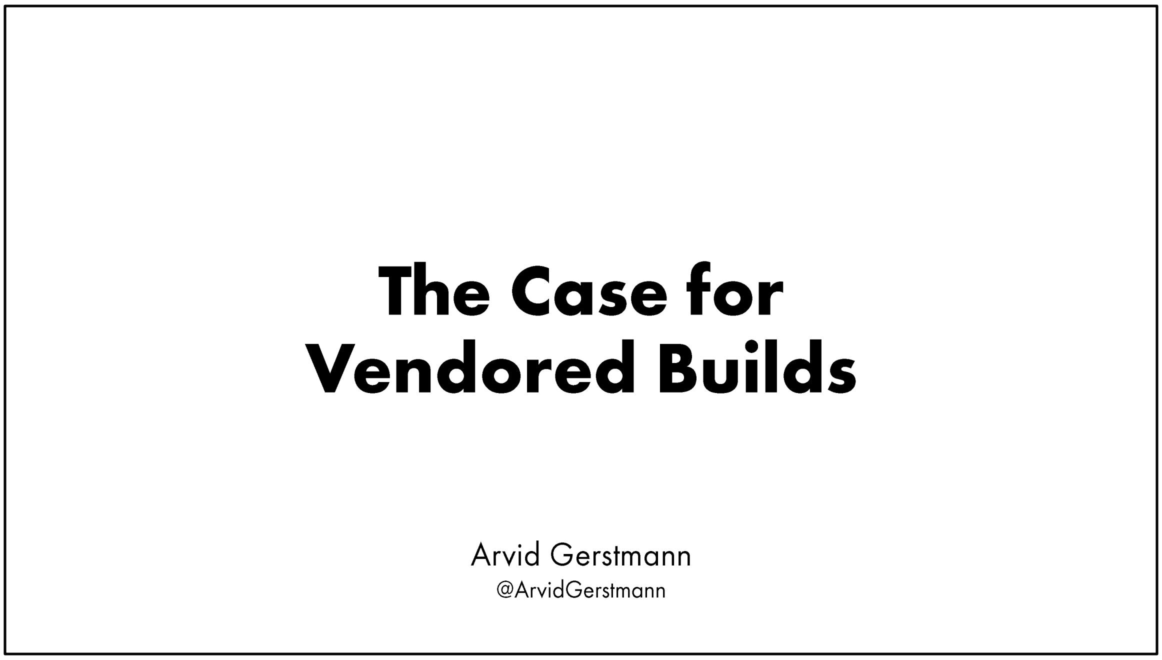


The Case for Vendored Builds

Arvid Gerstmann
@ArvidGerstmann

Hi, my name is Arvid Gerstmann. And this is „The Case for Vendored Builds“.
Let‘s get started!

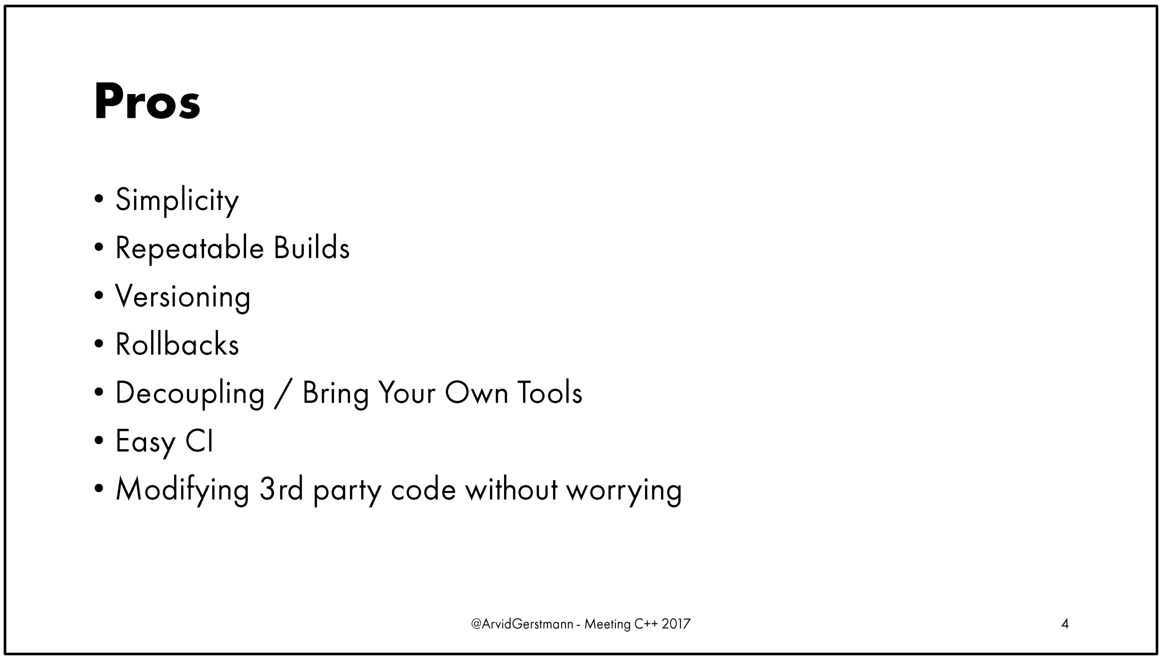
What are Vendored Builds?

What are vendored builds? Vendored builds, also known as „vendorized toolchain“, or simply „vendorization“, is an often employed technique to remove dependency on the local development environment, by bundling absolutely all dependencies inside your repository. The only exception being optional development tools and the source control client itself.

Pros

@ArvidGerstmann - Meeting C++ 2017 3

But why, you might ask? By including all 3rd party code, the build system, compiler toolchains for each supported operating system, platform SDKs you gain a bunch of benefits:



Pros

- Simplicity
- Repeatable Builds
- Versioning
- Rollbacks
- Decoupling / Bring Your Own Tools
- Easy CI
- Modifying 3rd party code without worrying

@ArvidGerstmann - Meeting C++ 2017 4

Simplicity. Adding a new developer to your team is not a tedious and long task anymore. Setting up a new development environment simply becomes installing your version control systems client and cloning your repository. No other setup required.

Repeatable Builds. Since everything is inside source control, rebuilding a year old build, after you‘ve upgraded Visual Studio twice, becomes as easy as checking it out and pressing build. No more trying to find out what Windows SDK version you were using, or installing every old Visual Studio version. Even 10 years later, you should be able to build your project without any hassle.

Versioning. Upgrading a library or your compiler does not require each engineer to change their local environment anymore, no more fiddling with „apt-get“ to pin specific versions of OpenSSL or Qt5. One engineer can upgrade the compiler or library for everyone in the team.

Rollbacks. Found a bug in an SDK? Don‘t worry. Simply revert the commit/changelist introducing the upgrade. No need to manually re-install an old version.

Decoupling. By decoupling your build-environment from your locally installed IDE or compiler, engineers can use the latest and greatest IDEs, or even don‘t use an IDE at all and keep using Vim.

Easy CI. Ever tried to maintain a continuuos integration environment accross multiple build servers for a large project? Ever fiddled with a ton of .bat or .sh scripts to install the necessary dependencies for each machine? Spinning up new servers becomes easy, when the only dependency is the source control itself. You can install a near stock Linux or Windows image and start building your project.

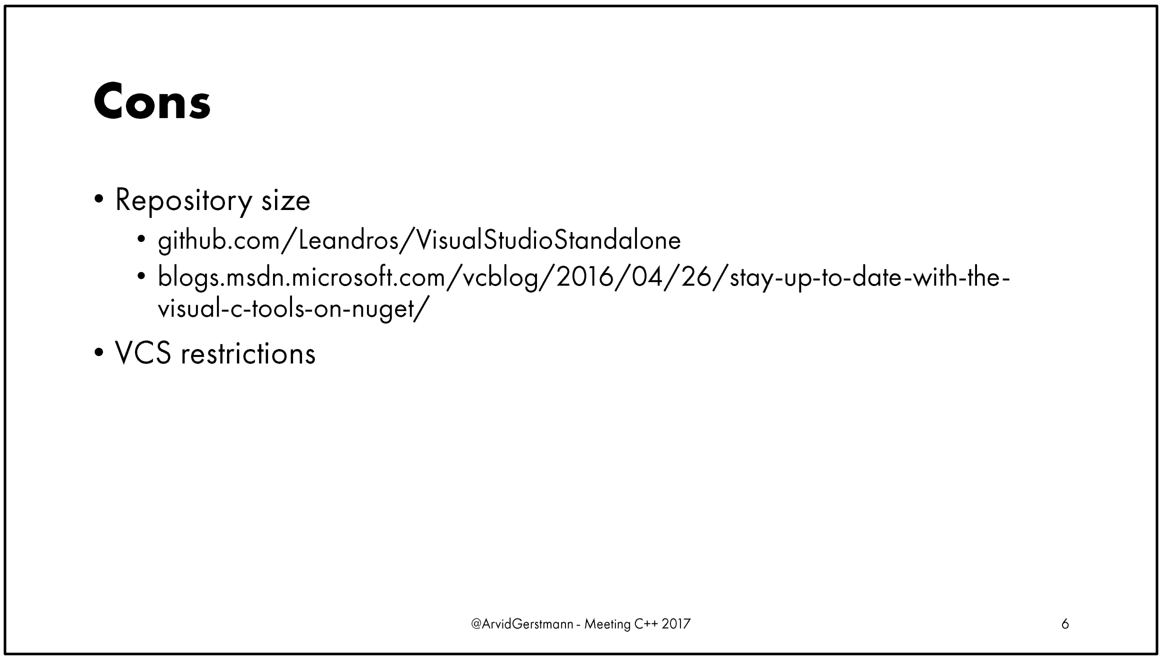
You can modify 3rd party code, without worrying. It‘s often required to fix a bug, or change a small feature in a 3rd party SDK. By having the SDK inside your source tree, you don‘t need to worry about maintaining an external fork somewhere on github, and only pulling code from there. It‘s all in your repository.

Cons

@ArvidGerstmann - Meeting C++ 2017

5

Everything has a downside. And so do vendored toolchains.



Cons

- Repository size
  - github.com/Leandros/VisualStudioStandalone
  - blogs.msdn.microsoft.com/vcblog/2016/04/26/stay-up-to-date-with-the-visual-c-tools-on-nuget/
- VCS restrictions

@ArvidGerstmann - Meeting C++ 2017 6

Repository size. Due to checking in absolutely all dependencies, your build tree might become quite large. Especially if you dump the full dependencies naively. With a little bit of effort, you can trim down compiler and SDK trees by removing all unused executables, libraries and public documentation.

One such way is a project of mine, called „VisualStudioStandalone“, which is extracting just the required files for using CL.EXE.

Although, the officially supported way, as outlined by Andrew Pardoes, is to use the NuGET packages, which package the full compiler toolchains for easy consumption.

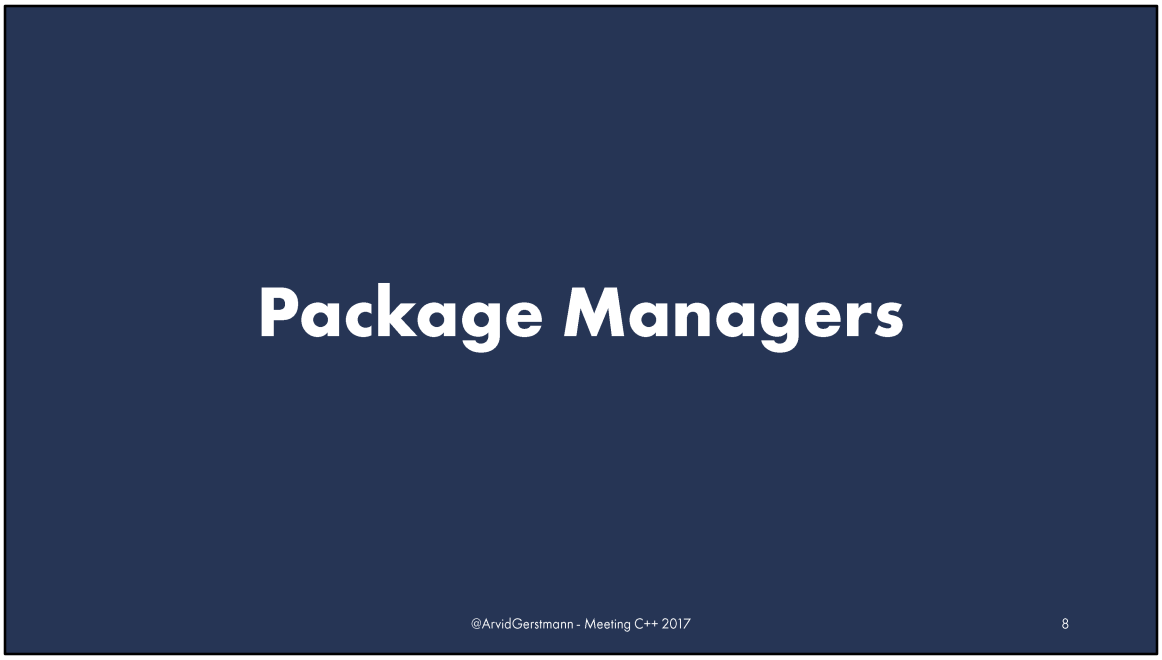
Repository size is only a problem, if your repositories currently only consists of source files, since, for example, in the game development world, repositories contain art assets, which are multiple times the size of all vendored dependencies, making the size not an actual problem.

VCS restrictions. With a large repository size, especially in binaries or assets, comes the restriction of what VCS you are able to use. Everybody probably knows git, everybody probably used git at least once, too, since it‘s likely the currently most used VCS. But a terrible choice to use for a repository which largely consists of binaries, since git, due to it‘s distributed nature, keeps the whole history locally, which will inevitebly become quite large. Which is the reason why Perforce largely dominates the VCS market for game developers, with PlasticSCM coming in second.

Compared to …

@ArvidGerstmann - Meeting C++ 2017 7

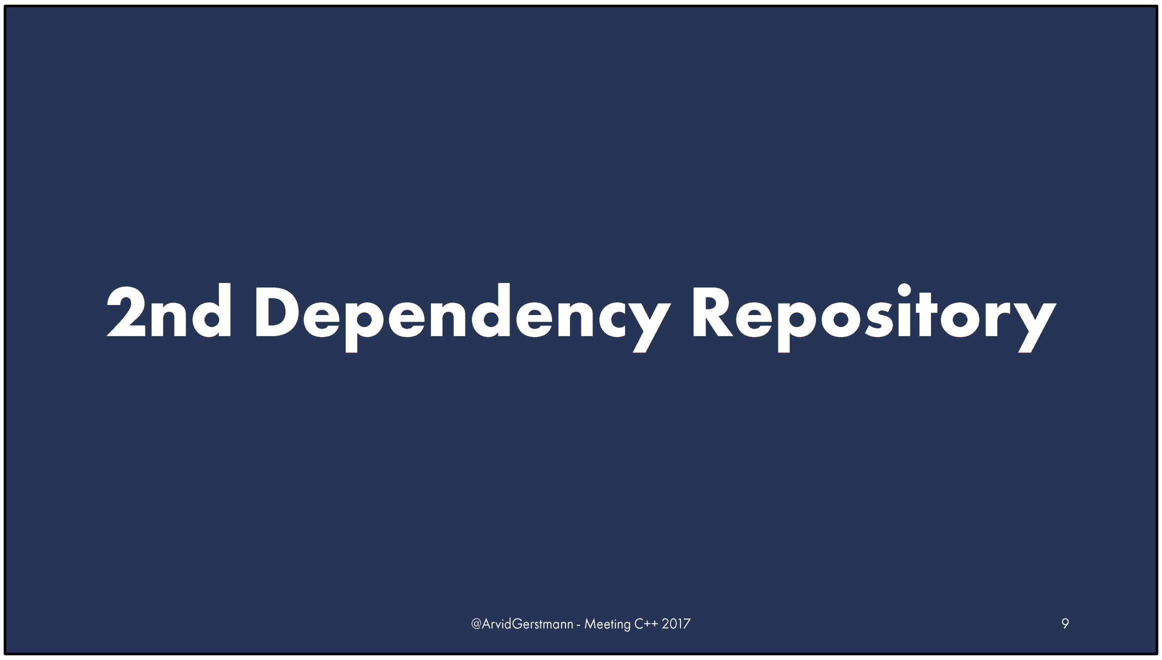
There are many alternatives. Vendorization is not the only solution, for achieving most of the goals I described. It‘s one out of many, but it‘s the one, which is by far the most powerful and most convenient one.



Package Managers

@ArvidGerstmann - Meeting C++ 2017 8

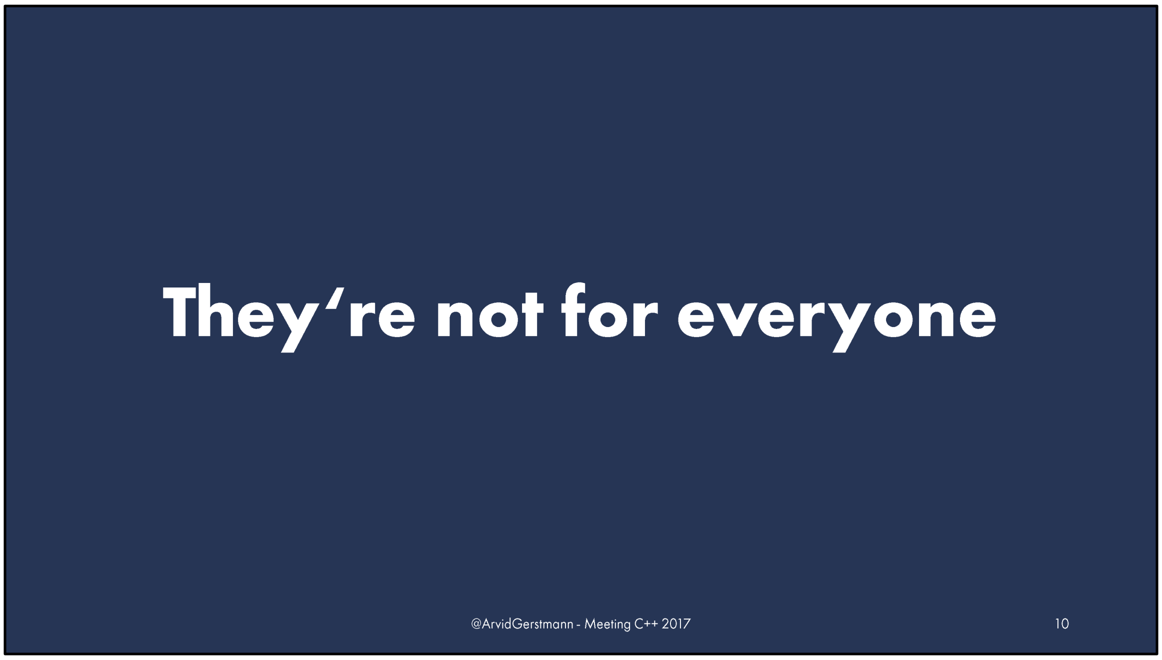
Package Managers. NPM, apt-get, Conan, Pacman, et cetera are all great package managers, without a doubt, but they all have one major downside: they‘re an external dependency. Relying on an external dependency can be dangerous, anyone remembers left-pad? Yes, you can maintain a snapshot of packages locally, or internally for your company, but doing so, just adds another layer of complexity which needs to be maintained by someone.



2nd Dependency Repository

@ArvidGerstmann - Meeting C++ 2017 9

Second dependency repository. Another common way of maintaining a bunch of dependencies, is to maintain them in a second „blob“ repository, which either contains pre-compiled binary blobs, or the latest source snapshot currently in use. Those are than pulled by your build system at configuration time, and used. This keeps the source tree clean of dependencies, and allows for the source to be kept in git.
Although, while it might sound convenient, you, again, have a second repository you need to maintain and properly version, otherwise you‘d lose the ability to easily revert back to old revisions.

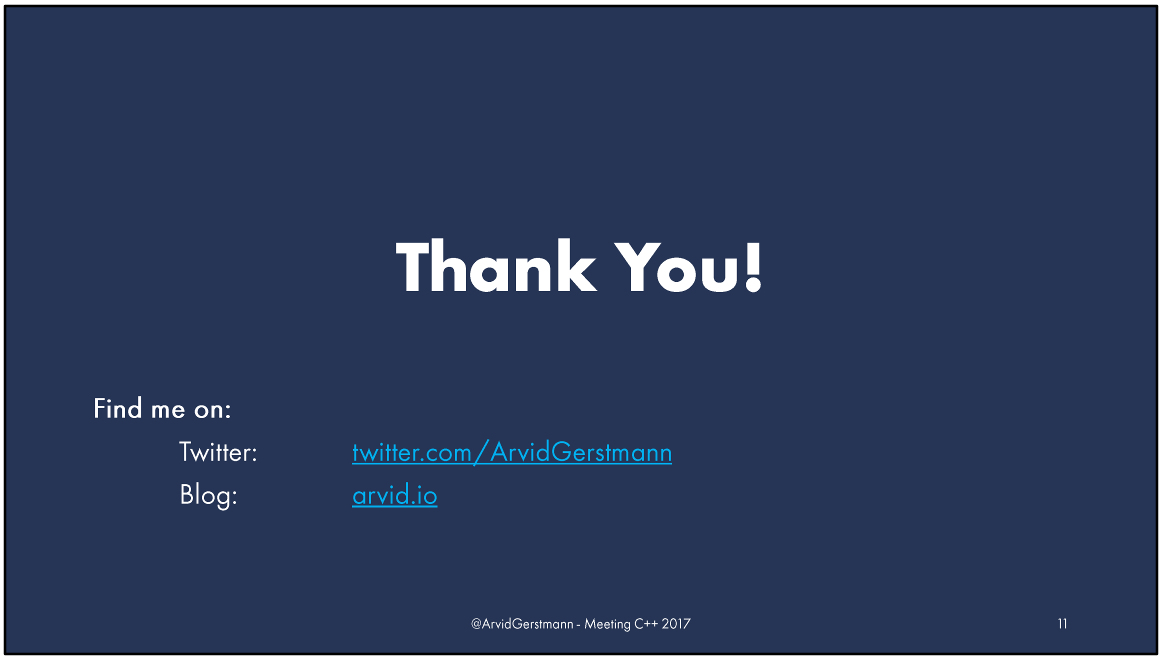


They‘re not for everyone

@ArvidGerstmann - Meeting C++ 2017

10

Vendored builds are not for everyone. But if you work with a large codebase, with many dependencies and you see yourself often figthing with them, you might wanna give them a try.



Thank You!

Find me on:

Twitter:	twitter.com/ArvidGerstmann
Blog:	arvid.io

@ArvidGerstmann - Meeting C++ 2017 11

Thank you!
Slides will be available to download from arvid.io in a couple of minutes.