

publishing



- [Publishing steps](#)
-

When you want to make a module available for other developers, you publish it so that it's visible to Go tools. Once you've published the module, developers importing its packages will be able to resolve a dependency on the module by running commands such as `go get`.

“ **Note:** Don't change a tagged version of a module after publishing it. For developers using the module, Go tools authenticate a downloaded module against the first downloaded copy. If the two differ, Go tools will return a security error. Instead of changing the code for a previously published version, publish a new version.

See also

- For an overview of module development, see [Developing and publishing modules](#)
- For a high-level module development workflow -- which includes publishing -- see [Module release and versioning workflow](#).

Publishing steps

Use the following steps to publish a module.

1. Open a command prompt and change to your module's root directory in the local repository.
2. Run `go mod tidy`, which removes any dependencies the module might have accumulated that are no longer necessary.

```
$ go mod tidy
```

3. Run `go test ./...` a final time to make sure everything is working.
This runs the unit tests you've written to use the Go testing framework.

```
$ go test ./...  
ok      example.com/mymodule      0.015s
```

4. Tag the project with a new version number using the `git tag` command. For the version number, use a number that signals to users the nature of changes in this release. For more, see [Module version numbering](#).

```
$ git commit -m "mymodule: changes for v0.1.0"  
$ git tag v0.1.0
```

5. Push the new tag to the origin repository.

```
$ git push origin v0.1.0
```

6. Make the module available by running the `go list` command to prompt Go to update its index of modules with information about the module you're publishing. Precede the command with a statement to set the `GOPROXY` environment variable to a Go proxy. This will ensure that your request reaches the proxy.

```
$ GOPROXY=proxy.golang.org go list -m example.com/mymodule@v0.1.0
```

Developers interested in your module import a package from it and run the `go get` command just as they would with any other module. They can run the `go get` command for latest versions or they can specify a particular version, as in the following example:

```
$ go get example.com/mymodule@v0.1.0
```

```
## #2  
[] [] [] 2026-10-26 20:26:29 UTC  
[] [] [] 2026-10-26 20:26:29 UTC
```