Be sure to familiarize yourself with the code review process from the official Contribution Guide first.

Reviewer Parlance

There are several terms code reviews may use that you should become familiar with.

- `LGTM` — looks good to me
- `SGTM` — sounds good to me
- `PTAL` — please take a look
- `s/foo/bar/` — please replace `foo` with `bar`; this is sed syntax
- `s/foo/bar/g` — please replace `foo` with `bar` throughout your entire change

CL Directives

- `R=foo` — assign a reviewer within the Go CL dashboard
- `RELNOTE=yes` or `RELNOTE=<subject>` — tag for release notes; scraped by the relnote tool
- `DO NOT SUBMIT` (in the commit message) — block submission; see the "Work in progress" section below
- `Updates #1234` or `Fixes #1234` (in the commit message) — link the CL from the GitHub issue and optionally close the issue after the CL is merged

Email

Messages from a code review are typically sent to three places:

- the reviewers, if any
- the golang-codereviews group
- the owner

Please do NOT reply code review via email, as the message will not be relayed to Gerrit. Always click on the link and post reply in Gerrit.

Work in progress

If you have changes that aren't ready to be reviewed, you can put a giant `DO NOT REVIEW` as the second line of the CL description, so that people who see it know not to look any further. Don't make it the first line, since then it will become the subject for the entire review, even after you've changed the description.

Similarly, if you would like to ensure that your changes are not merged by mistake, you can put `DO NOT SUBMIT` as the second line of the CL description.

If you don't need Gerrit's features, but just want to backup your work, share work between multiple clients, or have a staging UI to examine your changes, you can use a regular git remote.

To use GitHub as a git remote, you can either fork github.com/golang/go or create a new repo. There are trade-offs. Forked repos will have a faster first push. Non-forked repos can be private. Forked repos are associated in GitHub's system. As a result, they are easily discoverable and support cross-repo comparisons in the GitHub UI; however, this also means that references to issues in commit messages in forked repos will create references to your fork in the issue.

To add your git remote, run something like:

```
$ git remote add fork git@github.com:yourusername/go.git
```

You can then push changes to the "fork" remote with `git push fork branchname`.

Gerrit's code review model is to rewrite a single commit until it is correct. GitHub will try to prevent you from accidentally overwriting your existing branch. You can work around this by forcing the push: `git push --force fork branchname`. Alternatively, you can set up your forked remote as a mirror by cloning it initially with:

```
$ git remote add --mirror=push fork git@github.com:yourusername/go.git
```

Then running `git push fork` will update GitHub to perfectly mirror *everything* (all branches, all tags, etc.). This is handy, but take care when using this on multiple clients. You are bypassing the usual git safeguards, so it is easy to overwrite (and thus lose) work pushed by a different client.