#### Universidad Autónoma del Sur “UNASUR”

****

# Ingeniería en Informática

# 2do. Año

## GIT

##### Nombre del profesor: Diego Francisco Correa

Integrantes:

* **Sergio Chamorro (sergiochamo96@gmail.com)**
* **José Paredes (joseparedespy@icloud.com)**
* **Julio Denis (moopiojulio@gmail.com)**
* **Iván Franco (octavioivan@live.com)**

**Asunción – Paraguay**

**Año 2016.**

Tabla de contenidos

1. *INTRODUCCION 3*
2. *OJETIVO 4*
   1. Objetivo general 4
   2. Objetivos específicos 4
3. *CONTENIDO 5*
   1. Capitulo - I 5
   2. Capitulo - II 5
   3. Capitulo – (n) 5
4. *ANEXOS 6*
5. *CONCLUSION 7*
6. *BIBLIOGRAFIA O WEBLIOGRAFIA 7*
7. **INTRODUCCIÓN**

Cuando realizamos algún proyecto de cualquier tipo siempre trabajamos sobre el mismo documento y lo vamos mejorando con el tiempo, pero hay veces que queremos modificar o probar algo y creamos otra versión del documento en otra carpeta, y así sucesivamente hasta que tenemos varias carpetas con distintos cambios o versiones del mismo proyecto, que en algún momento se convierte en un lio de carpetas.

Git es un sistema de control de versiones que se encarga de guardar los cambios realizados en un proyecto, separándolos por estados o versiones del archivo y el estado actual del mismo.

Nos da la posibilidad de poder recorrer esas versiones y tener acceso a cada cambio que se hizo en el proyecto, ya sea para encontrar errores o volver a la versión inicial, entre otras.

1. **OBJETIVOS** 
   1. **Objetivo general**

* Aprender a utilizar el control de versiones Git para crear repositorios donde tengamos registrados los cambios de un archivo o conjunto de archivos.
  1. **Objetivos específicos**
* Comprender los conceptos básicos de Git y las ventajas que nos provee.
* Conocer los comandos necesarios para la utilización de Git.
* Alojar un proyecto en GitHub para repositorios remotos.

1. **CONTENIDO**

**Control de Versiones**

Un control de versiones es un sistema que se utiliza para registrar los cambios que hacemos en un documento o proyecto a medida que pasa el tiempo. Nos da la posibilidad de poder volver a cualquier versión específica de nuestro proyecto, poder ver los cambios realizados en esas versiones, comparar los cambios que hicimos durante el proceso de desarrollo, ver quien modificó y en qué momento, entre otras cosas más.

Existen 3 tipos de sistemas de control de versiones, las cuales son:

* Sistema de control de versiones locales.
* Sistema de control de versiones centralizado.
* Sistema de control de versiones distribuido.

**Ramas y Fusiones**

Una de las ventajas de Git es el trabajo con ramas, se pueden crear ramas de forma muy fácil y es cómodo trabajar con ellas. Es muy importante tener muy claro que es una rama, para que sirve, así para explotar todo el potencial que proporcionan.

Otro concepto relacionado con las ramas son las fusiones, que nos permite unir el trabajo desarrollado en diferentes ramas.

Antes de explicar lo que son las ramas y fusiones debemos tener en cuenta el concepto de head. ¿Cómo sabe git en que rama estas en algún momento dado?, mediante un apuntador especial denominado head.

El concepto de head simplemente es el commit donde nosotros nos encontramos ahora, es decir si nosotros hacemos un **$ git checkout** al primer commit, segundo o cualquier commit ese va ser nuestro head.

Si vamos a nuestro proyecto y ponemos en la consola:

$ git log

Vemos todos los commit que hicimos en el proyecto, el ultimo commit es nuestro head. Un comando que nos indicará más específicamente cuál es nuestro actual head es:

$ git log --pretty --decorate

Este comando nos listará todos los commit, con sus respectivos autores, fecha de creación, además nos indica el head actual.

**¿Qué es una rama?**

Es una línea de tiempo en nuestro proyecto, que nos sirven para arreglar errores, experimentar, para hacer cambios de interfaz, cambios de lógica... cualquier cambio que no afecte a nuestro proyecto actual en sí.

La rama por defecto de Git es **master**. Con la primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación.

En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.

Si queremos ver las ramas con las que contamos en nuestro proyecto colocamos en la consola:

$ git branch

Ese comando nos mostrará todas las ramas de nuestro proyecto, nos marca con un asterisco [**\***] y nos colorea la rama en la que nos encontramos en este momento.

Por ejemplo:

$ git branch

**\* master**

**Crear una nueva rama**

Supongamos que estamos trabajando en un proyecto y queremos añadir una nueva funcionalidad al mismo. La forma adecuada de hacerlo con Git es crear una nueva rama con el nombre de la funcionalidad donde añadiremos nuestros cambios.

Comando para crear una nueva rama:

$ git branch Desarrollo

Una vez creada nuestra rama, en este ejemplo **Desarrollo**, podemos comprobar en que rama nos encontramos utilizando el comando **$ git branch.**

$ git branch

Desarrollo

**\* master**

Vemos que seguimos teniendo master coloreado con asterisco, eso es porque donde estamos ahora, pero ya tenemos la rama Desarrollo que habíamos creado. ¿Entonces cómo nos movemos a Desarrollo?

Para eso usamos el siguiente comando:

$ git checkout Desarrollo

Existe un atajo para crear y movernos a una rama que aún no existe usando directamente el comando checkout, que consiste en pasar el parámetro **-b** en la llamada.

$ git checkout –b Desarrollo

Si volvemos a colocar en la consola **$ git branch**, vemos que ahora la rama Desarrollo es la que tiene el asterisco y esta coloreada, eso quiere decir que es la rama en la que nos encontramos ahora.

$ git branch

**\* Desarrollo**

master

En ejemplo anterior podemos ver como nos encontramos en la rama Desarrollo. Esto significa que, si modificamos, borramos o añadimos archivos, ésta rama Desarrollo incluirá esos cambios, pero no así la rama master.

**FUSIONES**

Cuando se trabaja con ramas en Git, en algún momento el trabajo realizado en una rama será fusionado con otra rama. Por ejemplo, hemos testeado profundamente nuestros cambios y estamos listos para moverla de la rama Desarrollo a la rama principal master. Dicha acción requiere que mezclemos el contenido de una de las ramas dentro de la otra.

Para realizar la mezcla de la rama Desarrollo dentro de master vamos a situarnos en la rama master y a mezclar ambas con el siguiente comando:

$ git checkout master

$ git merge Desarrollo

Visualizamos los commit de master y vemos que los commit de Desarrollo ya están en la rama, absorbimos los cambios de la rama Desarrollo.

1. **ANEXOS**

Cualquier información adicional ya sea de carácter informativo o gráfico.

1. **CONCLUSION**

Resumen de los aspectos más importantes aprendidos durante el desarrollo del trabajo.

1. **BIBLIOGRAFIA O WEBLIOGRAFIA**

Se describirá en forma detallada los libros o direcciones completas URL, desde donde fueron extraídos los datos o informaciones utilizados en el desarrollo del trabajo.