The Monorepo

Storing your source code has never been so much fun

Gareth Rushgrove
- A recent history of code organisation
- The cost of a new repo
- Monorepo examples
- Build tools for monorepos
A recent history

Of code organization
A single team with a monolithic application in a single repository
Multiple teams sharing a monolithic application in a single repository
Multiple teams with separate applications in a single repository
Multiple teams with separate applications in separate repositories
Multiple teams with many separate applications in many separate repositories
Multiple teams with many separate applications microservices in many separate repositories
A single team with many microservice in many repositories
Many teams with many applications in many repositories
Many teams with many applications in one big Monorepo
Enter monorepo

What if we aimed for fewer repositories
Monorepo:
A repository which contains more than one logical project
In some cases the monorepo contains nearly all of the projects in a given organisation.
The point is not that you should have a single project, but that one repository can contain multiple distinct projects.

David R. MacIver
Why
- Simplified organisation
- Simplified dependencies
- Tooling
- Cross-project changes

https://danluu.com/monorepo/
- Single lint, build, test process
- Easy to coordinate changes
- Single place to report issues
- Easier to setup dev environment
- Integration tests

https://github.com/babel/babel/blob/master/doc/design/monorepo.md
You can make atomic refactors across the entire codebase without worrying about versioning issues

Ben Maurer, Facebook
I used to think it was silly but I now think that it is awesome. Wanna rename a db column? Just grep the repo. Etc
All of the above benefits of having *one* repository also extend to having *fewer* repositories

David R. MacIver
The cost of a new repo

Downsides of so many code repositories
It’s very easy to press the button
24 million users on GitHub have 67 million repos

Octoverse 2017
New repos spring into existence every day without you even knowing about it.
Juggling a multimodule project over multiple repos is like trying to teach a newborn baby how to ride a bike

https://github.com/babel/babel/blob/master/doc/design/monorepo.md
If project A depends on library B, you have to encode which revision of library B you use.

Ben Maurer, Facebook
Worse, if Project A also depends on C and D both of which use library B you have to ensure that the version of B that C and D depend on are compatible.

Ben Maurer, Facebook
- End-to-end testing
- Running whole applications locally
- Is that repository used?
- Implicit dependencies in CI config
- High cost of onboarding

Roman Heinrich, Monorepos - Why and How
Examples

Observations from public and private repos
Huge companies often talk about their monorepos
People from huge companies often talk about their monorepos
Why Google Stores Billions of Lines of Code in a Single Repository

By Rachel Potvin, Josh Levenberg
Communications of the ACM, Vol. 59 No. 7, Pages 78-87
10.1145/3354146

Early Google employees decided to work with a shared codebase managed through a centralized source control system. This approach has served Google well for more than 16 years, and today the vast majority of Google's software assets continues to be stored in a single, shared repository. Meanwhile, the number of Google software developers has steadily increased, and the size of the Google codebase has grown exponentially (see Figure 1). As a result, the technology used to host the codebase has also evolved significantly.

Back to Top

Key Insights
Google (business version)
Scaling Mercurial at Facebook

With thousands of commits a week across hundreds of thousands of files, Facebook's main source repository is enormous—many times larger than even the Linux kernel, which checked in at 17 million lines of code and 44,000 files in 2013. Given our size and complexity—and Facebook's practice of shipping code twice a day—improving our source control is one way we help our engineers move fast.

Choosing a source control system

Two years ago, as we saw our repository continue to grow at a staggering rate, we sat down and extrapolated our growth forward a few years. Based on those projections, it appeared likely that our then-current technology, a Subversion server with a Git mirror, would become a productivity bottleneck very soon. We looked at the available options and found none that were both fast and easy to use at scale.

Our code base has grown organically and its internal dependencies are very complex. We could have spent a lot of time making it more modular in a way that would be friendly to a source control tool, but there are a number of benefits to using a single repository. Even at our current scale, we often make large changes throughout our code base, and having a single repository is useful for continuous integration and deployment.
Microsoft claim the largest git repo on the planet

The largest Git repo on the planet
05/24/2017 by Brian Harry MS // 59 Comments

It’s been 3 months since I first wrote about our efforts to scale Git to extremely large projects and teams with an effort we called “Git Virtual File System”. As a reminder, GVFS, together with a set of enhancements to Git, enables Git to scale to VERY large repos by virtualizing both the .git folder and the working directory. Rather than download the entire repo and checkout all the files, it dynamically downloads only the portions you need based on what you use.

A lot has happened and I wanted to give you an update. Three months ago, GVFS was still a dream. I don’t mean it didn’t exist – we had a concrete implementation, but rather, it was unproven. We had validated on some big repos but we hadn’t rolled it out to any meaningful number of engineers so we had only conviction that it was going to work. Now we have proof.

Today, I want to share our results. In addition, we’re announcing the next steps in our GVFS journey for customers, including expanded open sourcing to start taking contributions and improving how it works for us at Microsoft, as well as for partners and customers.

Windows is live on Git

Over the past 3 months, we have largely completed the rollout of Git/GVFS to the Windows team at Microsoft.

As a refresher, the Windows code base is approximately 3.5M files and, when checked in to a Git repo, results in a repo of about 300GB.
There are two sides to Google Infrastructure for Everyone Else
I’m more interested in
Smaller teams and organisations, open source projects and widely available tools
Put another way
Do monorepos scale down as well as scale up?
A monorepo that holds all of Foursquare's open source projects

- pants
- foursquare
- monorepo
- mongodb
- rogue
- scala

- 538 commits
- 1 branch
- 2 releases
- 16 contributors
- Apache-2.0

Commit by mateur: Upgrade Fsq.io Travis config to use mongodb3.0+ (#780)

- 3rdparty: Update the testinfra deployed file (#748)
- build-support: Monolithic Ivy resolve commit (#530)
- scripts/fsqio: Add a check for the current file before deleting (#709)
- src: Add installation instructions to pom
- test: Spindle: Make ThriftParserTest actually depend on its input (#735)
- .dockerignore: Update fsqio/fsqio Dockerfile and add one for fsqio/twofishes
- .gitignore: Upgrade Fsq.io Travis config to use mongodb3.0+ (#780)
- BUILD.opensource: Monolithic Ivy resolve commit (#530)
- BUILD.tools: Drop a BUILD.tools in Fsq.io.
- CLA.md: Move deployed files to consolidated directory.
- CONTRIBUTING.md: Post a CONTRIBUTING.md

Latest commit 4948379 on 1 Aug
foursquare/fsqio
16 contributors, 15 components, 3 languages
At Puppet we ran a small experiment. 1 team. 6 people. 7 components. 4 languages.
Tools for monorepos

Build tools, migration and SaaS
How do monorepos affect tooling?

Zach Holman
@holman

During’s services are all in one monorepo, which has been interesting. Really great for feature work, but many tools aren’t set up for it.

11:46 AM - 6 Sep 2017

3 Likes
Build tools
Most build systems start small and simple, but over time grow into hairy monsters that few dare to touch.

Mokhov, Mitchell, Peyton Jones, Marlow
Non-recursive Make Considered Harmful

Build Systems at Scale

Andrey Mokhov \(^*\)  
Newcastle University, UK  
andrey.mokhov@necl.ac.uk

Neil Mitchell \(^!\)  
Standard Chartered Bank, UK  
ndmitchell@gmail.com

Simon Peyton Jones  
Microsoft Research, UK  
simong@microsoft.com

Simon Marlow  
Facebook, UK  
smarlow@fb.com

Abstract

Most build systems start small and simple, but over time grow into hairy monsters that few dare to touch. As we demonstrate in this paper, there are a few issues that cause build systems major scalability challenges, and many pervasively used build systems (e.g. Make) do not scale well.

This paper presents a solution to the challenges we identify. We use functional programming to design abstractions for build systems, and implement them on top of the Shake library, which allows us to describe build rules and dependencies. To substantiate our claims, we engineer a new build system for the Glasgow Haskell Compiler. The result is more scalable, faster, and spectacularly more maintainable than its Make-based predecessor.

Categories and Subject Descriptors: D.3 [Software]: Programming Languages

Keywords: build system, compilation, Haskell

1. Introduction

In 1998 Peter Miller published his famously influential paper “Recursive Make Considered Harmful” (Miller 1998). He made a compelling case that, when designing the build system for a large project, it is far better to ensure that Make can see the entire dependency graph rather than a series of fragments.

Miller was right about that. But he then went on to say “But, but, but! I hear you cry. ‘A single makefile is too big, it’s unmaintainable, it’s too hard to write... it’s just not practical’ , after which he addresses each concern in turn. Here, however, he is wrong. Using Make for large projects really is unmaintainable, and the rules really are too hard to write.

• Using the Glasgow Haskell Compiler (GHC) as a substantial example, we give concrete evidence of the fundamental lack of scalability of Make and similar build systems (§2 and §4). GHC’s build system is certainly large; it consists of over 10,000 lines of (often incomprehensible) code spread over 200 Makefiles. Motivated by its shortcomings, GHC developers have implemented no fewer than four major versions of the build system over the last 25 years; it improved each time, but the result is still manifestly inadequate.

• We describe Shake, an embedded domain specific language (or library) in Haskell that directly addresses these challenges in §3. Although Shake has been introduced before (Mitchell 2012), here we describe several key features that were mentioned only in passing if at all, notably: post-use and order-only dependencies; how to use polymorphic dependencies; resources; and content hashes.

• We show in some detail how Shake’s built-in abstractions address many of the scalability challenges that have caused the GHC developers such pain over two decades (§4).

• A huge benefit of using an embedded DSL as a build system is that we can use the facilities of the host language (Haskell) to build abstractions on top of Shake, to fit our particular use case. This sort of claim is easier to make than to substantiate; so in §5 we present an overview of the new build system we have developed for GHC, and the new abstractions (not part of Shake) that we built to support it.

• To validate our claims, we have completely re-implemented GHC’s build system, for the fifth and final time. The new version is only a little shorter than the old — Make is already very short — but it is more maintainable, readable, and scalable.
Buck is a build system developed and used by Facebook. It encourages the creation of small, reusable modules consisting of code and resources, and supports a variety of languages on many platforms.

Why Buck?
Buck can help you and your team in many ways:

- **Speed up your builds.** Buck builds independent artifacts in parallel to take advantage of multiple cores on your machine. Further, it reduces incremental build times by keeping track of unchanged modules so that the minimal set of modules is rebuilt.

- **Add reproducibility to your builds.** Buck only uses the declared inputs, which means everybody gets the same results.

- **Get correct incremental builds.** Buck looks at the contents of your inputs, not their timestamps, to figure out what needs to be built. As a result, incremental builds should be more accurate.
java_binary(
    name = "hello-buck-java",
    main_class = "com.facebook.buck.demo.HelloBuckJava",
    deps = [":main"],
)

java_library(
    name = "main",
    srcs = glob(["*.java"]),
)
Pants: A fast, scalable build system

Pants is a 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.

Pants supports Java, Scala, Python, C/C++, Go, Javascript/Node, Thrift, Protobuf and Android code. Adding support for other languages, frameworks and code generators is straightforward.

Pants is a collaborative open-source project, built and used by Twitter, Foursquare, Square, Medium and other companies.

Getting Started

- Installing Pants
- Setting Up Pants
- Tutorial

Cookbook

The Common Tasks documentation is a practical, solutions-oriented guide to some of the Pants tasks that you’re most likely to carry out on a daily basis.
python_library(
    name = 'pants',
    sources = globs('*.py'),
    dependencies = [
        '3rdparty/python:pantsbuild.pants',
    ],
)
Bazel from Google

{Fast, Correct} - Choose two

Build and test software of any size, quickly and reliably

GET BAZEL  GET STARTED

Why Bazel?

Speed up your builds and tests
Bazel only rebuilds what is necessary. With advanced local and distributed caching, optimized dependency analysis and parallel execution, you get fast and incremental builds.

One tool, multiple languages
Build and test Java, C++, Android, iOS, Go and a wide variety of other language platforms. Bazel runs on Windows, macOS, and Linux.

Scalable
Bazel helps you scale your organization, codebase and Continuous Integration system. It handles codebases of any size, in multiple repositories or a huge monorepo.

Extensible to your needs
Easily add support for new languages and platforms with Bazel's familiar extension language. Share and re-use language rules written by the growing Bazel community.
Bazel BUILD file

cc_library(
    name = "hello-greet",
    srcs = ["hello-greet.cc"],
    hdrs = ["hello-greet.h"],
)

cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
    deps = [
        ":hello-greet",
    ],
)
We used Make
At smaller scale picking one build tool is probably more important than which one
Migration tools
You can move from multi-repos to a monorepo, without losing history or work-in-progress.
tomono - create a monorepo from many repos

Multi- To Mono-repository

Merge multiple repositories into one big monorepository. Migrates every branch in every subrepo to the eponymous branch in the monorepo, with all files (including in the history) rewritten to live under a subdirectory.

Usage
Maintain history from separate repos with tomono

$ cat repos.txt

```bash
git@github.com:mycompany/service-one.git one
git@github.com:mycompany/service-two.git two
git@github.com:mycompany/service-three.git three
```

$ cat repos.txt | tomono.sh
You can keep your monorepo private and still engage with open source
split.sh - split a repository into read-only repos
fbshipit - copy commits between repositories

Copy commits between repositories - git => git, git => hg, hg => hg, or hg => git

- hacklang
- mercurial
- git
- sync
- commits
- facebook
- hack

517 commits   4 branches   1 release   78 contributors

Branch: master   New pull request

- yns88 Merge pull request #40 from yns88/master
- fb-examples resync due to change in shipit, fbshipit path mappings
- src
- tests
- .gitignore
- .hhconfig
- .travis.sh
- .travis.yml
- CONTRIBUTING.md
- DEBUGGING.md
- LICENSE

Latest commit #483722 3 days ago

- 3 days ago
- 3 days ago
- 2 months ago
- a year ago
- 2 years ago
- 6 days ago
- 6 days ago
- 2 years ago
- 9 months ago
- 2 years ago
JavaScript  🥀  Monorepos
Oao - Javascript monorepo management tool
Projects tend to grow over time, and, occasionally, some pieces of a project can be useful elsewhere in other projects. For example, Jest, being a generic testing tool, gave birth to many packages, one of them is jest-snapshot that is now used in other projects like snapguidist and chai-jest-snapshot.

**Monorepos**

Those who have tried splitting a project into multiple packages know how hard it is to make changes across multiple packages at one time. To make the process easier, some big projects adopted a monorepo approach, or multi-package repositories, which reduces the burden of writing code across packages.
Rush from Microsoft
Lerna - a tool for managing Javascript monorepos

A tool for managing JavaScript projects with multiple packages.

Follow Lerna on Twitter
SaaS products
Introducing code owners

Published: July 6, 2017  Author: jdpaces  Category: New Features

While effective code review is essential to every successful project, it’s not always clear who should review files—even with GitHub’s reviewer suggestions. Now repository maintainers can define exactly which people and teams need to review projects using code owners. This new feature automatically requests reviews from the code owners when a pull request changes any owned files.

How code owners work

To specify code owners, create a file named CODEOWNERS in the repository’s root directory (or in .github/ if you prefer) with the following format:

```
# Lines starting with '#' are comments.
# Each line is a file pattern followed by one or more owners.
# These owners will be the default owners for everything in the repo.
* @defunkt

# Order is important. The last matching pattern has the most precedence.
# So if a pull request only touches javascript files, only these owners
# will be requested to review.
*.js @octocat @github/js

# You can also use email addresses if you prefer.
docs/* docs@example.com
```

GitHub code owners
# Lines starting with '#' are comments.
# Each line is a file pattern followed by one or more owners.

# These owners will be the default owners for everything in the repo.
*   @garethr

# Order is important. The last matching pattern has the most precedence.
# So if a pull request only touches javascript files, only these owners
# will be requested to review.
*.js   @octocat @github/js

# You can also use email addresses if you prefer.
docs/*  docs@example.com
Gain more control over your delivery process with Build Stages

Sven Fuchs, 11 May 2017

We are very excited to introduce the availability of Build Stages as a beta feature today.

Build stages are our interpretation of a flexible and practical continuous delivery pipeline. They give you more control over your build process, and over how your code flows from check-in to production. This allows Travis CI to provide you and your team a better continuous delivery solution.

What are Build Stages?

With build stages, we’re now implementing one of the longest standing feature requests from Travis CI users: in the simplest and most common use case, you can now make one job run only if several other, parallel jobs have completed successfully.

What could this look like in practice? Let’s say you want to test a library like a Ruby gem or an npm package against various runtime (Ruby or Node.js) versions in parallel (we have supported this parallelization with our build matrix feature from the beginning). And you want to release your gem or package only if all tests have passed and completed successfully. Build stages already make this very basic and common workflow possible.

Of course, there are a lot more, and a lot more complicated use cases than this one. You can find many more examples and tutorials in our documentation.
Explicit build matrix in Travis

matrix:

include:
- go: 1.8
  language: go
  install: cd consumer
  script: 
    - make dependencies test
- jdk: oraclejdk8
  language: clojure
  services: docker
  install: cd reporter
  script: 
    - make ensure-table test
- python: 3.6
  language: python
  install: cd data
  script: 
    - make test
- services: docker
Conclusions

If all you remember is...
Monorepos are not just for Google, Facebook and Microsoft and Twitter
Learn monorepo and multi-repo approaches to find out what works **best for you**
No silver bullet

Fred Brooks
I’d argue that we don’t really yet have a good handle on how to scale that area that exists at the intersection of engineering and human organization

Peter Seibel, Twitter
Code organisation will only get more important as more people write ever more code.
Any questions?

And thanks for listening