

authors :

- s. Lavazais
- d. Six

sources:

- [https://en.wikipedia.org/wiki/Dependency\\_hell](https://en.wikipedia.org/wiki/Dependency_hell)
- <https://semver.org/>
- <https://github.com/semantic-release/semantic-release>

# SEMANTIC VERSION

Semantic version is the set of rules dictating how version numbers of an API are assigned and incremented. Based on but not limited to common practices of closed and open source software development.

Consider a version like `X.Y.Z` where `X` is `MAJOR` number, `Y` is `MINOR` number and `Z` is `PATCH` number.

- Bug fixes represent the `PATCH` number with obvious backward compatibility.
- Addition/changes with backward compatibility increment the `MINOR` number.
- and any other changes with backward incompatibility increment the `MAJOR` number.

# SUMMARY

```
version x.y.z
      ^  ^  ^
      ||  ||  ||
      ||  ||  || Patch : backward compatible bug fixes
      ||  ||  || Minor : features backward compatible manner
      ||  ||  || Major : breaking changes API
```

When version numbers are not standardized, solving a simple issue of a common dependency can be a real nightmare, since there is no common way to increment version of dependents libs

# DEPENDENCY HELL

```
1 my-app 4.7.2
2 |└─ internal lib 5.6.1
3 |  |└─ logger lib 2016.3.4
4 |  |  |└─ another external lib 0.3.0
5 |└─ database lib 12(cookie)
6 |  |└─ io lib 3.0.0
7 |  |  |└─ another external lib 0.3.0
8 |└─ list lib 0.1.2alpha
9 |  |└─ external lib 4.8
10 |  |  |└─ another external lib 0.3.0
```

# SET OF RULES

# PUBLIC API

only after the release of version 1.0.0

## Speaker notes

any application/lib must declare a public API (it could be simple documentation). It must be only define after the first major version.

- [rule #1](#)
- [rule #5](#)

# X.Y.Z FORM

1.2.3

^ ^ ^

Patch

Minor

Major

Speaker notes

a normal version must be formatted with 3 non-negative integer separated by dot. (not lead by 0)

each integer represent a different type of modification of the application/lib imply.

- rule #2

# PRE-RELEASE RULES

init dev start with version "0"

```
0.x.x
```

## Speaker notes

initial development start at version 0.x.x, the increment of the version a state can occur any time and the version is not stable.

- [rule #4](#)

# PRE-RELEASE RULES

add "–" and other identifiers separated by "."

```
1.1.2-alpha.1  
1.1.2-5.7.9
```

pre-released version can be defined by adding identifiers after the patch integer lead by a hyphen.

then each identifier must be separated by a dot.

- [rule #9](#)



versions for build metadata should be separated from the rest of the version by adding a plus sign

- [rule #10](#)

# BUILD METADATA

can add "+" and other identifiers separated by "."

```
1.1.2+012  
1.1.2+21AF26D3  
1.1.2-beta+exp.sha.749f34
```

any modification after the first release must imply a modification of the version

- [rule #3](#)

# INCREMENT RULES

any modification after first release is a new version

1.2.3

# INCREMENT RULES

Patch version

bug fixes (backward compatible)

1.2.3

- [rule #6](#)

# INCREMENT RULES

Patch version

bug fixes (backward compatible)

1 . 2 . 4

# INCREMENT RULES

Minor version

new features (backward compatible)

1 . 2 . 4

# INCREMENT RULES

Minor version

new features (backward compatible)

1.3.0

## Speaker notes

adding a new features should increment the minor version, only if these modifications has not broken the backward compatibility of the application/lib

this incrementation imply that the patch version is reset to 0

- [rule #7](#)

# INCREMENT RULES

Major version

any backward incompatible changes

1.3.0

any breaking changes (that broke the backward compatibility) should increment the major version,

this incrementation imply that the patch and the minor versions are reset to 0

- [rule #8](#)

# INCREMENT RULES

Major version

any backward incompatible changes

2.0.0



# PRECEDENCE RULES

```
1.0.0 < 2.0.0 < 2.1.0 < 2.1.1
```

## Speaker notes

the precedences rules define how versions are compared to each other

the precedence must be calculated by separating the version into major, minor, patch and pre-release identifiers in that order (Build metadata does not figure into precedence).

- [rule #11](#)

pre-released version has lower precedence compared to normal version

- [rule #11](#)

# PRECEDENCE RULES

## with pre-release

```
1.0.0-alpha < 1.0.0
```

# PRECEDENCE RULES

## with pre-release

```
1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta  
1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0
```

precedence of pre-released versions of same core version must determine by comparing each dot separated identifier from left to right until a difference is found.

- [rule #11](#)

Semantic Versioning can be easily implemented to any project by using tools like semantic-release

semantic-release parse commit messages to build a version number. by default this tool will build the first release if no tag of version exist (1.0.0)

# USING SEMANTIC-RELEASE



```
verb(scope) : message
```

```
notes
```

```
fix(security): fix security check
```

```
feat(security): add security standard
```

```
feat(security): new security standard
```

```
BREAKING CHANGES: don't support old security standard
```

# CONFIG SEMANTIC-RELEASE

```
1  {
2    "branches": ["main"],
3    "tagFormat": "${version}",
4    "plugins": [
5      "@semantic-release/commit-analyzer",
6      "@semantic-release/exec",
7      "@semantic-release/release-notes-generator",
8      "@semantic-release/changelog",
9      "@semantic-release/git",
10     "@semantic-release/github"
11   ]
12 }
```

## Speaker notes

Semantic Release (SR) use a simple file to run (releaserc). here an example of settings.

setting `branches` are the release branches. setting `tagFormat` is the format of tag (when a release occur). `plugins` allow SR to manipulate different aspect on the release mechanism

we're gonna tweak `exec` and `git` to allow us to add more parameters to our releases.

on `exec`, we describe what we do just before the release.

on `git`, we describe what we commit on release, and the commit message

you can refer to external script to make what you want with the execution script

# CONFIG SEMANTIC-RELEASE

```
[ "@semantic-release/exec", {  
  "prepareCmd": "VERSION=${nextRelease.version} make exec-release",  
}]
```

```
exec-release: ## Execution of a new release  
  ./mvnw -q versions:set -DnewVersion=${VERSION}  
  sed -Ei 's/version:./version: ${VERSION}/g' openapi.yaml
```

```
[ "@semantic-release/git", {  
  "assets": ["CHANGELOG.md", "pom.xml", "openapi.yaml"],  
  "message": "chore(release): version ${nextRelease.version}"  
}]
```

# RUN SEMANTIC-RELEASE

```
npx semantic-release
```

# RUN SEMANTIC-RELEASE

```
[2:50:44 PM] [semantic-release] > i Start step "analyzeCommits" of plugin "@semantic-release/commit-analyzer"  
[2:50:44 PM] [semantic-release] [@semantic-release/commit-analyzer] > i Analyzing commit: feat(discuss): add swagger file example  
[2:50:44 PM] [semantic-release] [@semantic-release/commit-analyzer] > i The release type for the commit is minor  
[2:50:44 PM] [semantic-release] [@semantic-release/commit-analyzer] > i Analysis of 1 commits complete: minor release  
[2:50:44 PM] [semantic-release] > ✓ Completed step "analyzeCommits" of plugin "@semantic-release/commit-analyzer"
```

```
[2:50:45 PM] [semantic-release] > ✓ Published release 1.9.0 on default channel  
[2:50:45 PM] [semantic-release] > i Release note for version 1.9.0:  
# 1.9.0 (2022-12-18)
```

## ### Features

- \* **discuss:** add swagger file example (6367d74)



# GOING FURTHER - PRE-RELEASE MANAGEMENT

pre-release branch management

```
{
  "branches": [
    "main",
    {
      "name": "pre-*",
      "prerelease": true
    }
  ]
}
```

# GOING FURTHER - ORDER MATTER

## plugins order matter

```
{
  "branches": ["main"],
  "tagFormat": "${version}",
  "plugins": [
    "@semantic-release/commit-analyzer",
    "@semantic-release/exec",
    "@semantic-release/release-notes-generator",
    "@semantic-release/git", //will commit without changelog
    "@semantic-release/changelog",
    "@semantic-release/github"
  ]
}
```

# GOING FURTHER

## release steps mechanism

- `verifyConditions`
- `analyzeCommits`
- `verifyRelease`
- `generateNotes`
- `prepare`
- `publish`
- `addChannel`
- `success`
- `fail`

# THANK YOU

Authors:

**S. LAVAZAIS**

**D. SIX**

Sources:

**[HTTPS://EN.WIKIPEDIA.ORG/WIKI/DEPENDENCY\\_HELL](https://en.wikipedia.org/wiki/Dependency_Hell)**

**[HTTPS://SEMVER.ORG/](https://semver.org/)**

**[HTTPS://GITHUB.COM/SEMANTIC-RELEASE/SEMANTIC-RELEASE](https://github.com/semantic-release/semantic-release)**

**[HTTPS://GITHUB.COM/ANGULAR/MASTER/CONTRIBUTING.MD](https://github.com/angular/master/contributing.md)**