

---

## setup-go



This action sets up a go environment for use in actions by:

- Optionally downloading and caching a version of Go by version and adding to [PATH](#).
- Registering problem matchers for error output.

### V5

The V5 edition of the action offers:

- Upgraded Node.js runtime from node16 to node20

See full release notes on the releases page.

### V4

The V4 edition of the action offers:

- Enabled caching by default

The action will try to enable caching unless the [cache](#) input is explicitly set to false.

Please see “Caching dependency files and build outputs” for more information.

### V3

The V3 edition of the action offers:

- Adds [GOBIN](#) to the [PATH](#)
- Proxy support
- Check latest version
- Caching packages dependencies
- stable and oldstable aliases
- Bug Fixes (including issues around version matching and semver)

---

The action will first check the local cache for a version match. If a version is not found locally, it will pull it from the [main](#) branch of the go-versions repository. On miss or failure, it will fall back to downloading directly from go dist. To change the default behavior, please use the check-latest input.

**Note:** The `setup-go` action uses executable binaries which are built by Golang side. The action does not build golang from source code.

Matching by semver spec:

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '^1.13.1' # The Go version to download (if necessary)
6       and use.
7   - run: go version
```

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '>=1.17.0'
6   - run: go version
```

**Note:** Due to the peculiarities of YAML parsing, it is recommended to wrap the version in single quotation marks:

```
1 go-version: '1.20'
```

The recommendation is based on the YAML parser's behavior, which interprets non-wrapped values as numbers and, in the case of version 1.20, trims it down to 1.2, which may not be very obvious.

Matching an unstable pre-release:

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '1.18.0-rc.1' # The Go version to download (if
6       necessary) and use.
7   - run: go version
```

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '1.16.0-beta.1' # The Go version to download (if
6       necessary) and use.
```

---

```
6 - run: go version
```

## Usage

See action.yml

### Basic

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '1.16.1' # The Go version to download (if necessary)
6       and use.
6   - run: go run hello.go
```

### Check latest version

The `check-latest` flag defaults to **false**. Use the default or set `check-latest` to **false** if you prefer stability and if you want to ensure a specific Go version is always used.

If `check-latest` is set to **true**, the action first checks if the cached version is the latest one. If the locally cached version is not the most up-to-date, a Go version will then be downloaded. Set `check-latest` to **true** if you want the most up-to-date Go version to always be used.

Setting `check-latest` to **true** has performance implications as downloading Go versions is slower than using cached versions.

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '1.14'
6       check-latest: true
7   - run: go run hello.go
```

### Using stable/oldstable aliases

If `stable` is provided, action will get the latest stable version from the `go-versions` repository manifest.

---

If `oldstable` is provided, when current release is 1.19.x, action will resolve version as 1.18.x, where x is the latest patch release.

**Note:** using these aliases will result in same version as using corresponding minor release with `check-latest` input set to `true`

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: 'stable'
6   - run: go run hello.go
```

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: 'oldstable'
6   - run: go run hello.go
```

## Caching dependency files and build outputs:

The action has a built-in functionality for caching and restoring go modules and build outputs. It uses `toolkit/cache` under the hood but requires less configuration settings. The `cache` input is optional, and caching is turned on by default.

The action defaults to search for the dependency file - `go.sum` in the repository root, and uses its hash as a part of the cache key. Use `cache-dependency-path` input for cases when multiple dependency files are used, or they are located in different subdirectories. The input supports glob patterns.

If some problem that prevents success caching happens then the action issues the warning in the log and continues the execution of the pipeline.

## Caching in monorepos

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version: '1.17'
6       check-latest: true
7       cache-dependency-path: |
8         subdir/go.sum
9         tools/go.sum
10      # cache-dependency-path: "**/*.sum"
11
```

---

```
12 - run: go run hello.go
```

## Getting go version from the go.mod file

The `go-version-file` input accepts a path to a `go.mod` file or a `go.work` file that contains the version of Go to be used by a project.

The `go` directive in `go.mod` can specify a patch version or omit it altogether (e.g., `go 1.22.0` or `go 1.22`).

If a patch version is specified, that specific patch version will be used.

If no patch version is specified, it will search for the latest available patch version in the cache, `versions-manifest.json`, and the official Go language website, in that order.

If both the `go-version` and the `go-version-file` inputs are provided then the `go-version` input is used. > The action will search for the `go.mod` file relative to the repository root

```
1 steps:
2   - uses: actions/checkout@v4
3   - uses: actions/setup-go@v5
4     with:
5       go-version-file: 'path/to/go.mod'
6   - run: go version
```

## Matrix testing

```
1 jobs:
2   build:
3     runs-on: ubuntu-latest
4     strategy:
5       matrix:
6         go: [ '1.14', '1.13' ]
7     name: Go ${ matrix.go } sample
8     steps:
9       - uses: actions/checkout@v4
10      - name: Setup go
11        uses: actions/setup-go@v5
12        with:
13          go-version: ${ matrix.go }
14      - run: go run hello.go
```

## Supported version syntax

The `go-version` input supports the following syntax:

- 
- Specific versions: 1.15, 1.16.1, 1.17.0-rc.2, 1.16.0-beta.1
  - SemVer's version range syntax: ^1.13.1, >=1.18.0-rc.1

For more information about semantic versioning, please refer to semver documentation.

## Using setup-go on GHES

setup-go comes pre-installed on the appliance with GHES if Actions is enabled. When dynamically downloading Go distributions, setup-go downloads distributions from [actions/go-versions](#) on github.com (outside of the appliance). These calls to [actions/go-versions](#) are made via unauthenticated requests, which are limited to 60 requests per hour per IP. If more requests are made within the time frame, then you will start to see rate-limit errors during downloading that looks like: `##[error]API rate limit exceeded for...` After that error the action will try to download versions directly from <https://storage.googleapis.com/golang>, but it also can have rate limit so it's better to put token.

To get a higher rate limit, you can generate a personal access token on github.com and pass it as the `token` input for the action:

```
1 uses: actions/setup-go@v5
2 with:
3   token: ${{ secrets.GH_DOTCOM_TOKEN }}
4   go-version: '1.18'
```

If the runner is not able to access github.com, any Go versions requested during a workflow run must come from the runner's tool cache. See "Setting up the tool cache on self-hosted runners without internet access" for more information.

## License

The scripts and documentation in this project are released under the MIT License

## Contributions

Contributions are welcome! See Contributor's Guide

## Code of Conduct

:wave: Be nice. See our code of conduct