Políticas de nomenclatura

MASTER

Tabla de contenido

Justificación: 1

Definición de variables 1

Definición de funciones 3

Definición de Clases 4

Git y GitHub 4

Bifurcaciones 4

Ramas temporales 5

Convenciones de nomenclatura 5

Justificación:

Las convenciones en los procesos de codificación u programación informática son esenciales para el correcto entendimiento, mantenimiento y desarrollo de un código ordenado y de calidad.

Por lo tanto, unas nomenclaturas en el nombramiento de métodos, clases, variables, y sistemas de arquitectura, como: Archivos, módulos, directorios , así como unas reglas de uso establecidas sobre las herramientas que se usan paralelas al desarrollo como los gestores de control de versiones, son un factor muy importante y aunque inconexo a la calidad del proyecto, si repercute significativamente en el entendimiento, ordenamiento y recursividad que proporciona este para desentrañar su comprensión.

Definición de variables

El formato de prevalencia para la definición de variables de la compañía es el camel case, “caja de camello”, consiste en la definición de variables de uso común iniciadas con letra minúscula, para variables con una descripción con más de una palabra se define toda la variable junta y por cada palabra adicional se inicia con la primera letra en mayúscula:

camelCase

Camel case

```
{
    "firstName": "John",
    "lastName": "Smith",
    "email": "john.smith@example.com",
    "createdAt": "2021-02-20T07:20:01",
    "updatedAt": "2021-02-20T07:20:01",
    "deletedAt": null
}
```

Sin embargo, este uso puede variar, si por ejemplo se hace uso de constantes en la realización de un algoritmo, se pude optar por hacer usar el formato Snake case en mayúsculas para indicar que el valor almacenado será constante y no debe cambiar:

snake_case

```
const NON_REPLACE_VALUE = anotherFunction();
```

- **Significativo**: un código legible, como ya hemos dicho, es susceptible de ser entendido por cualquiera que lo lea. Ser conciso no significa escribir palabras cortas o usar identificadores no discursivos. Puede ser explicativo escribiendo las palabras correctas en el contexto correcto. El uso de una convención de nomenclatura que sea demasiado genérica podría ser engañoso para aquellos que leerán nuestro código.

```
const USERS; vs const NUMBER_OF_USERS;

const FRIENDS; vs const NUMBER_OF_FRIENDS;
```

- **Legibilidad**: al declarar una variable o definir un método, es importante elegir un término (o más términos) apropiado para lo que refleja esa

```
String lblFname; vs String label_first_name;
```

entidad en el código. **Dar un nombre confuso no ayuda a que el código se explique por sí mismo**. Al leer una variable o un método, la comprensión de lo que hace ese bloque de código, o lo que es, debe ser **Abreviaturas**: como se mencionó anteriormente, ser firme y conciso en la

- **Abreviaturas**: como se mencionó anteriormente, ser firme y conciso en la redacción del código es importante para que sea compacto y comprensible. Para ello es importante saber abreviar los nombres que damos a las entidades en el código. Si necesitamos asignar un nombre a un botón, llamaremos a la variable *first_btn* en lugar de *first_button* y adoptaremos esta técnica para todos los nombres asignados dentro del código.

Definición de funciones

La definición de funciones/métodos comparte la misma nomenclatura a las variables anteriormente definidas sin embargo aquí siempre prevalece el uso de camel case para su nombramiento.

```
function convertToString(input) {
      if(input) {
           if(typeof input === "string") {
                return input;
            }
           return String(input);
      }
      return '';
}
```

También se debe respetar, no solo el formato de definición, sino que también hay que tener una forma concreta en la semántica con la cual se define un método y que este sea siempre consiste en todo el programa:

```
getName() {…}

retrieveName() {…}

bringBackName() {…}
```

Definición de Clases

Las clases siempre deben ir definidas bajo las nomenclaturas de camel case, con la variación de que la primera letra debe ser siempre en mayúscula, este estándar debe ser siempre respetado para tener una mayor consistencia en el código.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  sayHello() {
    console.log(`Hello, my name is ${this.name}`);
  }
}
```

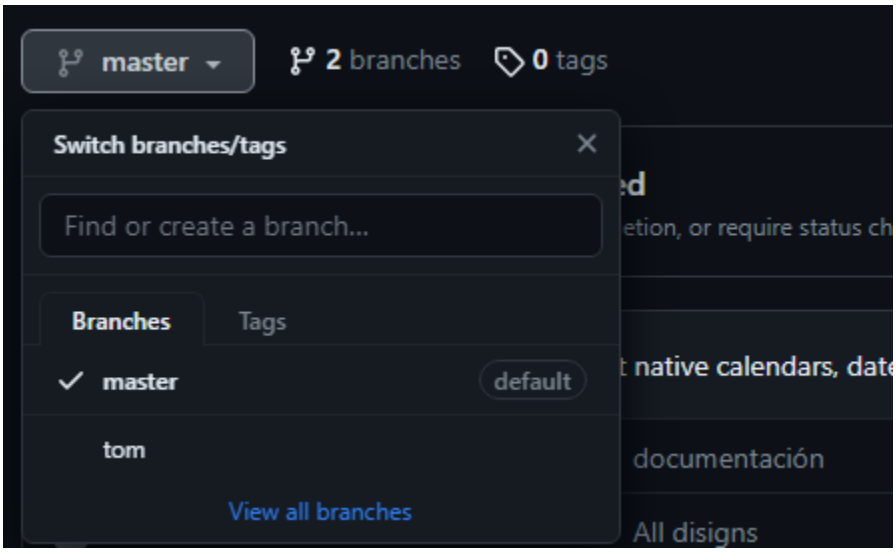
Git y GitHub

Bifurcaciones

Una estrategia de bifurcación es una convención o un conjunto de reglas que especifican cuándo se crean las bifurcaciones.

Ayuda a los equipos y desarrolladores al describir las pautas de nomenclatura de las ramas y explica qué uso deberían tener las ramas, y así sucesivamente. Debido a la falta de convenciones de nomenclatura adecuadas, el equipo de mantenimiento del código sufre numerosas confusiones y complicaciones. La convención de nomenclatura de ramificación de Git admite el crecimiento orgánico de una base de código de manera sistemática. Ayuda a separar el trabajo estratégicamente.

El estándar de nomenclatura predeterminado por la compañía para la bifurcación de ramas es el de rama Maestra; esto significa que la rama maestra en este caso será la rama predeterminada disponible en el repositorio Git, y que los miembros del equipo deberán mantener esta actualizada y estable, lo que significa que no permite el check-in directo la fusión de una bifurcación con la rama maestra sólo será posible después de una revisión del código:



Ramas temporales

Los miembros del equipo pueden crear y eliminar estas ramas cuando sea necesario.

- Arreglo del fallo
- Solución caliente
- Ramas de características

Convenciones de nomenclatura

1. Comenzar el nombre de la sucursal con una palabra de categoría

Uno de los mejores métodos para mejorar la eficiencia es agregar una palabra que categorice la rama. La idea general es usar palabras cortas. La selección de palabras podría ser cualquier cosa que se adapte a su sistema de trabajo.

Use palabras de categoría como:

- WIP: trabajo en progreso y necesita su atención.
- Error: un error que debe corregirse de inmediato.

Con la ayuda de la palabra de categoría, es fácil identificar el propósito de la rama de Git y atenderlo.

2. Uso de identificaciones únicas de seguimiento de problemas en nombres de sucursales

Prefijos como; revisión, función, tarea o cualquier otra variante para categorizar una tarea, aumentar el trabajo que requiere más toma de decisiones al nombrar.

Con identificadores de seguimiento de problemas únicos, básicamente está marcando la categoría de la tarea en el seguimiento y agregando muchos contextos útiles.

Los desarrolladores trabajan principalmente en varios problemas en un momento dado, y un rastreador de problemas ayuda a conectar la rama de trabajo con las tareas relevantes. Hace que el seguimiento del progreso del equipo sea muy fácil.

El uso de una identificación de seguimiento de problemas externos en el nombre de la sucursal puede facilitar el seguimiento del progreso desde sistemas externos.

3**.** Uso de separadores de guiones o barras

La preferencia entre un separador de guiones, barras o guiones bajos se basa en su elección y la de su equipo. La idea es mantenerlo estrechamente consistente. Sin los separadores, los nombres se vuelven más difíciles de leer, creando confusión para el equipo.

Usando separadores como guiones bajos, puede mejorar la legibilidad y hacer que el nombre sea más cómodo de manejar.

4. Usar el nombre del autor en la rama de Git

Numerosas empresas utilizan esta técnica de agregar el nombre del autor a los nombres de las sucursales.

Este método ayuda a rastrear el trabajo de diferentes desarrolladores. Con más requisitos, también son posibles adiciones progresivas.

Ejemplo - *name_feature_new-experimental-changes*

5. Evite usar solo números

Los separadores son especialmente más significativos si se trata de una gran cantidad de sucursales.

Tal confusión durante el proceso de fusión de las ramas de Git puede generar muchos errores.

6. Evite la convención de nomenclatura simultánea

Mezclar o combinar convenciones de nomenclatura de ramas de Git solo genera complicaciones y hace que el proceso sea propenso a errores. La consistencia es un aspecto crítico, y el equipo debe mantener las convenciones decididas.

7. Evita los nombres largos

La precisión es el aspecto crucial de la denominación de ramas de Git. El nombre debe ser informativo y breve. Los nombres largos y detallados solo generan confusión y afectan la eficiencia.

En lugar de: *wip_login_module_which_will_used_in_the_public_website, puede optar por wip_feature_login_module.*

Las prácticas para la asignación de nombres de ramificación de Git son efectivas solo con el compromiso de una aplicación adecuada. El elemento crítico es que el equipo se mantenga en la misma página y sea consistente en todas partes.

Con el uso apropiado de las convenciones, usted y el equipo pueden mejorar notablemente la eficiencia y el flujo del trabajo.