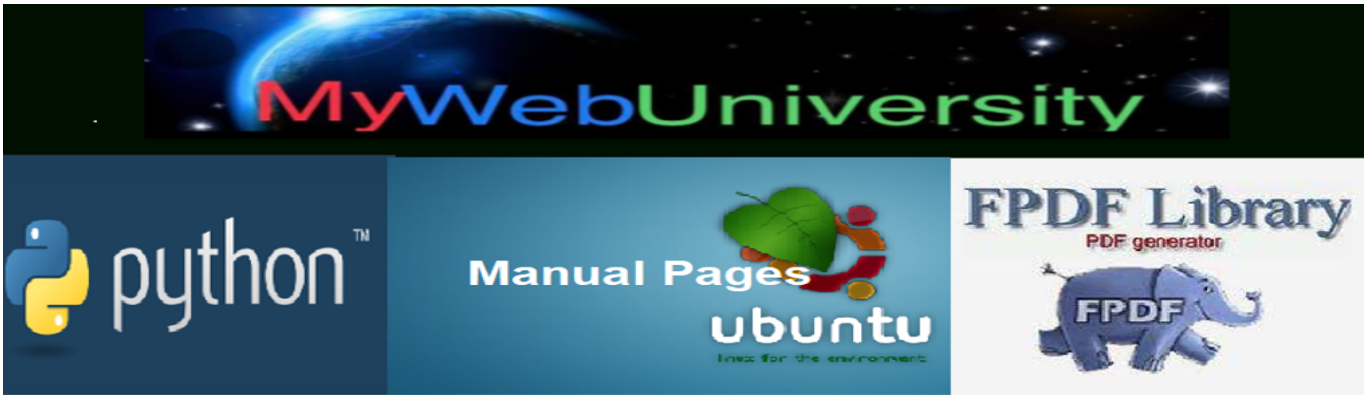

Rocky Enterprise Linux 9.2 Manual Pages on command 'go-list.1'

$ man go-list.1

GO-LIST(1) General Commands Manual GO-LIST(1)

NAME

go-list - list packages or modules

SYNOPSIS

go list [-f format] [-json] [-m] [list flags] [build flags] [packages]

DESCRIPTION

List lists the packages named by the import paths, one per line.

The most commonly-used flags are -f and -json, which control the form of the output printed for each package. Other list flags, documented below, control more specific de? tails.

The default output shows the package import path:

bytes

encoding/json

github.com/gorilla/mux

golang.org/x/net/html

OPTIONS

-f The -f flag specifies an alternate format for the list, using the syntax of package template. The default output is equivalent to -f '{{.ImportPath}}'. The struct being passed to the template is:

```
type Package struct {
    Dir          string   // directory containing package sources
    ImportPath   string   // import path of package in dir
    ImportComment string  // path in import comment on package statement
```

```
Name          string   // package name
Doc           string   // package documentation string
Target        string   // install path
Shlib         string   // the shared library that contains this package (only set when -linkshared)
Goroot        bool     // is this package in the Go root?
Standard      bool     // is this package part of the standard Go library?
Stale         bool     // would 'go install' do anything for this package?
StaleReason   string   // explanation for Stale==true
Root          string   // Go root or Go path dir containing this package
ConflictDir   string   // this directory shadows Dir in $GOPATH
BinaryOnly    bool     // binary-only package (no longer supported)
ForTest       string   // package is only for use in named test
Export        string   // file containing export data (when using -export)
BuildID       string   // build ID of the compiled package (when using -export)
Module        *Module  // info about package's containing module, if any (can be nil)
Match         []string // command-line patterns matching this package
DepOnly       bool     // package is only a dependency, not explicitly listed
// Source files
GoFiles       []string   // .go source files (excluding CgoFiles, TestGoFiles, XTestGoFiles)
CgoFiles      []string   // .go source files that import "C"
CompiledGoFiles []string   // .go files presented to compiler (when using -compiled)
IgnoredGoFiles  []string   // .go source files ignored due to build constraints
IgnoredOtherFiles []string // non-.go source files ignored due to build constraints
CFiles        []string   // .c source files
CXXFiles      []string   // .cc, .cxx and .cpp source files
MFiles        []string   // .m source files
HFiles        []string   // .h, .hh, .hpp and .hxx source files
FFiles        []string   // .f, .F, .for and .f90 Fortran source files
SFiles        []string   // .s source files
SwigFiles     []string   // .swig files
SwigCXXFiles  []string   // .swigcxx files
SysoFiles     []string   // .syso object files to add to archive
TestGoFiles   []string   // _test.go files in package
```

```
        XTestGoFiles    []string   // _test.go files outside package
        // Embedded files
        EmbedPatterns      []string // //go:embed patterns
        EmbedFiles         []string // files matched by EmbedPatterns
        TestEmbedPatterns  []string // //go:embed patterns in TestGoFiles
        TestEmbedFiles     []string // files matched by TestEmbedPatterns
        XTestEmbedPatterns []string // //go:embed patterns in XTestGoFiles
        XTestEmbedFiles    []string // files matched by XTestEmbedPatterns
        // Cgo directives
        CgoCFLAGS    []string // cgo: flags for C compiler
        CgoCPPFLAGS  []string // cgo: flags for C preprocessor
        CgoCXXFLAGS  []string // cgo: flags for C++ compiler
        CgoFFLAGS    []string // cgo: flags for Fortran compiler
        CgoLDFLAGS   []string // cgo: flags for linker
        CgoPkgConfig []string // cgo: pkg-config names
        // Dependency information
        Imports      []string          // import paths used by this package
        ImportMap    map[string]string // map from source import to ImportPath (identity entries omitted)
        Deps         []string          // all (recursively) imported dependencies
        TestImports  []string          // imports from TestGoFiles
        XTestImports []string          // imports from XTestGoFiles
        // Error information
        Incomplete bool            // this package or a dependency has an error
        Error      *PackageError   // error loading package
        DepsErrors []*PackageError // errors loading dependencies
    }
```

Packages stored in vendor directories report an ImportPath that includes the path to the vendor directory (for example, "d/vendor/p" instead of "p"), so that the Im? portPath uniquely identifies a given copy of a package. The Imports, Deps, TestIm? ports, and XTestImports lists also contain these expanded import paths. See golang.org/s/go15vendor for more about vendoring.

The error information, if any, is

```
    type PackageError struct {
```

```
        ImportStack   []string // shortest path from package named on command line to this one
        Pos           string   // position of error (if present, file:line:col)
        Err           string   // the error itself
    }
```

The module information is a Module struct, defined in the discussion of list -m be?
low.

The template function "join" calls strings.Join.

The template function "context" returns the build context, defined as:

```
    type Context struct {
        GOARCH        string   // target architecture
        GOOS          string   // target operating system
        GOROOT        string   // Go root
        GOPATH        string   // Go path
        CgoEnabled    bool     // whether cgo can be used
        UseAllFiles   bool     // use files regardless of +build lines, file names
        Compiler      string   // compiler to assume when computing target paths
        BuildTags     []string // build constraints to match in +build lines
        ToolTags      []string // toolchain-specific build constraints
        ReleaseTags   []string // releases the current release is compatible with
        InstallSuffix string   // suffix to use in the name of the install dir
    }
```

For more information about the meaning of these fields see the documentation for
the go/build package's Context type.

-json The -json flag causes the package data to be printed in JSON format instead of us?
ing the template format.

-compiled

The -compiled flag causes list to set CompiledGoFiles to the Go source files pre?
sented to the compiler. Typically this means that it repeats the files listed in
GoFiles and then also adds the Go code generated by processing CgoFiles and Swig?
Files. The Imports list contains the union of all imports from both GoFiles and
CompiledGoFiles.

-deps The -deps flag causes list to iterate over not just the named packages but also all
their dependencies. It visits them in a depth-first post-order traversal, so that a

package is listed only after all its dependencies. Packages not explicitly listed on the command line will have the DepOnly field set to true.

-e The -e flag changes the handling of erroneous packages, those that cannot be found or are malformed. By default, the list command prints an error to standard error for each erroneous package and omits the packages from consideration during the usual printing. With the -e flag, the list command never prints errors to standard error and instead processes the erroneous packages with the usual printing. Erro? neous packages will have a non-empty ImportPath and a non-nil Error field; other information may or may not be missing (zeroed).

-export

The -export flag causes list to set the Export field to the name of a file contain? ing up-to-date export information for the given package.

-find The -find flag causes list to identify the named packages but not resolve their de? pendencies: the Imports and Deps lists will be empty.

-test The -test flag causes list to report not only the named packages but also their test binaries (for packages with tests), to convey to source code analysis tools exactly how test binaries are constructed. The reported import path for a test bi? nary is the import path of the package followed by a ".test" suffix, as in "math/rand.test". When building a test, it is sometimes necessary to rebuild cer? tain dependencies specially for that test (most commonly the tested package it? self). The reported import path of a package recompiled for a particular test bi? nary is followed by a space and the name of the test binary in brackets, as in "math/rand [math/rand.test]" or "regexp [sort.test]". The ForTest field is also set to the name of the package being tested ("math/rand" or "sort" in the previous ex? amples).

The Dir, Target, Shlib, Root, ConflictDir, and Export file paths are all absolute paths.

By default, the lists GoFiles, CgoFiles, and so on hold names of files in Dir (that is, paths relative to Dir, not absolute paths). The generated files added when using the -compiled and -test flags are absolute paths referring to cached copies of generated Go source files. Although they are Go source files, the paths may not end in ".go".

-m The -m flag causes list to list modules instead of packages.

When listing modules, the -f flag still specifies a format template applied to a Go struct, but now a Module struct:

```
type Module struct {
    Path      string       // module path
    Version   string       // module version
    Versions  []string     // available module versions (with -versions)
    Replace   *Module      // replaced by this module
    Time      *time.Time   // time version was created
    Update    *Module      // available update, if any (with -u)
    Main      bool         // is this the main module?
    Indirect  bool         // is this module only an indirect dependency of main module?
    Dir       string       // directory holding files for this module, if any
    GoMod     string       // path to go.mod file used when loading this module, if any
    GoVersion string       // go version used in module
    Error     *ModuleError // error loading module
}
type ModuleError struct {
    Err string // the error itself
}
```

The file GoMod refers to may be outside the module directory if the module is in the module cache or if the -modfile flag is used.

The default output is to print the module path and then information about the version and replacement if any. For example, 'go list -m all' might print:

```
my/main/module
golang.org/x/text v0.3.0 => /tmp/text
rsc.io/pdf v0.1.1
```

The Module struct has a String method that formats this line of output, so that the default format is equivalent to -f '{{.String}}'.

Note that when a module has been replaced, its Replace field describes the replacement module, and its Dir field is set to the replacement's source code, if present. (That is, if Replace is non-nil, then Dir is set to Replace.Dir, with no access to the replaced source code.)

-u The -u flag adds information about available upgrades. When the latest version of a given module is newer than the current one, list -u sets the Module's Update field to information about the newer module. list -u will also set the module's Re?

tracted field if the current version is retracted. The Module's String method in?
dicates an available upgrade by formatting the newer version in brackets after the
current version. If a version is retracted, the string "(retracted)" will follow
it. For example, 'go list -m -u all' might print:

my/main/module
golang.org/x/text v0.3.0 [v0.4.0] => /tmp/text
rsc.io/pdf v0.1.1 (retracted) [v0.1.2]

(For tools, 'go list -m -u -json all' may be more convenient to parse.)

-versions

The -versions flag causes list to set the Module's Versions field to a list of all
known versions of that module, ordered according to semantic versioning, earliest
to latest. The flag also changes the default output format to display the module
path followed by the space-separated version list.

-retracted

The -retracted flag causes list to report information about retracted module ver?
sions. When -retracted is used with -f or -json, the Retracted field will be set to
a string explaining why the version was retracted. The string is taken from com?
ments on the retract directive in the module's go.mod file. When -retracted is used
with -versions, retracted versions are listed together with unretracted versions.
The -retracted flag may be used with or without -m.

The arguments to list -m are interpreted as a list of modules, not packages. The main module is the module containing the current directory. The active modules are the main module and its dependencies. With no arguments, list -m shows the main module. With ar? guments, list -m shows the modules specified by the arguments. Any of the active modules can be specified by its module path. The special pattern "all" specifies all the active modules, first the main module and then dependencies sorted by module path. A pattern containing "..." specifies the active modules whose module paths match the pattern. A query of the form path@version specifies the result of that query, which is not limited to active modules. See 'go help modules' for more about module queries.

The template function "module" takes a single string argument that must be a module path or query and returns the specified module as a Module struct. If an error occurs, the re? sult will be a Module struct with a non-nil Error field.

For more about build flags, see go-build(1) or 'go help build'.

For more about specifying packages, see go-packages(7) or 'go help packages'.

For more about modules, see https://golang.org/ref/mod.

AUTHOR

This manual page was written by Michael Stapelberg <stapelberg@debian.org> and is main? tained by the Debian Go Compiler Team <team+go-compiler@tracker.debian.org> based on the output of 'go help list' for the Debian project (and may be used by others).

2021-10-15 GO-LIST(1)