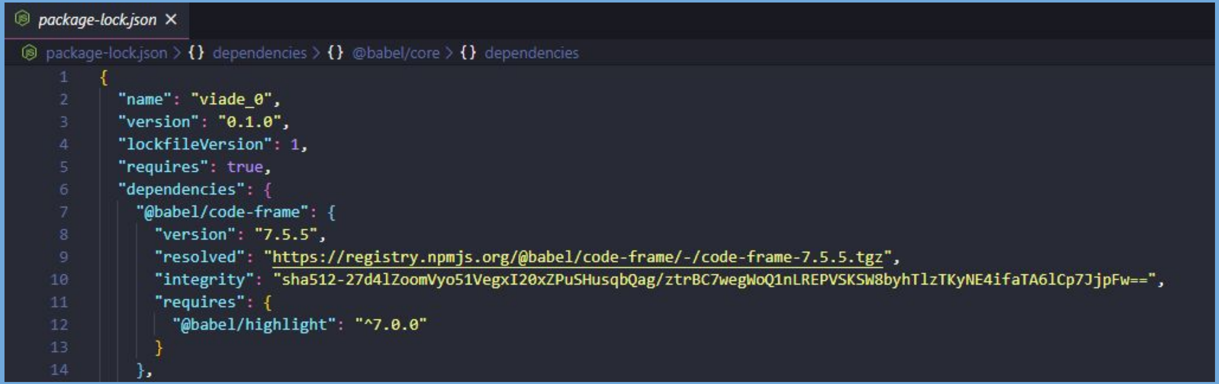
- Bundler
- Cargo
- npm

JavaScript

- Programming decisions.
- Follow language specific coding styles.

Lockfile

Supporting lockfiles has its tradeoffs



package-lock.json ×

package-lock.json > {} dependencies > {} @babel/core > {} dependencies

```json
{
  "name": "viade_0",
  "version": "0.1.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "@babel/code-frame": {
      "version": "7.5.5",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.5.5.tgz",
      "integrity": "sha512-27d4lZoomVyo51VegxI20xZPuSHusqbQag/ztrBC7wegWoQ1nLREPVSKSW8byhTlzTKyNE4ifaTA6lCp7JjpFw==",
      "requires": {
        "@babel/highlight": "^7.0.0"
      }
    },
```

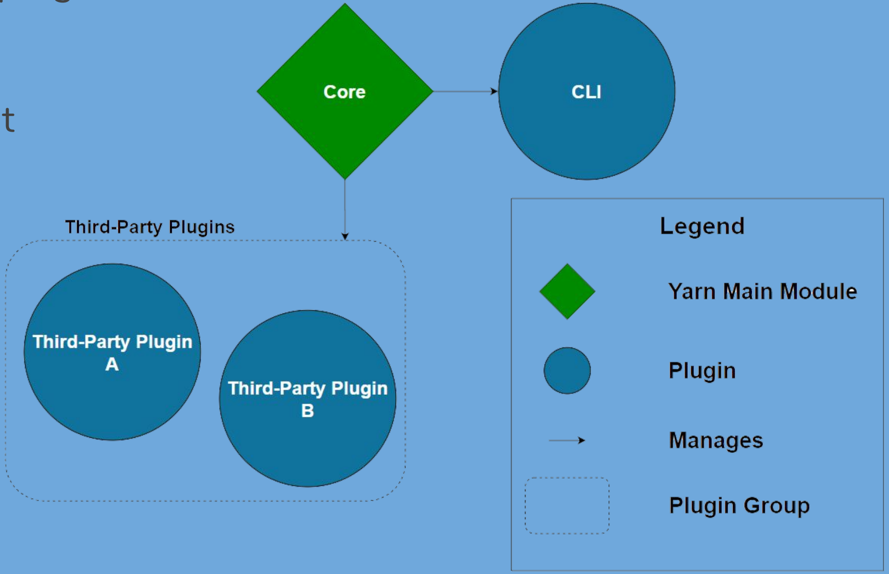
Technical Context

Basic concepts

- Dependency: Piece of software on which a project relies
- Package: A package contains all the code being shared as well as a manifest
- Manifest: File named *package.json* which describes the package

General architecture

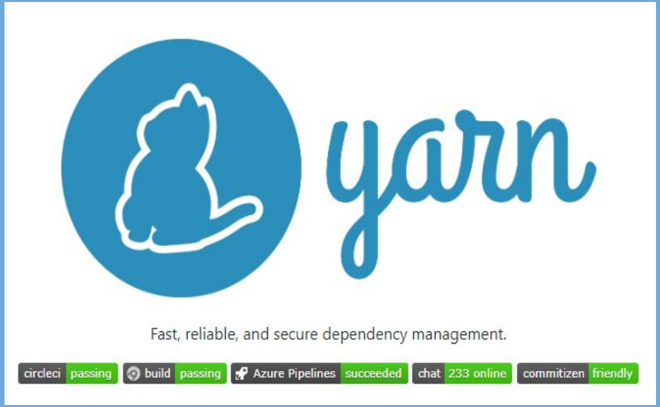
- Core: Module that contains the logic required to manage a program
  - Exposes the base components that make up a program
  - Clases from the API
- CLI: Injects prebuilt Yarn plugins into the environment
- Third-party plugins
  - Simple codebase
  - Freedom to implement external logic



Contributing to Yarn

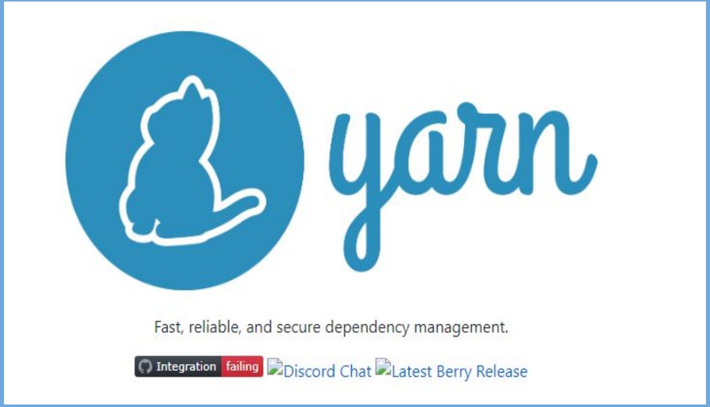
Repositories

- Github: Two different repositories



Fast, reliable, and secure dependency management.

circleci passing | build passing | Azure Pipelines succeeded | chat 233 online | commitizen friendly



Fast, reliable, and secure dependency management.

Integration failing | Discord Chat | Latest Berry Release

Repositories

Two different versions:

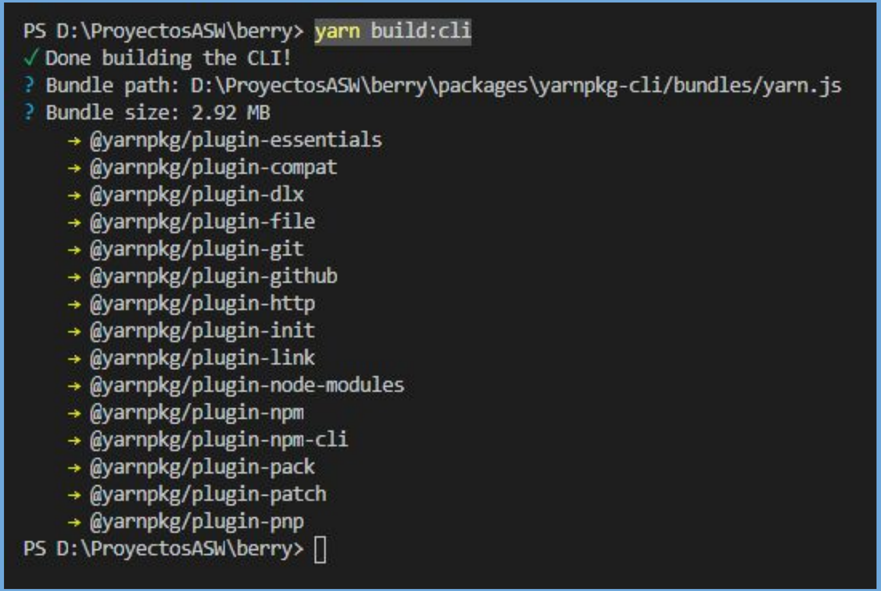
- Yarn 1.22.0 (classic): original implementation of Yarn written completely in JavaScript
- Yarn berry (version 2): modular API implementation, mainly written in TypeScript and JavaScript. Being implemented since 2019.

Building Yarn

Steps to build Yarn berry locally:

- Install Yarn
- Clone the repo
- Open local directory in an editor (VSCode)
- Run `yarn build:cli`

Building Yarn



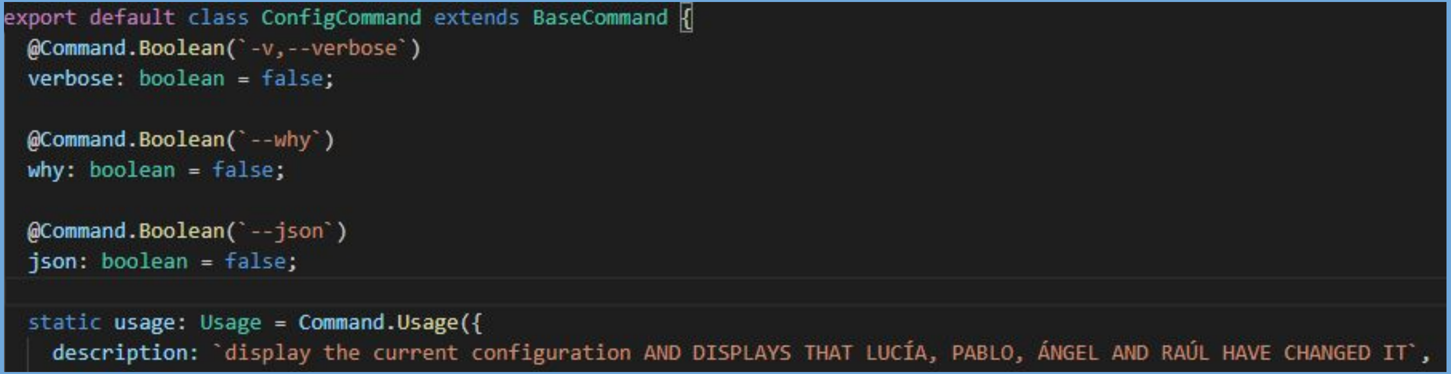
```
PS D:\ProyectosASW\berry> yarn build:cli
✓ Done building the CLI!
? Bundle path: D:\ProyectosASW\berry\packages\yarnpkg-cli/bundles/yarn.js
? Bundle size: 2.92 MB
    → @yarnpkg/plugin-essentials
    → @yarnpkg/plugin-compat
    → @yarnpkg/plugin-dlx
    → @yarnpkg/plugin-file
    → @yarnpkg/plugin-git
    → @yarnpkg/plugin-github
    → @yarnpkg/plugin-http
    → @yarnpkg/plugin-init
    → @yarnpkg/plugin-link
    → @yarnpkg/plugin-node-modules
    → @yarnpkg/plugin-npm
    → @yarnpkg/plugin-npm-cli
    → @yarnpkg/plugin-pack
    → @yarnpkg/plugin-patch
    → @yarnpkg/plugin-pnp
PS D:\ProyectosASW\berry> 
```

Building cli allows our initial global installation of yarn to reflect changes done to the code locally

Modifying a command

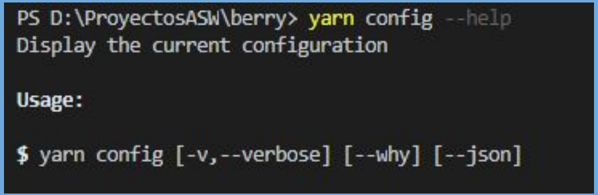
Modifying a command

- Help shown when typing yarn config --help will be modified
- Found in the file config.ts, in plugin-essentials package



```
export default class ConfigCommand extends BaseCommand {
  @Command.Boolean(`-v,--verbose`)
  verbose: boolean = false;

  @Command.Boolean(`--why`)
  why: boolean = false;

  @Command.Boolean(`--json`)
  json: boolean = false;

  static usage: Usage = Command.Usage({
    description: `display the current configuration AND DISPLAYS THAT LUCÍA, PABLO, ÁNGEL AND RAÚL HAVE CHANGED IT`,
```

Modifying a command

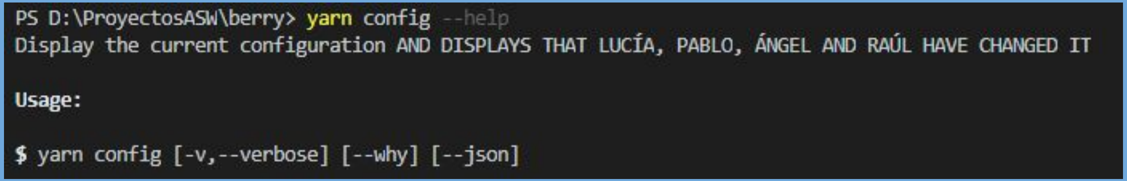


```
PS D:\ProyectosASW\berry> yarn config --help
Display the current configuration

Usage:

$ yarn config [-v,--verbose] [--why] [--json]
```

This is the normal output of running

yarn config --help

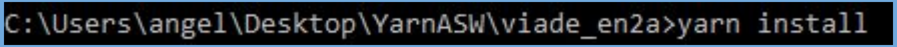
This is the output after modifying config.ts



```
PS D:\ProyectosASW\berry> yarn config --help
Display the current configuration AND DISPLAYS THAT LUCÍA, PABLO, ÁNGEL AND RAÚL HAVE CHANGED IT

Usage:

$ yarn config [-v,--verbose] [--why] [--json]
```

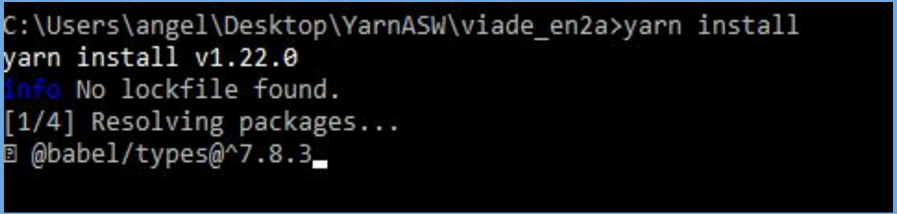
Installing Dependencies

Normal installation



```
C:\Users\angel\Desktop\YarnASW\viade_en2a>yarn install
```

Run yarn install on your project folder (package.json needed).



```
C:\Users\angel\Desktop\YarnASW\viade_en2a>yarn install
yarn install v1.22.0
info No lockfile found.
[1/4] Resolving packages...
⠁ @babel/types@^7.8.3
```

Yarn will install all the dependencies

Other options

- yarn install --flat: Install only one version of the package.
- yarn install --force: Re-download all packages.
- yarn install --production: Install only production dependencies