

Proyecto 3 – Got

Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
Algoritmos y Estructuras de Datos II (CE 2103)
Primer Semestre 2020
Valor 15%



Objetivo General

- Diseñar e implementar un programa de consola para facilitar el manejo de código.

Objetivos Específicos

- Investigar y desarrollar una aplicación en el lenguaje de programación C++
- Investigar acerca de programación orientada a objetos en C++.
- Aplicar patrones de diseño en la solución de un problema.
- Aplicar el patrón de arquitectura repositorio.

Descripción del Problema

Got es un manejador de versiones que registra los cambios efectuados en archivos de computadoras permitiendo la colaboración entre personas.

Aplicación de Consola

La aplicación en C++ deberá funcionar mediante comandos de consola. Estos comandos tendrán la siguiente sintaxis:

```
got <comando>
```

Got no es un sistema de control de versión (VCS) distribuido como Git, sino que se comporta mediante un esquema centralizado como SVN. **Todas las funcionalidades solo están disponibles si el server es accesible.**

Comandos

Los comandos pueden tener ciertas variaciones las cuales van a ser explicadas con los comandos.

```
got init <name>:
```

Instancia un nuevo repositorio en el servidor y lo identifica con el nombre indicado por <name>. Además crea cualquier estructura de datos necesaria para llevar el control del lado del cliente sobre cuáles archivos están siendo controlados por el server y cuáles no. Debe crear un archivo llamado .gotignore que permite especificar cuáles archivos son ignorados por el control de versiones. El servidor almacena esta información también.

```
got help:
```

Este comando va a mostrar en la consola la información de lo que hace cada comando en Got.

```
got add [-A] [name]:
```

Permite agregar todos los archivos que no estén registrados o que tengan nuevos cambios al repositorio. Ignora los archivos que estén configurados en .gotignore. El usuario puede indicar cada archivo por agregar, o puede usar el flag -A para agregar todos los archivos relevantes.


Cuando los archivos se agregan, se marcan como pendientes de commit.

```
got commit <mensaje>:
```

Envía los archivos agregados y pendientes de commit al server. **Se debe especificar un mensaje** a la hora de hacer el commit. El server recibe los cambios, y cuando ha terminado de procesar los cambios, retorna un id de commit al cliente generado con MD5.

El server debe verificar que el commit del cliente esté al día con el resto de cambios realizados por otros usuarios. En caso que no sea así, rechaza el commit.

```
got status <file>:
```

Este comando nos va a mostrar cuales archivos han sido cambiados, agregados o eliminados de acuerdo al commit anterior. Si el usuario especifica <file>, muestra el historial de cambios, recuperando el historial de cambios desde el server 

```
got rollback <file> <commit>
```

Permite regresar un archivo en el tiempo a un commit específico. Para esto, se comunica al server y recupera el archivo hasta dicha versión.

```
got reset <file>
```

Deshace cambios locales para un archivo y lo regresa al último commit.

```
got sync <file>
```

Recupera los cambios para un archivo en el server y lo sincroniza con el archivo en el cliente. **Si hay cambios locales, debe permitirle de alguna forma, que el usuario haga merge de los cambios interactivamente**

Repositorio - Servidor

El servidor expone una interfaz a través de un REST API que permite al cliente de Got, enviar los comandos y los datos asociados. El servidor estará desarrollado en el lenguaje y tecnología de preferencia del equipo de trabajo.

El servidor utiliza MySQL para el almacenamiento de los datos. El equipo de trabajo es responsable de diseñar una **base de datos relacional** que se ajuste a las necesidades de Got. Se recomienda investigar la forma en la que SVN u otros manejadores de versiones cliente/servidor almacenan los datos

La primera vez que se hace el commit de un archivo nuevo, el contenido de dicho archivo es guardado en la base de datos. **Commits posteriores del mismo archivo, únicamente guardan la diferencia entre el commit previo y el actual (investigar el comando diff de Linux)** junto con un checksum del archivo. Los datos se guardan comprimidos mediante **Huffman codes**.

El equipo de trabajo deberá decidir muchos de los aspectos técnicos no especificados en este documento

Documentación requerida

1. Internamente, el código se debe documentar utilizando DoxyGen y se debe generar el HTML de la documentación.
2. El archivo Readme del repositorio debe incluir las instrucciones técnicas detalladas para replicar el ambiente de desarrollo y que otros programadores cuenten con la información suficiente para realizar cambios al programa. **Si esta documentación se encuentra incompleta no se realizará la**

revisión del proyecto.

3. Dado que el código se deberá mantener en GitHub o GitLab, la documentación externa se hará en el Wiki de GitHub. El Wiki deberá incluir:
 - a. Breve descripción del problema
 - b. **Planificación y administración del proyecto:** se utilizará la parte de project management de GitHub para la administración de proyecto. Debe incluir:
 - Lista de features e historias de usuario identificados de la especificación
 - Distribución de historias de usuario por criticalidad
 - Plan de iteraciones que agrupen cada bloque de historias de usuario de forma que se vea un desarrollo incremental
 - Descomposición de cada user story en tareas.
 - Asignación de tareas a cada miembro del equipo.
 - c. Diagrama de clases en formato JPEG o PNG.
 - d. Diagrama de arquitectura en formato JPEG o PNG
 - e. Descripción de las estructuras de datos desarrolladas.
 - f. Descripción detallada de los algoritmos desarrollados.
 - g. Problemas encontrados en forma de bugs de *GitHub o GitLab*: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.

Aspectos operativos y evaluación:

1. **Fecha de entrega: De acuerdo con el cronograma del curso**
2. El proyecto tiene un valor de 15% de la nota del curso.
3. El trabajo es **en grupos de 4.**
4. Es obligatorio utilizar un GitHub o GitLab.
5. **Es obligatorio integrar toda la solución, si esto no se realiza se obtiene nota de 0%.**
6. El código tendrá un valor total de 85%, la documentación 15%.
7. De las notas mencionadas en el punto anterior se calculará la Nota Final del Proyecto.
8. Se evaluará que la documentación sea coherente, acorde a la dificultad/tamaño del proyecto y el trabajo realizado, se recomienda que realicen la documentación conforme se implementa el código.
9. La documentación se revisará según el día de entrega en el cronograma.
10. Las citas de revisión oficiales serán determinadas por el profesor durante las lecciones o mediante algún medio electrónico.
11. Los estudiantes pueden seguir trabajando en el código hasta 15 minutos antes de la cita revisión oficial
12. Aun cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones
 - a. Si no se entrega documentación, automáticamente se obtiene una nota de 0 en el proyecto.
 - b. Si no se utiliza un manejador de código se obtiene una nota de 0 en el proyecto.
 - c. Si la documentación no se entrega en la fecha indicada se obtiene una nota de 0 en el proyecto.
 - d. Sí el código no compila se obtendrá una nota de 0 en el proyecto, por lo cual se recomienda realizar la defensa con un código funcional.

- e. El código debe ser desarrollado en las tecnologías y herramientas especificadas, en caso contrario se obtendrá una nota de 0 en el proyecto.
 - f. Si no se siguen las reglas del formato de email se obtendrá una nota de 0 en el proyecto.
 - g. La nota de la documentación debe ser acorde a la completitud del proyecto. Y por lo tanto su nota es proporcional a la completitud el proyecto.
13. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto. El único requerimiento que se consultará durante la defensa del proyecto es el diagrama de clases, documentación interna y la documentación en el manejador de código.
 14. Cada estudiante tendrá como máximo 15 minutos para exponer su trabajo al profesor y realizar la defensa de éste, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo cual se recomienda tener todo listo antes de ingresar a la defensa.
 15. Cada excepción o error que salga durante la ejecución del proyecto y que se considere debió haber sido contemplada durante el desarrollo del proyecto, se castigará con 2 puntos de la nota final del proyecto.
 16. Durante la revisión únicamente podrán participar el estudiante, asistentes, otros profesores y el coordinador del área.

ANEXO DEL PROYECTO

Objetivo General

- Elaborar un documento que permita el establecimiento y la ejecución de los lineamientos para el trabajo en equipo.

Objetivos Específicos

- Establecer los lineamientos adecuados para el trabajo en equipo y la descripción de los roles y responsabilidades de cada uno de los miembros del equipo.
- Describir la ejecución de los lineamientos establecidos para el trabajo en equipo y las responsabilidades de cada uno de los miembros del equipo.

Atributos de Acreditación

- Trabajo individual y en equipo (Medio).

Descripción del Entregable

Cada grupo debe elaborar un documento que tenga la siguiente estructura:

1. Portada.
2. Tabla de contenidos.
3. Introducción.
4. Descripción de los lineamientos establecidos para el trabajo en equipo y los roles y responsabilidades de cada miembro del equipo.
5. Plan de iteraciones que describa la distribución de las historias de usuario en las diferentes iteraciones y que a su vez describa la carga de cada miembro del equipo durante las diferentes iteraciones.
6. Bitácora con el trabajo realizado durante el proyecto por cada miembro del equipo.

Aspectos operativos y evaluación

1. El documento debe ser enviado en formato PDF.