Should I really be using make?

Asked 11 years, 11 months ago Modified 11 years, 11 months ago

Viewed 958 times

7

At some point, a colleague and I grabbed an "example" makefile from the net for building our embedded code for an xmega chip. We've found the experience very frustrating. Neither of us are make experts, novice at best. We get by making the occasional tweak/adjustment. We spend hours reading make manuals and throwing darts to try and do more serious changes.

Usually when we build though, we always start with a clean all. Because the auto depend generation doesn't seem to work reliably and we've just learned that it's safer this way. Since it's an embedded project, on a small processor, the compile actually flies. It's done in an eye blink, which leads me to my real question here:

If I'm not using make to do any sort of dependency management and leverage incremental building, is there any real sense in using it instead of a simple shell script?

I'm much more confident writing C code, python, and good ol' bash scripts. Today's latest frustration, was trying to move a handful of our source files related to FreeRTOS into a subdirectories. The only real advantage of make for us, is that it is stock installed on OSX, and that vi and QtCreator and XCode have some ability to

integrate with our makefile (but I could make a very minimal makefile that bridged here).

c embedded makefile

Share

Improve this question

Follow

asked Jan 3, 2013 at 0:35

Travis Griggs

22.2k • 22 • 102 • 174

6 Just a guess, but the down vote was probably because this is an open ended, and highly subjective question. I wouldn't be surprised if this one is closed. Read up on: What kind of questions should I not ask here? – Nocturno Jan 3, 2013 at 1:13

There's lots of good answers here. None is exactly what I ended up doing, but all of them influenced. What is the best way to grant an answer in this case? – Travis Griggs Jan 7, 2013 at 16:30

6 Answers

Sorted by: Highest score (default)

17

Make is a venerable expert system for building software. It is still used heavily and extensively, so yes it is worth learning. You may be lucky enough to have a project at the moment that builds almost instantaneously, however, in my experience this is the **exception** not the rule.

Given that dependency tracking and action inference is the purpose of Make, using it as just a batch system is a

zero gain, and also it interferes with you learning how to use the tool (Make).

From your post it sounds like you have run afoul of using Make with subdirectories. While it probably won't solve your immediate problem, this paper: Recursive Make Considered Harmful, may help explain the situation, and give you insight into operation of Make itself.

Make does not automatically generate dependencies. You will find numerous resources on the net for how to automatically generate dependencies for GNU Make. I'm not sure if OSX uses GNU Make or BSD Make (or even if the BSD variant still exists). If it is the latter (BSD) I'm sure you can find recipes for how to automatically generate dependencies for BSD variant.

The more existential question I infer in this post is: is there value in learning this old tool when it does not provide immediate benefit to my project?

In answer I would say: your small project is the perfect opportunity to learn this tool. The pain threshold is bound to be far higher if you are faced with a project where incremental compilation really saves time.

There are many other build systems that either replace Make such as: Jam, SCons, Rake, or act as a meta-makefile generator: CMake, some IDEs. The sheer number of replacements is an indication that many find Make inadequate (or perhaps in some cases maybe just disagreeable).

However, a little knowledge of Make is likely to hold you in good stead for the foreseeable future. It is also possibly the first (or most obvious) step in understanding how build systems work, and the problems associated with rule production, and dependency tracking.

Best of luck.

Share Improve this answer

Follow

answered Jan 3, 2013 at 1:26

Marc Butler

1,376 ● 8 ● 13

It seems to me, that the experience here would be contextual. For deeply embedded projects, your cpu, memory, and flash limitations are often so tight, that you'd be really hard pressed to code something that did take appreciable time to full compile and at the same time fit within the constraints of the target system. – Travis Griggs Jan 3, 2013 at 17:55

I'm not sure what you mean by *deeply embedded*. Speaking only from my own experience worked with a chip much like a PIC, but it was small part of a larger project. The whole project was wrapped up with Make so the release team could just start a top-level make and build everything. Also build system (even Make) also allows you generate source code for compilation automatic. I've seen this used for custom RPC bindings for a M68K board. Obviously, only you can evaluate the utility of learning Make. I just offer that is in my experience a worthwhile investment, and small project is an ideal lab. – Marc Butler Jan 3, 2013 at 22:21

This is a question I set to myself a while ago. Personally I use python script to compile my code for AVR (link). I am

2

not a make expert but I do have some experience with it. Still I find it frustrating and for uC needs worthless, unless you are anyway already an expert.

Not that I would encourage anyone to do so, but nothing can stop you from implementing the same dependency checking in Python (Perl, Ruby, Bash...), possibly even much more powerful. I believe it is just a matter of language choice or what you feel comfortable with. make is just a kind of programing language and it might seem quite obscure for beginner.

Share Improve this answer

Follow

answered Jan 3, 2013 at 11:35

Stefan

336 ● 2 ● 4

This advice actually was actually what I was looking for I guess. I wrote my own python scripts. Loved doing it. Have been able to tune it exactly to my needs. And it works great. – Travis Griggs Feb 11, 2013 at 20:55

2

I would personally not have used make directly, almost regardless of the size of your system (meaning I could, in rare cases, use it on extremely small systems). I would have used a tool that generates makefiles for you. Of course, you would loose the installed on a stock OS part, but will get rid of a lot of headache.

Personally I would recommend you to have a look at cmake and autotools. Personally I prefer cmake, and find it to be brilliant, while autotools remove some of the

headache, but adds some new ones. Either one of them I find better than makefiles though.

Cmake also has the advantage that it is more cross platform, and can create makefiles, visual studio files, code::block files and many more. It also supports complete out-of-source builds, meaning you can make a _build folder, and build from there. That will then put all generated files into the _build folder instead of polluting your source tree.

EDIT: To answer your question: I would learn basic make files, to understand what it does. But then I would have switched over. So I would learn make, but not use it.

Share Improve this answer

Follow

answered Jan 3, 2013 at 11:48

martiert
741 ● 4 ● 13

2

Just about any non-trivial C program that is not a single compilation unit will have dependencies within itself, and almost all have dependencies on system libraries.

Unless you intend to write a script that performs a clean build each time, some kind of dependency management tool is required to keep your work-flow sane.

`make` is the canonical choice and is well suited to be job provided you find a way of automatically generating the dependency rules in the first place.

The QT build system doesn't as such integrate with Makefiles, it creates them from an even higher level description of your project. `QTCreator` is one of many tools providing a GUI around this process. When you compile from within `QTCreator` it is actually the resultant `Makefile` that is invoked. You can also use `qmake` to generate XCode and Visual Studio workspace files.

The QT build system is actually a fairly good choice for embedded projects, even when not using QT as it is relatively easy to set it up to handle cross-compilation.

Share Improve this answer

Follow

edited Jan 3, 2013 at 11:59

Jens

72.4k ● 16 ● 130 ● 183

answered Jan 3, 2013 at 1:19

marko

9,149 ● 4 ● 33 ● 47

1

Unless you're using an IDE, make is really the de-facto tool for this kind of thing, unless you want to go further down that road and use automake or similar. I think that's overkill for your case.

The auto-dependency generation is somewhat tricky, but you can generally find a recipe on the web that works. I'm not sure what specific problem you had, but perhaps that's worth a separate question.

It could be that a bash script is enough for your needs; if that's true, go for it. Given that you're more familiar with

Python, you might also want to look at something like SCons.

I've done a fair bit of Makefile work, but I'm in the process of moving to Rake. I find the power of Ruby under the hood to suit my needs better than make.

Share Improve this answer

Follow

answered Jan 3, 2013 at 1:07

Randy Coulman

546 ● 1 ● 3 ● 6

0

If you're doing a full build every time, using make to run exactly the same sequence of commands every time gives you no real advantage. However, make is an extremely powerful system, and you're making a mistake to shun it. Make adds up to a whole lot more than you can hope to cobble together with custom "python build scripts".

1. Even if *this* project is small enough to build quickly from scratch, you will soon tackle something bigger. Make is easier to learn with a simple use case than with a complex one. You say that "auto depend generation doesn't seem to work reliably": if you define the dependencies correctly, make is *extremely* reliable. So there must be a mistake in your set-up (more on this below).
2. Make is useful even if you always build from scratch: Instead of hand-crafting a bunch of little build scripts, you can use the Makefile as a container for all your

build-related commands, all controlled by the same symbolic variables: E.g., if you have a variable like `ALLSOURCE`, you can use it with pseudo-targets that make a zipfile, a ctags index, etc. (`make clean` works this way too). If you write separate batch scripts for everything, you'll be scrambling to update them as your project evolves-- and they'll be cluttering your directory for no reason.

But you can't get the dependencies to work reliably, you say. I suspect that starting with someone else's example script was a mistake: It probably did too much you don't need, and don't entirely understand. Start with a clean Makefile, add targets for the binaries you generate, and write your own dependencies. Keep it simple; if your project is so easy to build that you're even thinking of doing your own script, it should be easy to write a makefile for it. If you get stuck, ask specific questions here.

Share Improve this answer

Follow

answered Jan 26, 2013 at 20:45

alexis

50.1k ● 17 ● 105 ● 168