A Guide to Risky Changes

Please take special care when working on **risky changes**. As a reminder:

A change is risky if it can cause **failures that are hard to diagnose** (for example, changes to the runtime, GC, compiler, linker, TLS, other low-level component, or **complex changes that need soak time under production workloads**), or if it **requires many CLs that are hard to revert** (for example, large CLs or stacks of CLs).

If you plan on working on a change that may be risky, please do the following:

1. Unless the entire change is absolutely trivial to revert, protect the new code paths with a boolean flag, prefixed with "go121", that can be used to quickly toggle back to the old implementation. It can be a simple bool constant, for example, const go121UseEvenBetterLinker = true. Such flags **must be findable** by a simple grep for the string "go121". That way we can find them without missing any, and they can be cleaned up when we get to the Go 1.22 cycle.

2. Consider how you would answer the following questions for your change:
   ➢ How risky is the change you're planning to make?
   ➢ How will you know if it is working as intended?
   ➢ How much production testing does it need for you to be confident it is working as intended?
   ➢ When should the keep/revert decision be made?

3. Create a tracking issue in the Go 1.21 milestone with a release-blocker label. This issue will be used to track progress on the feature and make the final decision for Go 1.21.

Best,
The Go team