UNIVERSIDAD CATÓLICA DE SANTA MARÍA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO 01:

GIT & GITHUB

I

OBJETIVOS

- Conocer los fundamentos de la herramienta de control de versiones Git y Github.
- Crear y gestionar repositorios locales.
- Vincular repositorios locales a remotos
- Evidenciar el manejo de versiones en los repositorios.
- Evidenciar el manejo de diferentes ramas

II

TEMAS A TRATAR

- Git
- Conceptos y nomenclatura
- Flujo de trabajo de Git
- Comandos básicos de Git
- GitHub
- Creación de repositorios
- Vincular un repositorio local con un remoto
- Clonar repositorio

III

MARCO TEÓRICO

¿Qué es Git?

Git es un sistema de control de versiones (VCS) lanzado en el año 2005 por el grupo de desarrollo del kernel de Linux, liderado por Linus Torvalds.

Es un software de control de versiones, su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos. Existe la posibilidad de trabajar de forma remota y una opción es GitHub.

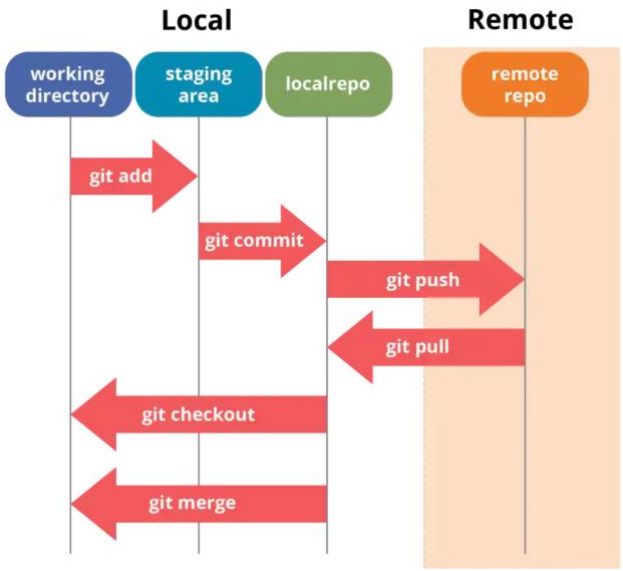
a) Conceptos/nomenclatura

- **repositorio**: Es el espacio donde se almacenan todos los ficheros del proyecto, con un historial de todos los cambios que han ido sufriendo a lo largo del tiempo.
    - **local**: Es un repositorio completo que tiene cada usuario en su máquina.

- **remoto**: Es una referencia a cualquier otro repositorio que no sea el local (por ejemplo Github). Para colaborar con otros usuarios se usan repositorios remotos, que van sincronizando y mezclando (merge) con el contenido de sus repositorios locales.

- **commit**: Conjunto de cambios (changeset) sobre uno o varios ficheros del proyecto, que lleva asociado un breve mensaje descriptivo y un autor.
- **rama**: Una rama es una bifurcación en la línea de desarrollo del proyecto, que tendrá su propio historial de commit y estará separada por completo de la rama por defecto (master). Se usan para implementar nuevas funcionalidades sin afectar a la versión estable del proyecto.
  - **rama master**: Es la rama creada por defecto.
- **HEAD**: Es un puntero, que apunta al último commit de la rama en la que te encuentres trabajando actualmente, salvo que hayamos movido HEAD a propósito a un momento en el pasado.

b) Flujo de trabajo de Git



~ **Working dir/Área de trabajo**: Es el estado actual de nuestros ficheros, aquellos en los que programamos y que podemos visualizar en nuestro navegador de archivos.

~ **Staging Area/Index/Área de ensayo**: Es una zona intermedia donde vamos almacenando los cambios que van a conformar un commit. Pueden ser ficheros completos o solo porciones concretas. Para "pasar" ficheros a esta zona usamos el comando **add**. Necesariamente hay que enviar los cambios al staging area antes de poder realizar un commit.

~ **El repositorio local**: Es donde se almacenan los commits mediante el comando **commit**, una vez hemos seleccionado los cambios en el index. Toda su información se guarda en el directorio *.git.*

~ **Los repositorios remotos**: Puede ser uno solo (por defecto se llama origin), o podríamos tener varios. La comunicación entre el repositorio local y los remotos se realiza mediante los comandos **push** (enviar) y **pull**(descargar).

~ ***Stash**: Es la zona de guardado rápido, una zona auxiliar para guardar los cambios en los que estamos trabajando cuando por algún motivo nos interrumpen y tenemos que cambiar de rama, pero aún no queremos hacer un commit porque es un commit a medias, sin acabar. Puede almacenar *n* estados y funciona como una pila, colocando siempre en el primero los últimos cambios que salvemos.

No es posible hacer un push a un remoto de un commit que se encuentre en la zona de stash. Por lo que esta zona es únicamente de uso privado.

c) Comandos básicos

- **Creación y configuración**

`git version`
muestra la versión de git

`git config –global user.name “mi nombre”`
Establece el nombre de usuario que va a figurar como autor del commit, que podremos ver en los logs.

`git config --global user.email correo@electronico`
Junto con el nombre aparece en el campo author de cada commit.

`git init`
Crea un repositorio local vacío en el directorio actual.

`git clone <url>`
Descarga una local copia de la rama master de un repositorio remoto. De este modo ya podría hacer pull sin tener que añadirlo manualmente.

`git fetch [repositorio remoto]`
Descarga los cambios (por ejemplo ramas nuevas) que se hayan producido en el repositorio remoto. Si se omite el repositorio, por defecto usa origin.

`git fetch --prune:`
Descarga los cambios y además borra las ramas que ya no existen en el remoto.

`git status:`
Este es el más usado de todos. Nos dice la rama en la que nos encontramos y los ficheros que contiene el stage junto a su estado. Además avisa de ficheros sin seguimiento por git.

- **Flujo entre stash/working/stage/repo**

`git add <lista de ficheros>`
Manda uno o varios ficheros separados por espacios o un directorio al *staging area*. Recién añadido un nuevo fichero al repositorio, este se encuentra en estado "untracked" y debemos hacer un primer *add*. Podríamos usar el carácter "." si queremos enviar todos los ficheros del directorio de trabajo que tienen algún cambio. También admite comodines, ej: git add *.js

`git commit -m "descripción breve"`
Registra definitivamente los cambios que previamente estaban en el *Stage*. Un commit contiene:
  ~ id: Identifica el cambio de forma única. Es el hash SHA-1 del resto de campos.
  ~ mensaje: que habitualmente se trunca a 72 caracteres al visualizarlo en el log o github.
  ~ autor: El autor del cambio identificado por su name y su email.
  ~ id del árbol o snapshot en ese momento.
  ~ id del anterior commit.

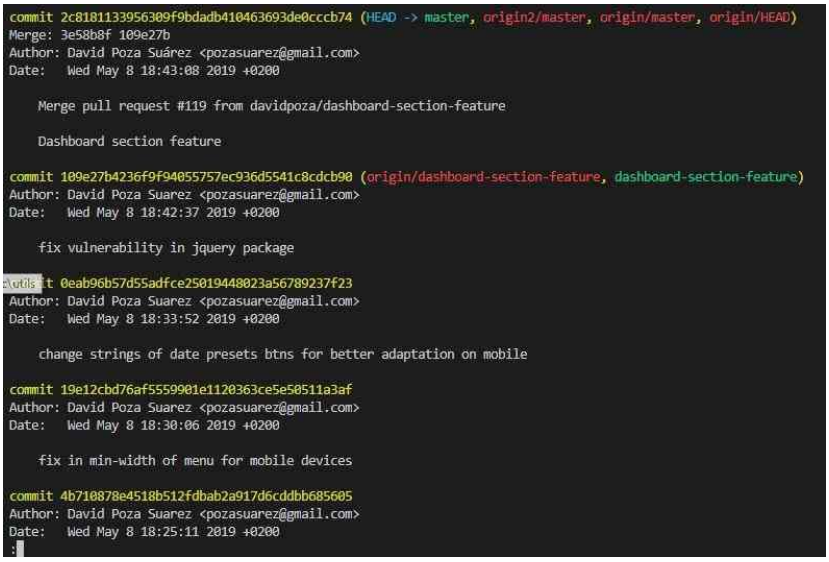
`git commit -am "descripción"`

Hace un commit con todos los ficheros con cambios. Por tanto esta opción a es para realizar el add previo y el commit de una sola vez.
*Importante: solo funciona si el fichero ya está en seguimiento por git (tracked).

- **Histórico de cambios**

Log

Otro de los comandos más usados es **git log**. Abre un listado que nos va a permitir ver el histórico completo de todos los commit que se han ido realizando en el repositorio a partir de HEAD hacia atrás, incluyendo los que nos traemos del remoto cuando hacemos un **fetch**. Funciona en modo interactivo y por defecto los últimos cambios aparecen los primeros.



Podemos desplazarnos con las flechas arriba/abajo o bien usando la tecla space para saltar de pagina en pagina. También podemos buscar dentro de cualquier contenido del commit usando la barra: /palabra y enter. Para salir del modo interactivo, como es habitual en linux, hay que pulsar la tecla q y luego enter.

Opciones de log

`git log <commit-id|rama|HEAD>`

Muestra el log a partir del commit, rama o puntero HEAD.

`git log -n`

Muestra los últimos n commits.

`git log -- <fichero>`

Muestra todos los commits en los que se ha modificado el fichero indicado.

- **Viajando en el tiempo**

`git checkout <commit_id|rama|HEAD>`

Podemos usar checkout para visualizar el estado del proyecto en un commit concreto, es decir, en ese momento en el tiempo. Con esta operación lo que internamente estamos haciendo es apuntar a ese commit con el puntero HEAD.

- **Inspeccionar un commit:**

  `git show <commit_id>`
  Nos permite visualizar los cambios en el código que incluye un commit concreto.

- **Encontrar diferencias:**

  `git diff`
  Permite ver las diferencias en el código entre el working directory y el Index.

  `git diff -Un`
  Por defecto diff usa un context de 3 líneas, sin embargo, se puede ampliar ese contexto a n líneas.

  `git diff <commit1_id> <commit2_id> -- [fichero]`
  Permite ver las diferencias en el código de un fichero concreto (si no especificamos entonces compara todos) y su estado en el commit1 y el commit2.

  `git diff --stat rama1..rama2:`
  Para visualizar un listado de ficheros que han cambiado entre ramas.

- **Trabajar con ramas**

  `git branch`
  Nos muestra la rama activa.

  `git branch <nuevarama>`
  Crea una nueva rama a partir del commit apuntado por HEAD.

  `git branch -d <rama>`
  Borra la rama.

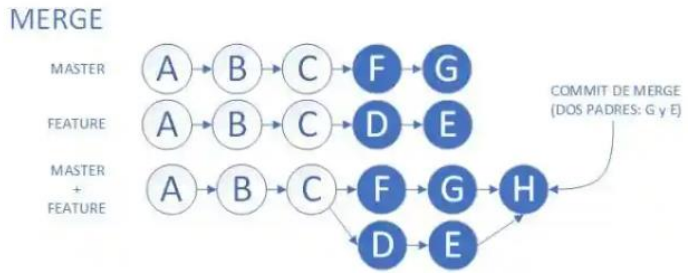
  `git branch –list`
  Listado de las ramas locales.

  `git branch -a`
  Lista de ramas tanto locales como remotas.

  `git checkout <rama>`
  Cambia la rama activa a la indicada.

  `git checkout -b <rama>`
  Hace dos pasos en uno: crea la rama a partir de la activa y se cambia a ella.

  `git merge <rama>`
  Se fusiona la rama indicada con la rama activa en ese momento, creando un commit de merge, que tiene dos commit padres, que podremos ver en el log.



- **Repositorios remotos**

  `git push <repo-remoto> <rama>`
  Envía todos los commits de la rama indicada que no existan en el repositorio remoto.

`git pull <repo>`

Es el equivalente a realizar un fetch + merge. Si no indicamos repositorio, por defecto usa el repositorio *origin*.

¿Qué es Github?

Github es la plataforma online para trabajar con proyectos en git, en simples palabras es utilizar git de forma remota pero entiendase que son cosas totamente distintas

IV

ACTIVIDADES

Instalación de Git

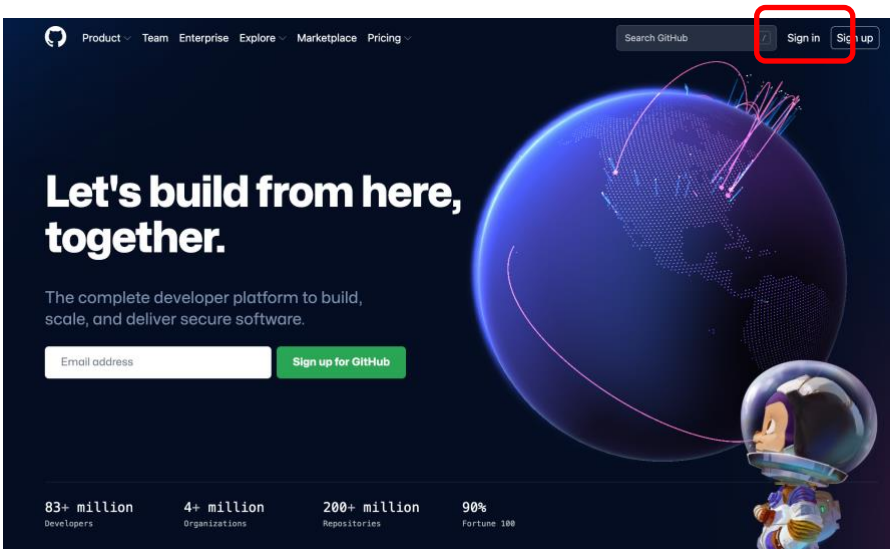
1. Para instalar Git, teniendo en cuenta el sistema operativo que utiliza, descargue los instaladores de la siguiente página, y procede a realizar la instalación según su profesor lo indique: https://git-scm.com/

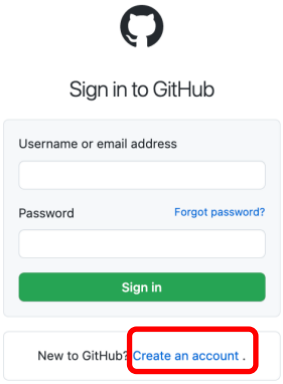
(Como ayuda puede usted ver el video que se encuentra en el aula virtual donde le muestra el proceso de instalación – del minuto 9.30 al 13.33)

2. Una vez instalado, para verificar si ya tenemos Git vamos a ver la versión:
   `git version`

Crear una cuenta en Github

Para comenzar a trabajar con github tienes que crear una nueva cuenta (en caso aún no la tenga) ingresando a la siguiente url: https://github.com/

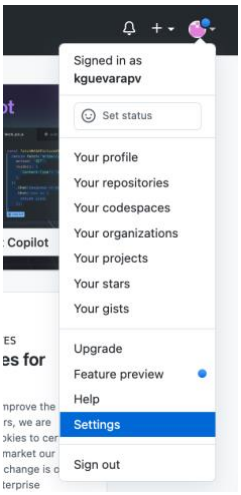


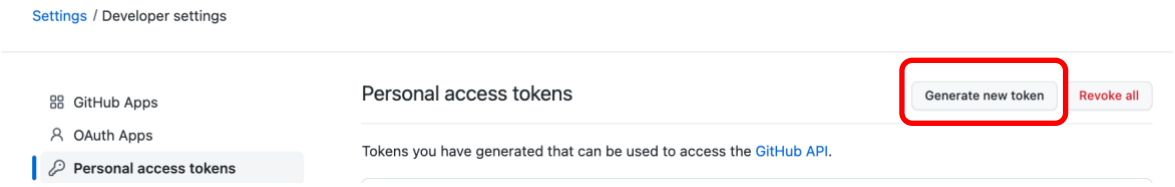


Nota: de preferencia utilice el correo institucional para crear su cuenta en github.

Configurar token en Github

Debido a las nuevas políticas e seguridad impllementadas en Github es necesario que además de su password, active su token de seguridad. Para esto, una vez que haya creado su cuenta, acceda a la opcion **Settings/Developer settings/Personal access tokens** y genere un buevo token





Crear un repositorio en Git

3. Ubíquese en algun directorio que le indique su profesor en donde creará el directorio LP3_lab01. Esto puede hacerlo a través del explorador de windows o escribiendo la orden correspondiente en el CMD.
   Para hacerlo a travás de comandos, abra una ventana de CMD o el shell de git y úbiquese en el directorio antes mencionado con el comando
   `cd ruta` ruta es el path del directorio a donde quiere moverse.

   Para crear el directorio:
   `mkdir LP3_lab01`

4. Úbiquese en este nuevo directorio con
   `cd LP3_lab01`

   Nota: puede utilizar el tabulador para definir la ruta del directorio

5. Estando en el directorio, vamos a crear el repositorio local escribiendo:
   `git init`

Hacer seguimiento a nuestros proyectos

Para demostrar como realizar el seguimiento al código de nuestros proyectos, haremos un ejemplo muy simple.

6. Abra Eclipse o cualquier otro IDE en donde escribiremos un programa simple en Java. Cree un proyecto nuevo con el nombre PyLab01. Y en el proyecto nuevo cree la clase Principal la misma que tenga el método **main()**

7. En la clase Principal escriba lo siguiente. Luego ejecutelo y compruebe los resultados.

```java
public class Principal {

    public static void main(String[] args) {
        int vida, rp;
        boolean fl = false;
        vida = 0;
        Scanner sc = new Scanner(System.in);
        do {
            vida ++;
            System.out.println("Vida:"+ vida + "\t¿Cual es el número secreto? ");
            rp = sc.nextInt();
            if (rp == 1234)
                fl = true;
        }while(vida < 3 && !fl );
        if (fl == true)
            System.out.println("Adivinaste!!!!! ");
        else
            System.out.println("ups, perdiste...");
    }

}
```

8. Una vez comprobados los resultados, copie los archivos .java (fuentes) a la carpeta LP3_lab01 que ha creado en el punto 3. Esto lo puede hacer desde el explorador de windows o utilizando el comando **cp** en la ventana de **cmd**.

9. Vamos a ver el estado de los archivos que han sido incluidos en el repositorio local.

git status
observe como el archivo se ve en color rojo.

10. Suba al staging area los archivos que están aún sin seguimiento.
    git add .

    git status
    observe como el archivo se ve ahora en color verde.

11. Vamos a actualizar el repositorio local con los archivos que se encuentra en el staging area.
    git commit -m “Clase Principal con ….. “

    git status
    observe como ahora nos indica que no hay nada pendiente de seguimiento

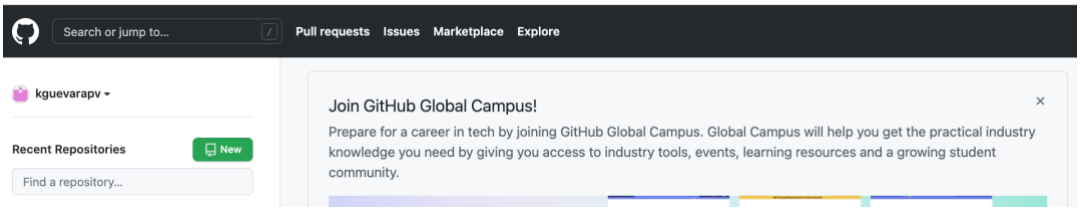
Nosotros podemos seguir trabajando en el repositorio local (máquina local) e ir actualizando las versiones de nuestro código en este repositorio sin necesidad de subir al repositorio remoto aún.

Actualizar el repositorio remoto

Todos las actualizaciones hechas en el repositorio local pueden ser registradas en el remoto en cualquier momento. Para esto, en caso aún no tengamos el repositorio remoto creado, primero debemos crearlo.

Crear un repositorio en Github

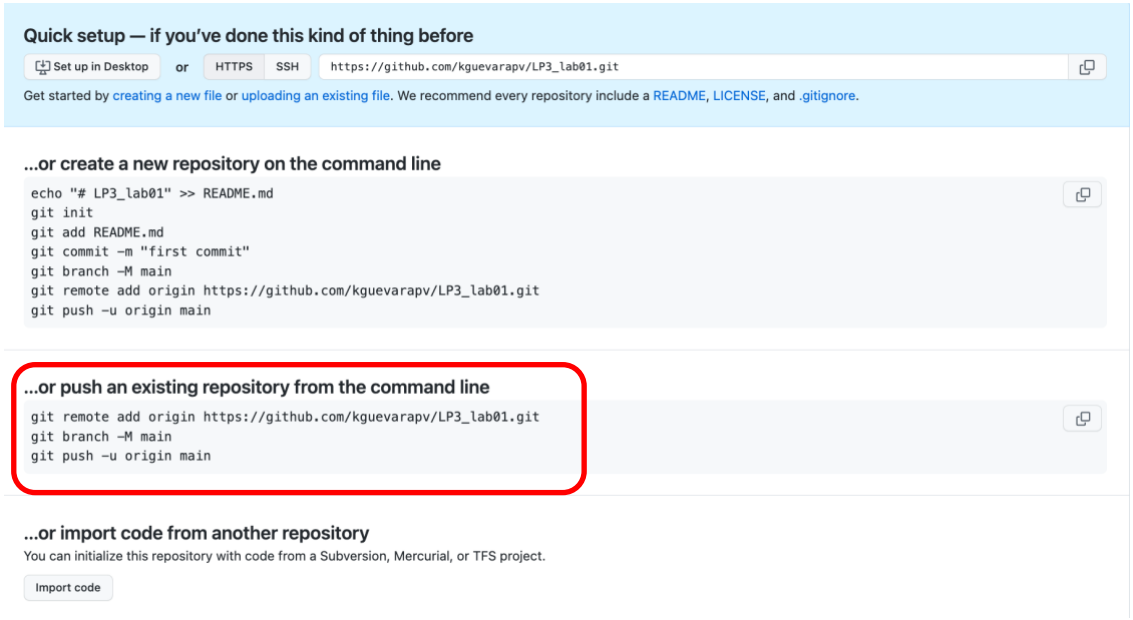
12. Una vez que haya iniciado sesión en Github, vamos a crear un repositorio llamado LP3_lab01. Lo va a crear sin el archivo readme.



Nota: el nombre del repositorio remoto no necesariamente debe ser el mismo que el repositorio local, sin embargo, se recomienda que sea el mismo para mantener un orden entre los proyectos que se vayan a desarrollar.

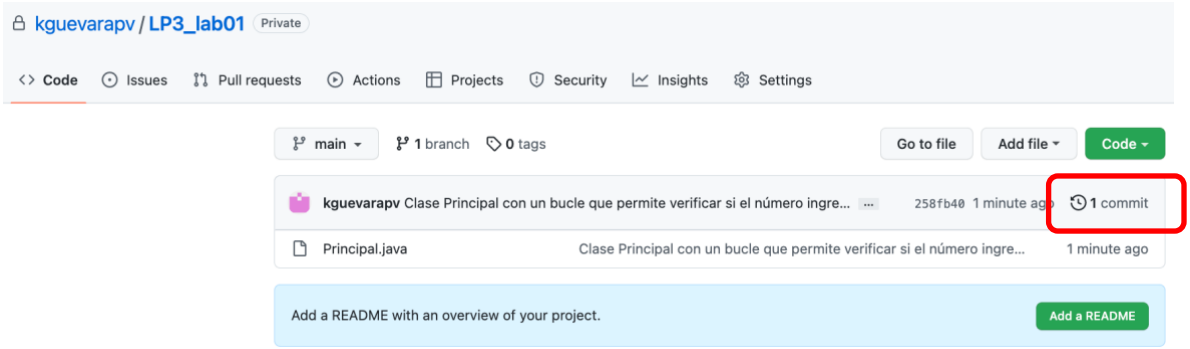
Vincular un repositorio local con el remoto y realizar un primer registro

13. Al momento de la creación el repositorio nos mostrará una serie de comandos para subir el proyecto. Debe de copiar y ejecutar en el **cmd** las instrucciones que correspondan al caso.



Nota: es posible que al realizar esta acción nos solicite las credenciales de Github

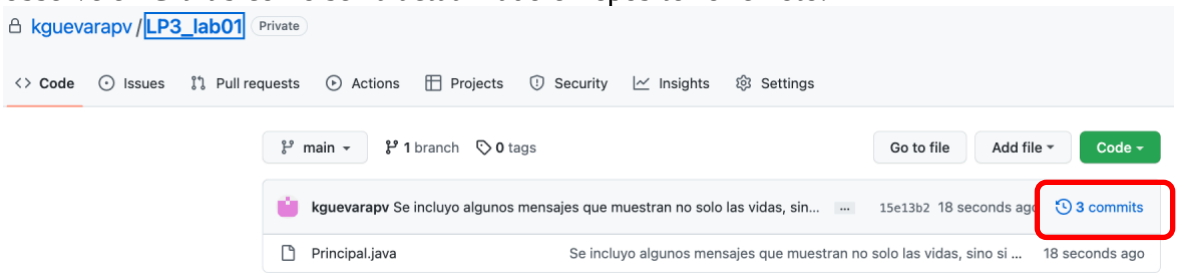
14. Ahora compruebe los commit que haya realizado en el repositorio local, los mismos que deben de figurar en su repositorio remoto



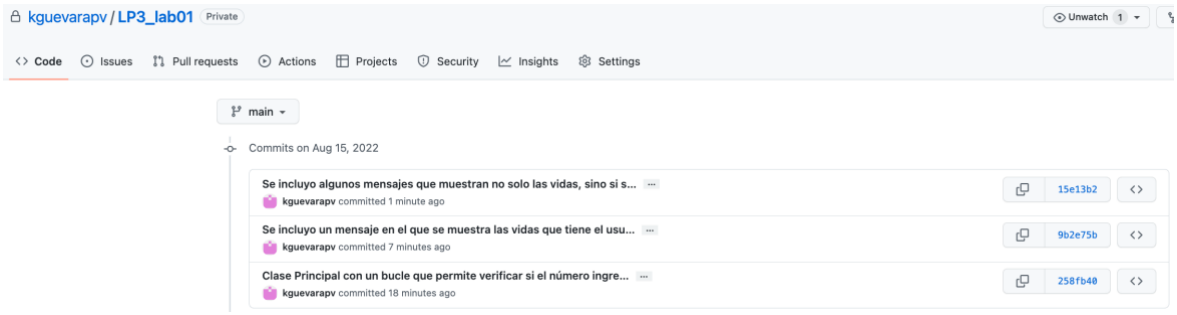
Actualizar el repositorio local y el remoto

15. Haga algunas modificaciones a su código y registre esos cambios en el repositorio local (mínimo dos commits más). Para registrar debe repetir para cada cambio realizado las acciones realizadas en los puntos 8 al 11.
16. Luego de las modificaciones realizadas en el repositorio local, ahora vamos a actualizar y subir esos cambios al repositorio remoto, que ya lo hemos vinculado anteriormente. Para esto sólo debe de realizar la siguiente orden:
    `git push`

    observe en Github como se ha actualizado el repositorio remoto.



Y observe cada commit realizado en el repositorio local, que ahora ya estan registrados en el remoto.



V

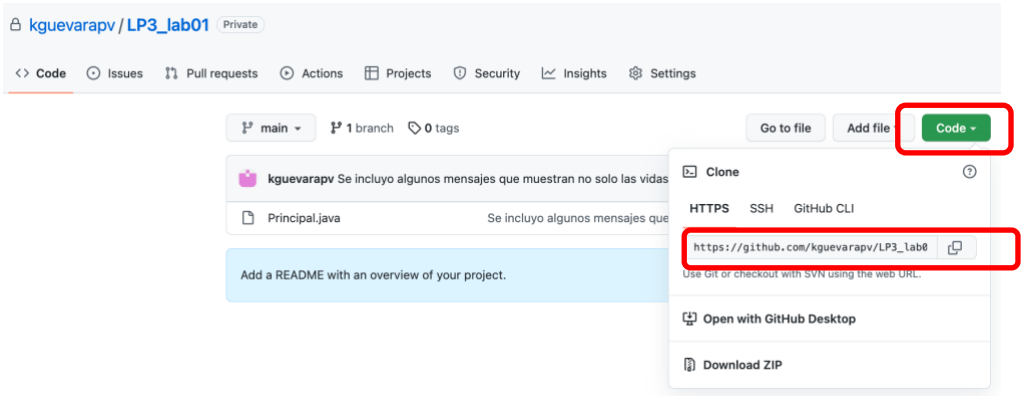
EJERCICIOS

Para presentar los ejercicios de este laboratorio, debe elaborar un informe que tenga en detalle la explicación de lo realizado en cada uno ellos y las capturas de las imágenes correspondientes como evidencia del trabajo.

Clonar un repositorio remoto a uno local

Ahora vamos a suponer que usted tiene código almacenado en el repositorio remoto (el que guardo la sesión anterior) y desea clonarlo en su máquina local, para realizar algunas actualización al mismo.

1. Ubíquese en una carpeta donde desea tener el directorio del repositorio que clonaremos. Esto lo puede hacer desde el explorador y desde la ventana del **cmd** con el comando **cd**
2. Estando en Github y en el repositorio que desea clonar, acceda a la opción **Code** y copie el url que le muestra.



3. Abra el **cmd** y estando en el directorio del punto 1, ejecute:
   git clone https://github.com/kguevarapv/LP3_lab01.git

   Nota: es posible que le pida sus credenciales de github.
4. Verifique que en la carpeta donde se encuentre ya tenga el directorio con el nombre del repositorio que acaba de clonar:

dir

acceda con el comando **cd** al directorio y observe que contiene los archivos fuentes que se encuentran en el repositorio remoto.

Nota: A partir de este punto usted podría hacer las modificaciones al código, ir registrando las versiones tanto en el repositorio local y en el repositorio remoto.

5. En el repositorio local cree dos ramas: **rama1** y **rama2**
6. Modifique el código de la clase Principal, de modo que, en caso el usuario adivine el número, se haga acreedor de un premio y el programa le indique el premio que se gano
7. Cambiese a la **rama1 y** realice el registre los cambios del punto 2 en la **rama1**.
8. Registre los cambios del repositorio local, especificamente de la rama1 en el repositorio remoto.

   Verifique que en el repositorio remoto ya se evidencie la rama1 y los commit que haya realizado en esa rama.
9. Modifique el proyecto, agregando la siguiente clase (esta no tiene el main()):

```
public class Regalo {
    public static String elegir(int vida) {
        String gift = "";
        switch(vida) {
        case 1: gift = "Un pasaje al caribe"; break;
        case 2: gift = "Un visita al museo más cercano a tu casa"; break;
        case 3: gift = "Una entrada al cine"; break;
        }
        return gift;
    }
}
```

   Y el la clase **Principal**, modifique el código de forma que al adivinar le muestre ell premio que se gano, de acuerdo a las vidas utilizadas.

10. Ahora, muevase a la **rama2** y realice las actualizaciones del repositorio local pero en la rama2.
11. Actualice estas últimas modificaciones en el repositorio remoto. Evidencie que éste repositorio ya tiene la rama2.
12. Realice un **merge** de la rama1 a la rama master. Evidencie lo realizado
13. Realice un **merge** de la rama2 a la rama master. Evidencie lo realizado

Nota: tenga mucho cuidado al momento de realizar los merge, pues no se sigue un orden adecuado, es posible que se generen conflictos en el registro de las versiones. Es por esta razón que cuando se trabaja en equipos de varias personas, se recomienda que solo el lider del proyecto sea el responsable de realizar los merge una vez que haya realizado la revisión correspondiente.

VI

BIBLIOGRAFÍA

- Deitel Paul, Deitel Harvey. “COMO PROGRAMAR EN JAVA”, 9na. Edición, Edit. Prentice Hall, 2012.
- Chacon Scott, Straub Ben. Pro Git. EVERYTHING YOU NEED TO KNOW ABOUT GIT. 2da. Edición, Apress.
- Video: Git y Github. FaztWeb.com. https://www.youtube.com/watch?v=HiXLkL42tMU

RÚBRICA PARA LA CALIFICACIÓN DEL LABORATORIO

CRITERIO A SER EVALUADO		Nivel				
		A	B	C	D	NP
R1	Uso de estandares y buenas práticas de programación	2.0	1.5	1.0	0.5	0
R2	Uso adecuado de los fundamentos de git y github	3.0	2.3	1.5	0.8	0
R3	Registro oportuno de los commits en el repositorio local y el remoto	3.0	2.3	1.5	0.8	0
R4	Informe con las evidencias de la ejecución correcta de cada uno de los ejercicios (cada ejercicio desde el segundo, se evalua con medio punto)	6.0	-	-	-	0
R5	Informe esta bien detallado, bien redactado y ordenado.	4.0	3.0	2.0	1.0	0
R6	Entrega oportuna y de acuerdo a las especificaciones dadas.	2.0	0.7	0.3	0.3	0
	Total	**20**				