# Gardening

## What is gardening?

"Gardening" in open source projects refers to the background maintenance tasks done to keep the project healthy & growing & nice looking.

This page lists common Go gardening tasks.

## Access

If you've been regularly active in the Go community for some time, feel free to ask for Gerrit and/or Github access to modify things.

See https://go.dev/wiki/GerritAccess and https://go.dev/wiki/GithubAccess

## Gardening Tasks

Before doing any gardening work, especially on the issue tracker, remember to familiarize yourself with the issues life-cycle, described here: Handling Issues - Issue States.

### Fix red

Look at https://build.golang.org/ — is anything red? Fix or file bugs or nag people. The build dashboard should never be red, even occasionally. If the tree is red, people can't work effectively because TryBots and such will just report failures, masking other problems.

### Triage new bugs

Look at the untriaged issues. For Go, we use the presence of certain labels (`Needs*` or `WaitingForInfo`) to indicate that an issue has been triaged. Issues labeled `Go2`, `Proposal`, `CherryPickCandidate`, `gopls`, `pkgsite` have their own, separate triage queues and can be skipped.

While triaging the bug:

- is it a duplicate? Close it, referencing the dup.
- is it a Question rather than a bug? Reply with something like "For questions about Go, see https://go.dev/wiki/Questions".
- is the subject the correct format? It should start with the package path and a colon: "net/http: fix crash in Server during foo operation".
- is it in a subrepo? Leave the milestone as `Unreleased` unless it's a subrepo that goes into a release, like `http2`.
- if it is a regression and you can reproduce it, use `git bisect` to find the bad commit (optional but very helpful).

When the issue has been triaged, add an appropriate label (per https://go.dev/wiki/HandlingIssues) to mark it as such.

### WaitingForInfo

Find bugs that are in state WaitingForInfo (https://github.com/golang/go/labels/WaitingForInfo) and ping them, remove the label when replies arrive, or close the bugs if a reply never arrived.

### "Unplanned" bugs

"Unplanned" issues have a habit of being neglected. Check out old ones and see if they're easily fixable (and can be moved to a Go1.n or Go 1.nMaybe milestone), or should be closed.

### Pending CLs

Review the format of commit messages and presence of tests and formatting of code and typos/grammar in incoming pending CLs. All of that can be done without determining the correctness of the change itself. See https://dev.golang.org/release for the list of pending CLs.

Once it has a +1, the owner of that area can give it a +2.

Read a +1 as meaning "triaged", or "not obviously wrong". If it has tests, is formatted properly (references a bug number, probably), and is ready for more review, give it a +1.

### Pending CLs: ask about tests

If a new CL arrives without a test, but could/should have a test, ask if they could add a test. Or suggest how.

### Pending CLs: run TryBots

If you have access (see https://go.dev/wiki/GerritAccess) to run the TryBots and you see a CL with plausible (and non-malicious) code, kick off the TryBots. (We've never seen malicious code trying to escape our TryBot sandboxes, but that's why it's not automatic yet. Please alert us if you see something.)