# Understanding Module Versioning

Michael Van Sickle

@vansimke

# Overview

Semantic versioning

Module versioning rules

Module queries

# Semantic Versioning

**v1.5.3-pre1**

# Semantic Versioning

# v1.5.3-pre1

| | |
|---|---|
| v | Version prefix (required) |
| 1 | Major revision (likely to break backward compatibility) |
| 5 | Minor revision (new features, doesn't break BC) |
| 3 | Patch (bug fixes, no new features, and doesn't break BC) |
| pre1 | Pre-release of new version, if applicable (text is arbitrary) |

https://semver.org

# Module Versioning Rules

v1 and earlier

v2+

Unversioned commits

Normally tied to a release/tag in source control

# Versioning Rules – v1 and Earlier

No promise of backward compatibility prior to v1.0.0

Precedence determined by major, minor, then patch versions

```
import github.com/gorilla/mux
```

# Versioning Rules – v2+

Backward compatibility should be preserved within a major version

Each major version has unique import path

```
import github.com/gorilla/mux/v2
```

# Versioning Rules – Unversioned Commits

Still uses semver

Prerelease identifier

- Timestamp
- Commit hash

```
require golang.org/x/tools v0.0.0-20180917221912-90fa682c2a6e
```

# Module Queries

| | | |
|---|---|---|
| Specific version<br>@v1.7.2 | Version prefix<br>@v1 | Latest<br>@latest |
| Specific commit<br>@c956192 | Specific commit<br>@master | Comparison<br>@>=1.7.2 |

# Comparison Rules

Closest match wins

Prereleases have lower precedence

Closest match wins

This is important enough to repeat

# Summary

**Semantic versioning**

**Module versioning rules**

- Prior to v1
- v1
- v2+
- Unversioned commits

**Module queries**

- Closest match wins