UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

GIT

Laboratorio 1

Sebastian Villalobos
Profesor: Roberto Gonzalez
Ayudante: Nicolas Alarcón

Santiago – Chile
1er semestre – 2020

Índice

Portada 01

Índice 02

Introducción 03

Contexto inicial del problema 04

Propuesta Solución 05

Lógicas de App 06

Marco técnico 07

Conclusiones 12

Introducción

El presente informe tiene por objeto registrar el trabajo realizado para el taller de la clase Paradigmas de programación.

El desarrollo de este proyecto ha sido usando el lenguaje de programación Scheme como respuesta al requerimiento inicial del profesor, además de considerar el paradigma funcional, revisado amplia y detalladamente durante las cátedras. Asimismo, la abstracción de la problemática fue formada con lo que denominamos TDAs, que son representaciones que nos sirven para establecer “cualquier cosa” como una unidad manejable por el lenguaje de programación en el marco del paradigma funcional.

El fin de todo esto, es lograr un programa que sirva, en términos generales, es representar una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git.

Contexto inicial del problema

La problemática es la implementación correcta usando el paradigma de programación “funcional”, y su lenguaje de programación “scheme”, con el que se debe representar de la mejor manera la app “GIT”

Propuesta técnica

Se realiza una solución con drRacket que representa el funcionamiento de la app, dando control a versiones de archivos, para esto se realizaron dos TDAs, que permitieron el desarrollo completo del laboratorio, estos fueron TDA Commit y TDA Zonas.

Lógicas de la app

La APP “GIT Scheme”, permite al o los usuarios trabajar colaborativamente en el desarrollo de archivos, manteniendo un total control de los cambios realizados y acceso a estos, la unidad básica que tiene toda la información necesaria para el trabajo son los “commits”, estos almacenan, el nombre del autor de los cambios realizados, el nombre del archivo en el que se realizaron cambios, la fecha en que estos fueron hechos, una descripción general o detallada de los cambios y los cambios realizados.

La aplicación cuenta con 4 zonas de trabajo, que son conformadas por la información de los commits, y estas zonas son manejadas por el usuario para obtener y compartir cambios en los archivos, la primera de ellas es el “Workspace” un entorno de trabajo local donde se trabaja en el o los archivos almacenados en el equipo o descargando las versiones actualizadas desde la nube, la siguiente es el “Index” donde se almacenan los cambios en los archivos realizados en el workspace, siguiendo a este espacio está el “Local Repository” donde los commits son guardados con un mensaje descriptivo, luego está el “Remote repository”, en esta zona se encuentran los commits en con la información compartida para todos los usuarios de estas zonas de trabajo, en forma paralela dentro de las zonas de trabajo existe un espacio que almacena los comandos aplicados por la APP (Logs).

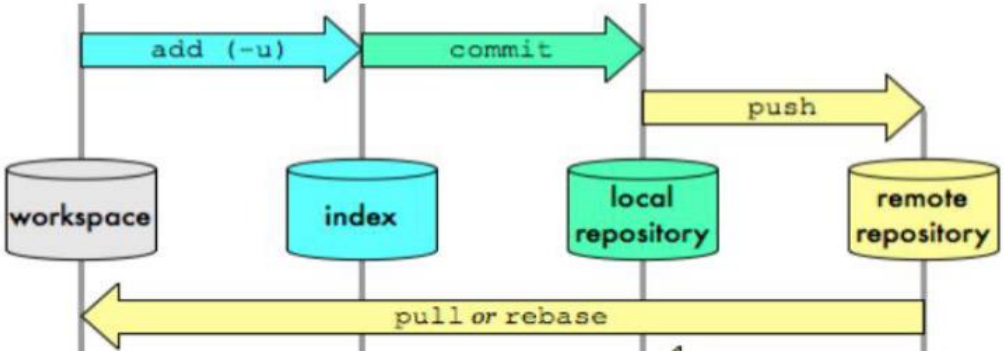
Para trabajar con la app se deben usar los siguientes comandos.

Pull: Trae todos los cambios (commits) desde el Remote Repository al Local Repository.

Add: Añade los cambios locales del Workspace al Index.

Commit: Genera un commit con los cambios almacenados en el Index.

Push: Envía los commits del Local Repository al Remote Repository.



Marco Técnico

El programa considera una serie de abstracciones para su ejecución basado en la terminología conocida como TDA, a fin de encapsular las partes esenciales de la aplicación y poder ser codificadas mediante la siguiente estructura técnica definida:

Representación – Constructor – Pertenencia - Selectores –
Modificadores - Funciones adicionales

TDA: commit

Un commit representado por una lista conformada por 5 elementos, nombre del archivo modificado X autor del commit X fecha de commit X descripción general de los cambios X cambios realizados:
'(Archivo, Autor, marca de tiempo, mensaje descriptivo, cambios almacenados)

- Archivo = String con el nombre del archivo con el formato "nombreArchivo.extencion".
- Autor = String Con el nombre del Autor con el formato "Nombre Apellido".
- Fecha = Int con el formato AñoMesDiaHora Ejemplo: 18:29 3 de abril 2020 será representado por 202003031829.
- Descripción = String con la descripción de los cambios.
- Cambios = String con los cambios realizados.

TDA: Zonas

Zonas está representado por una lista con 5 elementos, los 4 primeros elementos representarán una zona de trabajo, cada una de estas zonas será una lista de commits, el último elemento de zonas, almacena los logs de comandos aplicados por el usuario.
'('(workspace) '(index) '(local) '(remote) '(logs))

- Workspace: está representado por una lista de commits
- Index: está representado por una lista de commits
- Local Repository: está representado por una lista de commits
- Remote Repository: está representado por una lista de commits
- Logs está representado por una lista de strings con el comando y el tiempo en el que se usaron.

Requerimientos Funcionales Obligatorio

- **TDAs**: abstracciones apropiadas para el problema.
- **GIT**: Función que permite aplicar los comandos sobre las zonas de trabajo. La función retorna una función que recibe los parámetros propios del comando que se pretende aplicar, procurando dejar un registro de histórico de los comandos expresados como funciones aplicados en el orden en que fueron aplicados.

  Dominio: Función de git x marca de tiempo

  Recorrido: Funcion, registro de logs
- **Pull**: Función que retorna una lista con todos los cambios (commits) desde el RemoteRepository al Workspace registrados en las zonas de trabajo. Los cambios reflejados en el retorno de la función corresponden a una nueva versión de zonas donde se vean reflejados los cambios.

  Dominio: zonas x marca de tiempo

  Recorrido: nuevas zonas
- **Add**: Función que añade los cambios locales registrados en el workspace al index en las zonas de trabajo, especificando la lista de nombre de archivos concretos (nombres como string).

  Dominio: nombres de archivos x zonas x marca de tiempo

  Recorrido: nuevas zonas

  Commit: Función que genera un commit con los cambios almacenados en el index, especificando un mensaje descriptivo "string" para llevarlos al Local Repository.

  Dominio: mensaje x zonas x marca de tiempo

  Recorrido: nuevas zonas
- **Push**: Función que envía los commits desde el repositorio local al repositorio remoto registrado en las zonas de trabajo, los cambios

reflejados en el retorno de la función corresponden a una nueva versión de zonas.

Dominio: zonas

Recorrido: nuevas zonas

- **Log:** Función que muestra los últimos 5 comandos utilizados sobre las zona de trabajo y su hora de ejecución.

  Dominio: zonas

  Recorrido: últimos 5 comandos

Conclusiones

Personalmente la realización de este trabajo utilizando el paradigma funcional y el lenguaje de scheme fue un verdadero desafío, la representación a través de funciones me ayudó al desarrollo de tareas cotidianas en mi trabajo, ya sea creando scripts de automatización como en la configuración de algunos servicios de monitoreo en REDES, el único inconveniente que tuve fue el tiempo, una mala planificación
Me pasó la cuenta.