# Buenas Prácticas para Definir Versionamientos

## Pregunta:

Explique cuáles son las buenas prácticas para definir los versionamientos de una aplicación, incluyendo letras, números y palabras reservadas.

## Respuesta:

### 1. Usar Versionado Semántico (SemVer)

El **Versionado Semántico** es una de las prácticas más comunes y recomendadas para versionar software. Su formato general es:

```
MAJOR.MINOR.PATCH
```

- **MAJOR** (Cambios grandes): Incrementa cuando realizas cambios incompatibles con versiones incompatibles con versiones anteriores o introduces nuevas características que rompen la compatibilidad.
- **MINOR** (Nuevas funcionalidades): Incrementa cuando añades nuevas características de manera compatible con versiones anteriores.
- **PATCH** (Corrección de errores): Incrementa cuando haces correcciones menores y arreglos de bugs que no afectan la compatibilidad.

Ejemplo:

- `2.0.0`: Cambios incompatibles con la versión anterior (por ejemplo, nueva arquitectura de la API).
- `2.1.0`: Nuevas funcionalidades, pero es compatible con `2.0.0`.
- `2.1.1`: Corrección de un bug en la versión `2.1.0`.

### 2. Uso de etiquetas adicionales

En versiones preliminares (antes del lanzamiento oficial) o para lanzar versiones especiales, puedes usar etiquetas adicionales con letras o palabras clave. Esto es útil para identificar el estado de desarrollo de la versión.

**Formato:**

```
MAJOR.MINOR.PATCH-prerelease+buildmetadata
```

- **Prerelease**: Se añade para indicar que es una versión inestable o en pruebas. Algunos ejemplos incluyen:
    - `alpha`: Versión temprana, aún inestable.
    - `beta`: Versión más estable que alpha, pero todavía en pruebas.

    - `rc` (Release Candidate): Una versión que puede ser liberada como final, a menos que se encuentren problemas graves.

  Ejemplo:

    - `2.1.0-alpha.1`: Primera versión alpha de la versión `2.1.0`.
    - `2.1.0-beta.1`: Primera versión beta de `2.1.0`.
    - `2.1.0-rc.1`: Primera release candidate de `2.1.0`.

- **Build Metadata**: Información adicional para identificar un build específico (no afecta la precedencia). Suele usarse para identificar builds internos o compilaciones automáticas en un entorno de CI/CD.

  Ejemplo:

    - `2.1.0-alpha.1+build123`: Versión alpha del build `123`.

### 3. Evitar usar fechas como versionado principal

Aunque usar fechas puede parecer conveniente, no es ideal como método de versionamiento primario. Las fechas no informan al usuario sobre la compatibilidad de la versión o los cambios realizados, lo cual es crucial en el desarrollo de software.

Sin embargo, puedes usar fechas como **metadata** o en el nombre del release para una referencia adicional.

**Ejemplo de uso incorrecto como versión principal:**

```
2023.09.14
```

**Uso recomendado como metadata o junto a versionado semántico:**

```
2.1.0+20230914
```

### 4. Palabras clave para versiones especiales

Existen convenciones bien aceptadas para etiquetar versiones especiales, pero es importante no abusar de ellas. Algunas palabras clave útiles incluyen:

- **LTS (Long-Term Support)**: Usado para indicar que una versión recibirá soporte a largo plazo. Esta etiqueta se utiliza cuando una versión específica será mantenida por un período extenso.
    - Ejemplo: `2.1.0-lts`
- **Legacy**: Usado para indicar una versión antigua o que ya no recibe actualizaciones nuevas, pero que sigue disponible por compatibilidad.
    - Ejemplo: `1.9.5-legacy`
- **Stable**: Aunque rara vez se usa explícitamente, podría aparecer para destacar que una versión es estable y lista para producción.

- Ejemplo: `2.1.0-stable`

### 5. Mantener un ciclo de vida de versiones claro

Es importante definir y comunicar cómo gestionas las versiones en tu aplicación. Esto incluye cuándo una versión será considerada **obsoleta** (deprecated), cuándo dejarás de darle soporte, y cómo los usuarios deberían migrar a una nueva versión.

**Buenas prácticas:**

- **Obsolescencia**: Anuncia con tiempo cuándo una versión será descontinuada o reemplazada por una nueva versión.
- **Documentación clara**: Proporciona documentación para que los usuarios sepan cómo migrar a la nueva versión y los cambios clave.

**Ejemplo de ciclo de vida:**

- `1.0.0`: Versión inicial.
- `1.1.0`: Se anuncian mejoras y nuevas funcionalidades.
- `1.2.0`: Se agrega funcionalidad y se avisa de la obsolescencia de la versión `1.0.0`.
- `2.0.0`: Nueva versión con cambios que rompen compatibilidad. La versión `1.0.0` deja de recibir soporte en 6 meses.

### 6. Versionar la API

Cuando versionas una API, es una buena práctica incluir el número de versión en la URL del endpoint o en los encabezados de la solicitud.

**Versionamiento en la URL:**

```
https://api.tuapp.com/v1/libros
https://api.tuapp.com/v2/libros
```

En este caso, `v1` o `v2` indica la versión de la API.

**Versionamiento en encabezados:**

Puedes definir la versión de la API en los encabezados de la solicitud HTTP, lo cual hace la URL más limpia:

```
GET /libros
Headers:
  Accept: application/vnd.tuapp.v1+json
```

### 7. Evitar uso de letras o palabras ambiguas en las versiones

Es una buena práctica evitar usar letras o palabras que puedan causar confusión. En lugar de etiquetas ambiguas, utiliza etiquetas bien conocidas, como `alpha`, `beta`, `rc`, o `legacy`. Esto facilita que cualquier

desarrollador entienda el propósito de la versión.

### 8. Evitar saltos grandes en la numeración

Los saltos innecesarios en los números de versión pueden confundir a los usuarios. Si la versión actual es `2.1.0`, evita saltar directamente a `4.0.0` sin una justificación clara (como un cambio radical en la funcionalidad o arquitectura).

### Ejemplo completo de versionamiento semántico:

1. **Primera versión pública estable**:
    - `1.0.0`
2. **Primera actualización menor con nuevas características**:
    - `1.1.0`
3. **Corrección de errores en la versión `1.1.0`**:
    - `1.1.1`
4. **Versiones inestables para pruebas de la versión `2.0.0`**:
    - `2.0.0-alpha.1`
    - `2.0.0-beta.1`
    - `2.0.0-rc.1`
5. **Versión final compatible pero con grandes cambios**:
    - `2.0.0`
6. **Actualización de seguridad para la versión `2.0.0`**:
    - `2.0.1`

## Resumen de buenas prácticas:

1. **Usa versionado semántico** (MAJOR.MINOR.PATCH).
2. **Usa etiquetas adicionales** como `alpha`, `beta`, `rc`, y metadata para builds.
3. **Evita usar fechas como versión principal**, pero se pueden usar como metadata.
4. **Utiliza palabras clave estándar** para versiones especiales, como `lts`, `legacy`.
5. **Define un ciclo de vida claro** para tus versiones, incluyendo cuándo serán obsoletas.
6. **Versiona tu API correctamente**, ya sea en la URL o en los encabezados.
7. **Evita saltos innecesarios en la numeración** y mantén consistencia.

Estas prácticas te ayudarán a gestionar el versionamiento de tu aplicación de manera clara, escalable y fácil de entender para los usuarios y desarrolladores.