trainologic

MonoRepo

Zacky Pickholz

DevJam May 2018

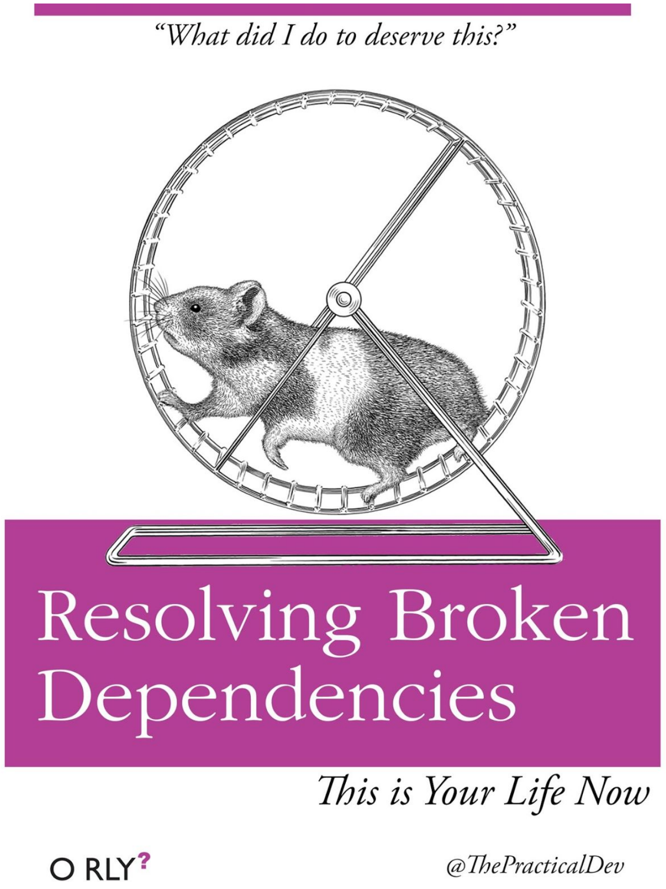
Hindsight

Agenda

- The problem
- MonoRepo & MultiRepo explained
- Which is better? Who uses what?
- npm link
- Yarn Workspaces
- Lerna
- Git Submodules
- Google Bazel
- Other tools

The Problem

- Projects tend to grow
- We split them into sub-projects
- Create separate repos
- The rat race is on


"What did I do to deserve this?"
Resolving Broken Dependencies
This is Your Life Now
O RLY?
@ThePracticalDev

The Philosophies

MonoRepo - a single multi package repository

MultiRepo - multiple single package repositories

The religious war is all about:

Which is better? Which is faster?

Is there?

MultiRepo

Pros

- Flexibility choosing tools & libs
- Pushing code easier & faster (consumer’s responsibility to fix issues)
- Deployment pipeline per project
- Project level access control

Cons

- Introducing bugs more probable (not immediately evident)
- Fixing bugs harder (need to clone to fix)
- Different toolchains
- Testing entire solution difficult
- Browsing sources harder

MonoRepo

Pros

- Everyone encouraged to make changes to sibling projects
- Short loop
- Browsing all code easier
- Global refactoring easier

Cons

- More friction (dependency graphs)
- Gets large over time (difficult to clone)
- Everything same version
- Coarse-grained access rights harder

Who Uses What?

- MonoRepo
    - Google (1 billion files, 86TB, 25K devs working on the repo)
    - Facebook
    - Babel
    - React
    - Symfony
    - Angular
    - React
    - Ember
    - Vue
- MultiRepo
    - Android (b/c codebase was too large for git) - *REAL PROBLEM (from experience)*
    - Amazon
    - Netflix

The Big Question

Library owner changes API: who fixes affected code?

- **MonoRepo**: library author (can’t check before build is clean)
- **MultiRepo**: library user (their code is now broken)

So - MutiRepo is faster then?

YOU CANNOT COMMIT THAT AND LEAVE!?
SURE I CAN. THEY WILL FIX IT.
cheekycoder.com

In Long Term...

- Authors must support multiple versions (not everyone upgrades right away)
- Consumers must eventually upgrade to new library version

Popularity

Indications more teams lean towards MonoRepos:

- Productivity Increase (up to 5 times some say)
- Better for teams who want to ship code faster
- Better developer testing
- Less code complexity
- Easier code reviews
- Easier refactoring

(However it doesn’t mean MultiRepo is not the right choice sometimes)

So popular that...

GO
MONO
radio 21

npm link

- **npm link (in my-package folder)**
    - Installs package-a globally
- **npm link my-package**
    - Symlinks to package-a
- **npm unlink**

Yarn Workspaces

npm link

- Very simple
- Affects entire system (installs globally)
- Can’t really *npm install* my-package

Yarn Workspaces

https://yarnpkg.com/lang/en/docs/workspaces/

- Yarn feature
- *yarn install* once (anywhere in project)
- That’s it!

Yarn Workspaces

```
{
  "name": "my-mono-repo",
  "private": true,
  "workspaces": [
    "packages/*"   // globs and/or specific folder paths
  ],
  "devDependencies": {
    "chalk": "^2.0.1"
  }
}
```

Yarn Workspaces

Yarn Workspaces - Summary

- Hoists common packages
- Creates symlinks
- Single yarn.lock
- Hoists npm_modules
- Better than yarn link (doesn't affect whole system)

Lerna

- https://lernajs.io/
- Tool for managing multi-package JS projects
- Optimizes workflow with git and npm/yarn
- Can reduce space & time (--hoist flag)
- Can leverage Yarn Workspaces
- Started by the ppl behind Babel

Lerna

```
{ // lerna.json
  ...
  "npmClient": "yarn",
  "useWorkspaces": true // Use yarn workspaces
}
{ // Root package.json
  "private": true,
  "devDependencies": {
    "lerna": "^2.2.0"
  },
  "workspaces": ["packages/*"]
}
```

Lerna

- **npm init**
- **lerna init**
  - lerna.json, add lerna as devDepencency
- Update package.json (npmClient, useWorkspaces)
- **lerna add (--scope=module-name)**
- **lerna publish (-m <message>)**

Lerna

Lerna

- **lerna run** <npm-script-name> [--parallel]

- **lerna diff**

Lerna

- **lerna clean**
- **lerna bootstrap**
    - install & symlink

Lerna

Versioning

Fixed (default)

- Single version line (lerna.json)

Independent (--independent)

- Independent package versions
- Prompts for each one

Lerna

Git Submodules

- Not only frontend
- Git repo in subfolder of another repo
- Keeps code & commits separate
- New .gitmodules file

- Think “repos embedded in main repo”

Git Submodules

Use cases

- Add submodule to multiple repos (reuse)
    - Component sharing
    - Easily update only shared components
    - Finer grained access

- Split code to different repos
    - Big components
    - Different technologies
    - Cleaner git logs (component specific)

Git Submodules

Add submodule

- git submodule add <git-url.git> <directory-name>
- git submodule init // init local git config file
- git submodule update // fetch submodules

Clone project

- git clone <git-url.git> <directory-name>
- (continue as above)

Git Submodules

Git Submodules

Pushing submodule updates

- Submodules are just separate repos
- git add + commit + push on subfolder
- git add + commit + push on root (b/c submodule commit number changed)

Keeping submodules up-to-date

- git submodule update
- Must be done when submodule updated
- Not automatically done by git pull (only retrieves submodule commit numbers, but doesn't update the submodule's code)

Google Bazel

- Originally backend, but lately trending for frontend

- Build & test tool (similar to Make, Maven, and Gradle)
- Works with MonoRepo or MultiRepo
- Proprietary build language
- Supports projects in multiple languages
- Builds for multiple platforms
- Supports large codebases across multiple repositories
- Extensible
- Fast (cache)

Google Bazel

Operation principle:

1. Loads BUILD files relevant to the target
2. Analyzes inputs and their dependencies
    a. Applies the specified build rules
    b. Produces an action graph (*)
3. Executes the build actions on the inputs
    a. until the final build outputs are produced

* Action graph: artifacts, relationships, required build actions

Google Bazel

```
java-tutorial
│
├── BUILD
├── src
│   └── main
│       └── java
│           └── com
│               └── example
│                   ├── cmdline
│                   │   ├── BUILD
│                   │   └── Runner.java
│                   ├── Greeting.java
│                   └── ProjectRunner.java
└── WORKSPACE
```

Google Bazel

Workspace: a directory containing:

- WORKSPACE: file designating a Bazel workspace
- BUILD: Bazel build instructions
- Project's sources
- Build outputs (bazel-bin folder)

A directory containing a BUILD file is called a package

Google Bazel

BUILD file

```
java_binary(
    name = "ProjectRunner",
    srcs = glob(["src/main/java/com/example/*.java"]),
)
```

ProjectRunner target instantiates Bazel's built-in java_binary rule

The rule tells Bazel to build a .jar file

Google Bazel

Building the project:

```
> bazel build //:ProjectRunner
```

the // part is the location of our BUILD file relative to the root of the workspace

Google Bazel

Multiple targets (BUILD file):

```
java_binary(
    name = "ProjectRunner",
    srcs = ["src/main/java/com/example/ProjectRunner.java"],
    main_class = "com.example.ProjectRunner",
    deps = [":greeter"],
)
java_library(
    name = "greeter",
    srcs = ["src/main/java/com/example/Greeting.java"],
)
```

Google Bazel

Working with external dependencies

- Bazel can depend on targets from other (Bazel or non-Bazel) projects
- Called external dependencies
- WORKSPACE file tells Bazel how to get them
- They can contain more BUILD files (with their own targets)
- BUILD files in main project can depend on external targets

```
local_repository(
    name = "coworkers_project",
    path = "/path/to/coworkers-project",
)
```

Google Bazel

Bazel and FrontEnd

- Requires Bazel rules for Frontend development
- Rules are like plugins for Bazel
- Many rule sets are available
- Relevant ones for FE Angular builds for example are:
    - JavaScript Rules
    - TypeScript Rules
    - Angular Rules

Google Bazel

Bazel JavaScript Rules

- Allows us to run JavaScript under Bazel
- Add the NodeJS runtime for executing tools in the Bazel toolchain
- And for building NodeJS applications

Google Bazel

WORKSPACE

```
git_repository(
    name = "build_bazel_rules_nodejs",
    remote = "https://github.com/bazelbuild/rules_nodejs.git",
    tag = "0.8.0", # check for the latest tag when you install
)

load("@build_bazel_rules_nodejs//:defs.bzl", "node_repositories")
```

Google Bazel

BUILD

```
nodejs_binary(
    name = "hello_world",
    ...
)
```

Meta

- https://github.com/mateodelnorte/meta
- “Why choose MultiRepo or MonoRepo when you can have both?”
- Tool for turning many repos into meta repo (MultiRepo → MonoRepo)
- Plugins for: git, npm, yarn
- Create branches on multiple repos
- Push multiple repos at once
- npm / yarn install against all your projects at once

Meta

- mkdir my-meta-repo
- git init
- meta init
- meta project add [folder] [repo url]

Other Tools

Facebook Buck

- https://buckbuild.com/
- Build system developed and used by Facebook
- Encourages creation of small reusable modules
- Supports a variety of languages on many platforms
- Parallels builds and caches unmodified build artifacts
- Prerequisites: JDK, Ant, Py, Git, Watchman

Other Tools

Twitter Pants

- https://www.pantsbuild.org/
- Linux only
- Build system designed for codebases that:
  - Are large and/or growing rapidly
  - Consist of many subprojects that share a significant amount of code
  - Have complex dependencies on third-party libraries
  - Use a variety of languages, code generators and frameworks
- Supports
  - Java, Scala, Python, C/C++, Go, Javascript/Node, Thrift, Protobuf, Android code
  - Adding support for other languages, frameworks and code generators is straightforward