**ESTÁNDAR DE SISTEMA DE CONTROL DE VERSIONES**

**FŌMU**

INTEGRANTES:

HEIDY VIVIANA CASTRO AVENDAÑO

ANA MARIA YELA BUITRAGO

DANIEL GUSTAVO ALZATE URBANO

FABIO EDUARDO PINZON GUERRERO

JAVIER FELIPE FEO CORTÉS

SERVICIO NACIONAL DE APRENDIZAJE

CENTRO DE ELECTRECIDAD ELECTRONICA Y TELECOMUNICACIONES

ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION

BOGOTA

2017

**ÍNDICE**

[SISTEMA DE CONTROL DE VERSIONES 3](#__RefHeading___Toc546_4087272394)

[INSTALACIÓN 4](#__RefHeading___Toc548_4087272394)

[REGISTRO 5](#__RefHeading___Toc550_4087272394)

[NOMBRE DEL REPOSITORIO 6](#__RefHeading___Toc552_4087272394)

[DESCRIPCIÓN DEL PROYECTO 7](#__RefHeading___Toc554_4087272394)

[ARCHIVO “GITIGNORE” 8](#__RefHeading___Toc556_4087272394)

[LICENCIA 10](#__RefHeading___Toc558_4087272394)

[LÉEME 11](#__RefHeading___Toc560_4087272394)

[RAMIFICACIÓN DEL PROYECTO 12](#__RefHeading___Toc562_4087272394)

[ORGANIZACIÓN DE ARCHIVOS Y CARPETAS 13](#__RefHeading___Toc564_4087272394)

[CONFIRMACIONES DE CAMBIOS 14](#__RefHeading___Toc566_4087272394)

[VERSIONES DEL PROYECTO 16](#__RefHeading___Toc568_4087272394)

[BIBLIOGRAFÍA 1](#__RefHeading___Toc518_2988102080)7

SISTEMA DE CONTROL DE VERSIONES

Se debe usar el sistema de control de versiones *Git* para el proyecto, esto se hará con la ayuda de *GitLab* el cual le da la facultad al usuario de realizar repositorios privados, así como el poder usar crear archivos de licencia y archivos *gitignore*, usados para enlistar los tipos y las rutas de archivo los cuales no debe tener en cuenta el repositorio.

Git ofrece un servicio rápido, y permite crear repositorios locales y sincronizarlos con el servidor.

INSTALACIÓN

Para poder hacer uso de Git, es necesario instalarlo en el ordenador.

El proceso de instalación de Git suele ser muy sencillo tanto en plataformas Windows como en plataformas Linux. Para instalar Git en Windows es necesario visitar la página oficial de Git:

[*https://git-scm.com*](https://git-scm.com/)

Desde la página principal se puede ubicar la sección de descargas, después de seleccionar la opción “Downloads for Windows”, el instalador de Git será descargado.

Después de haber sido descargado, se debe ejecutar el instalador de Git. El proceso de instalación suele ser muy sencillo, desde que solo se debe confirmar la instalación en la mayoría de los casos.

Después de haberse instalado se puede utilizar la consola Git Bash, o el entorno gráfico incluído para poder trabajar en Git.

Instalar Git en Linux se puede considerar como un proceso aún más sencillo. Para instalar Git en distribuciones basadas en Debian, es necesario introducir el siguiente comando en la terminal:

*sudo apt-get install git-all*

Para distribuciones basadas en Red Hat:

*sudo yum install git-all*

Para distribuciones basadas en SUSE:

*sudo zypper install git-core*

Para distribuciones basadas en Arch Linux:

*sudo pacman –S git*

Para distribuciones basadas en Gentoo

*sudo emerge --ask dev-vcs/git*

Después de instalado en Linux, Git puede ser utilizado mediante la terminal.

REGISTRO

Después de haberse instalado Git, es indispensable registrarse en GitLab, para poder crear repositorios y sincronizarlos con repositorios locales, a medida en lo que se progresa en el proyecto.

El proceso de registro en GitLab se considera algo sencillo, solo se debe disponer de un correo electrónico. En este caso, se debe hacer uso el correo electrónico proporcionado por el Servicio Nacional de Aprendizaje SENA, también conocido como Correo Misena.

Es indispensable que canda uno de los integrantes del grupo de proyecto, dispongan de una foto de perfil, por lo menos durante el transcurso del desarrollo del proyecto, cada integrante debe incluir sus nombres y apellidos.

NOMBRE DEL REPOSITORIO

En el sistema de control de versiones, el proyecto debe ser nombrado con cada palabra comenzando con letra en mayúscula, todas las palabras deben ir unidas, el uso de caracteres que no pertenezcan al estándar ASCII, por ejemplo, letras con tildes o la letra Ñ, no están permitidos.

DESCRIPCIÓN DEL PROYECTO

GitLab ofrece al usuario la facultad de crear una breve descripción de sus repositorios. En este caso, se debe realizar una descripciòn breve y general acerca del proyecto, para qué entidad se va a realizar, en este caso, el SENA, el cual es una entidad educativa, por lo tanto, se debe incluir el programa de formación y el grupo.

ARCHIVO “GITIGNORE”

El archivo “.gitignore” consiste en un pequeño archivo de texto el cual se usa para añadir excepciones a tipos de archivos, archivos y directorios específicos. Esto para evitar posibles problemas de funcionalidad del repositorio, así como para evitar añadir archivos que no aportan valor al repositorio.

En este caso, se va a manejar un repositorio para una aplicación en Java, por lo cual se deben añadir las siguientes excepciones de archivos:

* Archivos compilados de Java, mejor conocidos como archivos *class*.
* Archivos de log, también conocidos como archivos con extensión *log*.
* Archivos de BlueJ, programa diseñado para el aprendizaje de Java, en este caso se añaden excepciones para los archivos de extensión *ctxt*.
* Archivos de Java 2 Mobile Edition (J2ME), archivos de extensión *mtj.tmp*.
* Archivos comprimidos, o paquetes de archivos:
  + Archivos ejecutables de Java, mejor conocidos como archivos *jar*.
  + Archivos web ejecutables de Java, mejor conocidos como archivos *war*.
  + Archivos ejecutables de Java EE, mejor conocidos como archivos *ear*.
  + Archivos comprimidos en ZIP.
  + Tarballs, archivos de compresión *tar.gz*, *tar.xz*, *tar.bz2*.
  + Archivos comprimidos en RAR.
* Logs de errores de la máquina virtual de Java.
* Directorios de un proyecto hecho en NetBeans IDE:
  + Directorio */build/*
  + Directorio */dist/*
* Directorio */target/*, directorio propio de proyectos construídos con Apache Maven.

Cabe aclarar que para añdir una excepción de extensiones de archivos, solo basta con añadir una línea con el nombre de la extensión del archivo, precedido de un asterisco y un punto:

#*Ejemplo de excepción de tipos de archivos*

*\*.class*

Para añadir una excepción a un directorio, se debe añadir el nombre del directorio entre barras inclinadas, enseguida se añade un asterisco para añadir la excepción a todos los archivos que se encuentren dentro del directorio.

*#Ejemplo de excepción de directorios*

*/directorio/\**

Para añadir una excepción de un archivo, se añade la ruta del archivo.

*#Ejemplo de excepción de un archivo*

*/directorio/ejemplo.txt*

LICENCIA

La licencia que se debe usar para el proyecto es la *Apache License 2.0*, en resumen, esta establece que:

* El proyecto puede ser distribuído bajo la licencia original.
* Cualquier alteración de el contenido original debe ser notificada.
* La documentación respecto a los derechos de autor, marcas registradas, patentes, etc, debe ser conservada.
* No se constituye ninguna garantía respecto a los posibles daños que este pueda causar.

La licencia debe ser creada justo después de haberse creado el repositorio.

LÉEME

El repositorio debe contener un archivo léeme, el cual tenga la descripción del proyecto, incluídos los objetivos general y específicos, así como su descripción del problema y su justificación.

Se debe especificar en qué plataformas es compatible la aplicación y cuáles son los requerimientos para poder usarla.

Entre cada versión del proyecto, se deben incluir los cambios realizados y errores solucionados.

RAMIFICACIÓN DEL PROYECTO

El proyecto se compone de dos ramas principales, las cuales son las ramas *master* y *developer*. La rama *master*, o rama maestra se utiliza para las versiones estables de un proyecto, esta rama solo puede ser modificada cuando se hayan realizado importantes cambios tales como la realización de una nueva versión, el cierre de una fase del ciclo de vida del desarrollo de software, etc.

La rama *developer*, o rama de desarrollo se utiliza para añadir todos los cambios hechos a el proyecto, antes de combinar esta rama con la rama maestra, la rama de desarrollo está disponible para todos los integrantes que tengan los priviliegios de desarrollador, para añadir los nuevos cambios y el nuevo contenido apenas este ya se haya realizado.

En el repositorio, la rama de desarrollo se debe ramificar según el ciclo de vida del desarrollo del software, en tal caso, la rama debe estar dividida en varias fases llamadas “sprints”, caracterìsticas de la metodología de desarrollo Scrum.

Solo puede estar activa una rama de un sprint al mismo tiempo, y esta se debe crear apenas se haya iniciado el sprint, y debe ser cerrado al momento en el que se hayan concluído todas las tareas del sprint.

La rama del primer sprint debe tener el nombre de *Inception*, el cual corresponde al sprint donde se realizan las actividades de análisis y gran parte de las actividades de diseño.

Cada rama de un sprint pueden ser ramificadas al gusto del Scrum Master, aunque preferiblemente se puede ramificar según las historias de usuario o según las tareas asignadas a los desarrolladores. Al momento de haberse concluído la tarea, la rama relacionada con la tarea puede unirse con la rama del sprint.

ORGANIZACIÓN DE ARCHIVOS Y CARPETAS

Las carpetas deben organizar el contenido del proyecto según la categoría la cual defina el grupo de desarrollo. Las carpetas deben ser nombradas comenzando con un número, desde “01”, y enseguida el nombre de la categoría en *Upper camel case* (las palabras deben de estar unidas y cada una debe comenzar con letra mayúscula), la numeración y el nombre de la categoría deben de estar separados con una línea.

Los subtemas de cada carpeta también deberán estar organizados en directorios, ordenados numéricamente, de manera cronológica, en la medida de que el proyecto haya progresado.

Los archivos y carpetas de cada subtema, también deben estar nombrados en *Upper camel case* y deben estar enumerados, pero a diferencia de las carpetas, el nombre y la numeración deben estar separados con una raya al piso.

Cabe aclarar que no se deben usar caracteres que no pertenezcan al estándar de ASCII.

Las excepciones para estas reglas de nombrado de archivos están en los archivos “.gitignore”, la licencia, el léeme, el directorio “.git” y los archivos y carpetas que estén afectados por otras convenciones de desarrollo, como los proyectos de Maven y ANT, las bases de datos o los archivos de código fuente, y los archivos .pdf y .docx, estos dos últimos quedan al criterio del autor.

Conociendo esto, una carpeta principal debe estar nombrada de esta manera:

*01-NombreCarpetaPrincipal*

Y una carpeta o archivo secundario:

*01\_NombreArchivoSecundario*

En este caso, el repositorio manejará los siguientes apartados como directorios principales:

* Levantamiento de Información
* Requerimientos y Planeación
* Diagramación y Modelado
* Informes y Manuales
* Pruebas
* Desarrollo

CONFIRMACIONES DE CAMBIOS

En Git, una confirmación de cambios en un repositorio se le conoce como *commit*, los commits se consideran como el paso final antes de añadir, modificar o eliminar archivos de un repositorio.

Los *commits*, por lo general son incluìdos con un corto mensaje, en el cual se detallan las acciones que se hayan llevado a cabo. Para este caso, se debe manejar una estructura en cada uno de los *commits* que se realicen.

El mensaje de un *commit* debe incluir los siguientes elementos:

* Nivel: indica el área en la cual se han aplicado los cambios, las áreas que pueden presentarse en este elemento son las usadas en el estándar para los *commits* de AngularJS según Parish, s.f.:
  + feat: implementación de nuevo contenido
  + fix: arreglos de bugs
  + docs: documentación
  + style: referente al estilo, reglas sintácticas y apariencia del código
  + refactor: refactorización del código
  + test
  + chore (mantenimiento)
* Asunto: indica brevemente las acciones realizadas, debe iniciar en mayúscula, no debe ser mayor a 60 caracteres, no debe ser escrito en primera persona y no debe terminar con un signo de puntuación. Este debe ser escrito en modo imperativo.
* Cuerpo: indica detalladamente las acciones que se realizaron dentro del *commit*, el largo del texto se toma en cuenta según el criterio del autor del commit, aunque no puede ser igual de corto al asunto. Si se está usando el editor de Vim, es necesario limitar cada línea a 72 caracteres.
* Tarea relacionada: indica la tarea con la que se relaciona el *commit*, en este caso se maneja la numeración de tareas de la página web de organización y seguimiento de proyectos, Taiga.
* Estado de la tarea: indica en una línea especial, el estado de la tarea en Taiga después de haberse realizado el *commit*.
* Ver también: indica tareas o historias de usuario relacionadas con la tarea, se usa la misma numeración de Taiga.

La línea del estado de la tarea determina en que estado la tarea debe permanecer después de haberse realizado el *commit*. Esta línea sigue la siguiente sintaxis:

*TG-\*\*\* #estado*

Siguiendo la línea, se debe escribir el número de la tarea asociada en Taiga. En estado, se debe colocar el *slug* del estado de la tarea. Un *slug* se puede definir como un nombre alternativo para un atributo en Taiga, estos son escritos en Spinal case (las palabras son escritas en minúscula y son separadas con un guión).

Los *slugs* que se deben manejar para los estados de las tareas son los siguientes:

* Nueva tarea: *new*
* Tarea en progreso: *in-progress*
* Tarea lista para testear: *ready-for-test*
* Tarea cerrada: *closed*
* Tarea con necesidad de información: *needs-info*

Tomando en cuenta esto, el mensaje de un *commit* se puede ver de esta manera:

Nivel: style

Asunto: Renombramiento de tablas de la base de datos

Se renombraron todas las tablas de la base de datos siguiendo las convenciones de nombrado de la empresa, las cuales indican que las tablas se deben nombrar en snake case (palabras en minúscula, separadas por una raya al piso), además de que las tablas deben ser nombradas con caracteres del estándar ASCII.

Tarea relacionada: #462

TG-462 #closed

Ver también: #129, #130

VERSIONES DEL PROYECTO

El manejo de las versiones del proyecto, en este caso, se hará mediante el manejo de la nomenclatura de versión semántica, la cual dispone de una estructura de tres números, separados por un punto, cada número está definido por la importancia de los cambios del proyecto. La estructura de esta nomenclatura, según SemVer, s.f., se conforma de:

* Un número para la versión *major*, el cual únicamente se aumenta dependiendo si existen cambios que alteren el paradigma del proyecto, en este caso, este número debe comenzar desde uno.
* Un número para la versión *minor*, el cual aumenta dependiendo de cualquier cambio realizado y el cual no afecte el pararadigma actual de proyecto, es decir, cuando se hayan realizado cambios que sean compatibles con la versión anterior.
* Un número para la versión *patch*, el cual aumenta dependiendo de cualquier cambio que represente algún arreglo de un bug, el cual no cause alteración del paradigma actual.

Tener en cuenta que, si el número *major* es incrementado, los números *minor* y *patch*, deben ser restaurados a cero, al igual de que si el número *minor* es incrementado, el número *patch* debe ser restaurado a cero.

En este proyecto, no se tiene planteado manejar versiones tipo *pre-release*, o versiones *build*.

Conociendo esto, el número con el cual se comienza en el proyecto sería:

1.0.0

En este ejemplo, respectivamente, la versión *major* es 1, la versión *minor* es 0, y la versión *patch* es 0.

BIBLIOGRAFÍA

1. Parish, S. (sin fecha). *AngularJS commit message conventions* (en inglés), GitHub Gist, recuperado de <https://gist.github.com/stephenparish/9941e89d80e2bc58a153>

2. SemVer. (sin fecha). *Versionado Semántico 2.0.0-rc.2,* Semantic Versioning, recuperado de <http://semver.org/lang/es/>