The vgo proposal is accepted. Now what?

Go & Versioning, Part 8

Russ Cox

May 29, 2018

research.swtch.com/vgo-accepted

Last week, the proposal review committee accepted the "vgo approach" elaborated on this blog in February and then summarized as proposal #24301. There has been some confusion about exactly what that means and what happens next.

In general, a Go proposal is a discussion about whether to adopt a particular approach and move on to writing, reviewing, and releasing a production implementation. Accepting a proposal does not mean the implementation is complete. (In some cases there is no implementation yet at all!) Accepting a proposal only means that we believe the design is appropriate and that the production implementation can proceed and be committed and released. Inevitably we find details that need adjustment during that process.

Vgo as it exists today is not the final implementation. It is a prototype to make the ideas concrete and to make it possible to experiment with the approach. Bugs and design flaws will necessarily be found and fixed as we move toward making it the official approach in the go command. For example, the original vgo prototype downloaded code from sites like GitHub using their APIs, for better efficiency and to avoid requiring users to have every possible version control system installed. Unfortunately, the GitHub API is far more restrictively rate-limited than plain `git` access, so the current vgo implementation has gone back to invoking `git`. Although we'd still like to move away from version control as the default mechanism for obtaining open source code, we won't do that until we have a viable replacement ready, to make any transition as smooth as possible.

More generally, the key reason for the vgo proposal is to add a common vocabulary and semantics around versions of Go code, so that developers and all kinds of tools can be precise when talking to each other about exactly which program should be built, run, or analyzed. Accepting the proposal is the beginning, not the end.

One thing I've heard from many people is that they want to start using vgo in their company or project but are held back by not having support for it in the toolchains their developers are using. The fact that vgo is integrated deeply into the go command, instead of being a separate vendor directory-writer, introduces a chicken-and-egg problem. To address that problem and make it as easy as possible for developers to try the vgo approach, we plan to include vgo functionality as an experimental opt-in feature in Go 1.11, with the hope of incorporating feedback and finalizing the feature for Go 1.12. (This rollout is analogous to how we included vendor directory functionality as an experimental opt-in feature in Go 1.5 and turned it on by default in Go 1.6.) We also plan to make minimal changes to legacy `go get` so that it can obtain and understand code written using vgo conventions. Those changes will be included in the next point release for Go 1.9 and Go 1.10.

One thing I've heard from zero people is that they wish my blog posts were longer. The original posts are quite dense and a number of important points are more buried than they should be. This post is the first of a series of much shorter posts to try to make focused points about specific details of the vgo design, approach, and process.