



UNIVERSIDAD
SERGIO ARBOLEDA

TODOS A LA U

Fórmate digital

MasterClass Unidad 2

Desarrollo de Software

¿Qué es GIT?

Git (pronunciado «guit»/git[aclaración requerida] 2) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.



¿Cómo inicia el proyecto?



- Un miembro del equipo inicia el repositorio

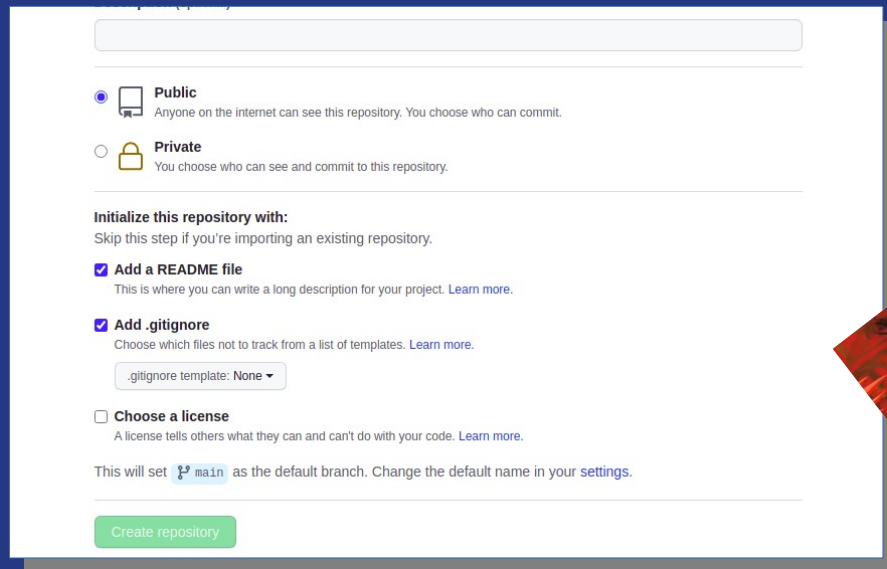
Hay dos caminos al iniciar

Crear el repositorio con al menos un archivo en la nube y CLONAR en local



- Crear el repositorio VACÍO en la nube y conectarlo al ya existente en local.

Iniciar en remoto, clonar en local



The screenshot shows the GitHub 'Create new repository' page. It includes options for repository visibility (Public or Private), a section to initialize the repository with a README file and a .gitignore file, and a section to choose a license. A green 'Create repository' button is at the bottom.

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☒ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
.gitignore template: **None**

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

Create repository



```
cd directorio_repo  
git clone url_repositorio
```


Iniciar en local y luego conectar con repositorio en nube.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. Learn more

☐ **Add .gitignore**

Choose which files not to track from a list of templates

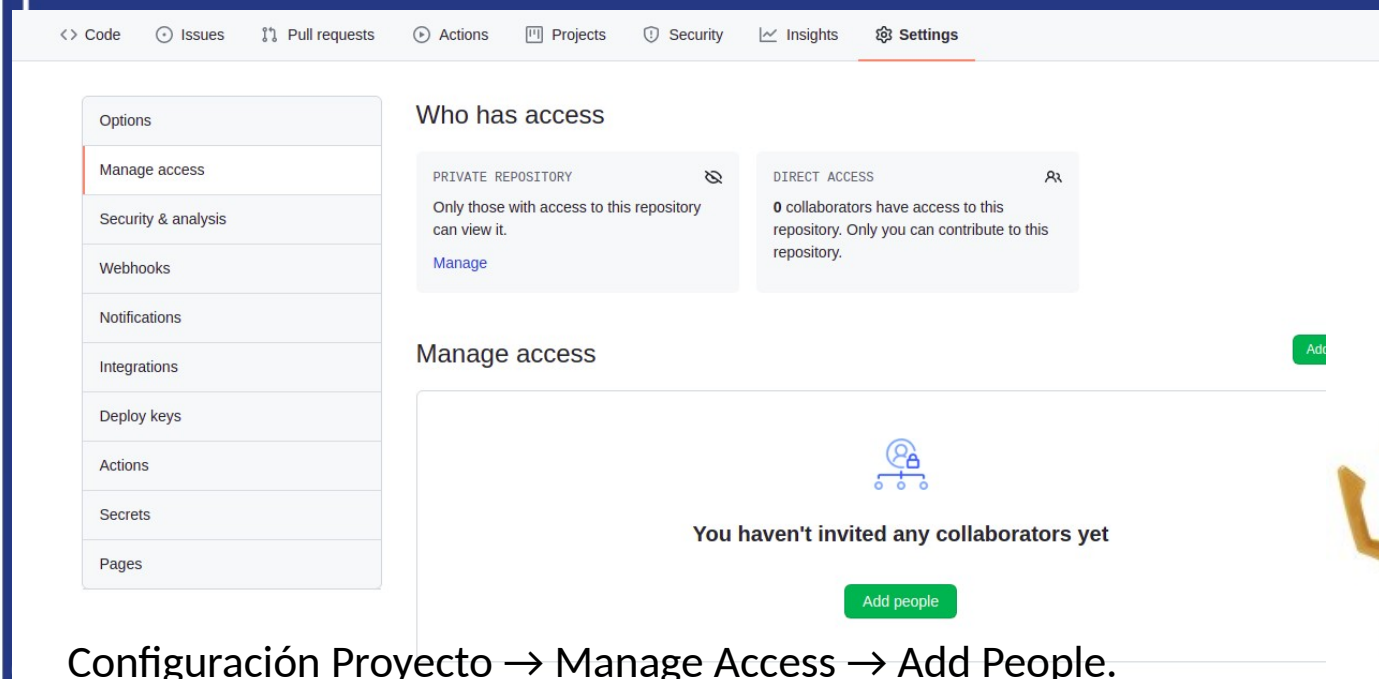
☐ **Choose a license**

A license tells others what they can and can't do with your code



- `git init`
- `git remote add origin url_repo`
- `.`
- *.subir el código, ¡ya lo veremos!.*
- `.`
- `git push -u origin master`

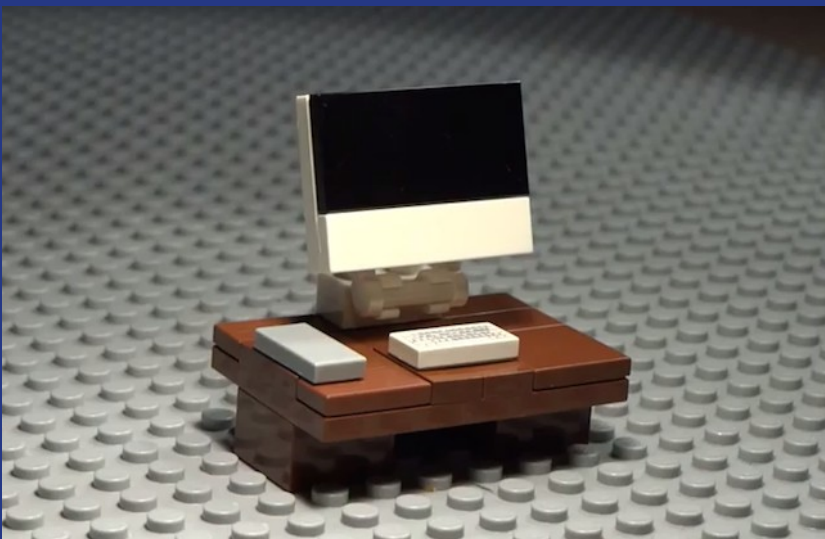
Una vez creado el repositorio, ¡el equipo se une!



Configuración Proyecto → Manage Access → Add People.
En esta parte se agrega el correo de cada miembro del equipo.



Una vez creado el repositorio, ¡el equipo se une!



Todos los miembros del equipo **ACEPTAN** la invitación que llega a su correo.



Ahora todos hacemos GIT CLONE

Con la instrucción *git clone* cada miembro del equipo descarga el código inicial.

```
cd directorio_proyecto  
git clone url_repositorio
```



CREACIÓN DE RAMAS...

¿Qué es una rama?

Una rama es una versión particular del código.
Cada miembro del equipo creará su rama a partir de una rama común.

La rama común suele ser master. En la rama máster **debe estar el código completamente funcional.**

¿Cómo se crea una rama?

git checkout -b *nombreRama*
→ *para crear la rama*

git checkout -b superman
git push -u origin superman



git push -u origin *nombreRama*
→ *para subir la rama!*

git checkout -b batman
git push -u origin batman



git checkout -b flash
git push -u origin flash



Ahora tenemos 3 ramas que vienen de la rama master

RAMA
MASTER

RAMA superman



RAMA batman



RAMA flash



Cada desarrollador empieza a modificar su código

RAMA superman



```

//...
}
// Author: Ravi Kantan Ganesan, Cavellia
// blog: www.distributedteam.com
// Email: ganesanr@gmail.com

HttpServletResponse response =
    HttpServletResponseImpl.getInstance();

public class HttpServletResponse implements HttpServletResponse {

    //...

    protected void service(HttpServletResponse request, HttpServletResponse response) throws
        ServletException, IOException {
        //...
        switch (request.getMethod()) {
            case "GET":
                doGet(request, response);
                break;
            case "POST":
                doPost(request, response);
                break;
        }
    }
}

```

RAMA batman

[illegible]

RAMA flash

[illegible]

Integrar el código

```
git checkout rama_destino  
git pull  
git merge mi_rama
```

Sitúo mi código en la rama destino

- Actualizo mi local
- Unifico código

```
/*  
 * Controller para el servicio de usuarios  
 *  
 * @author: jorge.fernandez@sergioarboleda.edu.co  
 */  
package edu.sergioarboleda.controller;  
  
import edu.sergioarboleda.model.Usuario;  
import edu.sergioarboleda.service.UsuarioService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.*;  
  
@RestController  
@RequestMapping("/usuarios")  
public class UsuarioController {  
  
    private final UsuarioService usuarioService;  
  
    public UsuarioController(UsuarioService usuarioService) {  
        this.usuarioService = usuarioService;  
    }  
  
    @GetMapping("/{id}")  
    public ResponseEntity<Usuario> getUsuarioById(@PathVariable Long id) {  
        Usuario usuario = usuarioService.findById(id);  
        if (usuario != null) {  
            return ResponseEntity.ok(usuario);  
        }  
        return ResponseEntity.notFound().build();  
    }  
  
    @PostMapping  
    public ResponseEntity<Usuario> crearUsuario(@RequestBody Usuario usuario) {  
        Usuario nuevoUsuario = usuarioService.crear(usuario);  
        return ResponseEntity.ok(nuevoUsuario);  
    }  
  
    @PutMapping("/{id}")  
    public ResponseEntity<Usuario> actualizarUsuario(@PathVariable Long id, @RequestBody Usuario usuario) {  
        Usuario usuarioActualizado = usuarioService.actualizar(id, usuario);  
        return ResponseEntity.ok(usuarioActualizado);  
    }  
  
    @DeleteMapping("/{id}")  
    public ResponseEntity<Void> eliminarUsuario(@PathVariable Long id) {  
        usuarioService.eliminar(id);  
        return ResponseEntity.noContent().build();  
    }  
}
```



Resolvemos conflictos si los hay.



Ajusto el archivo con el bloque correcto, o la mezcla de los dos.

[illegible]

Resuelto el conflicto, guardamos cambios y los subimos

```
git add --all  
git commit -m "Cuento qué hice"  
git push
```

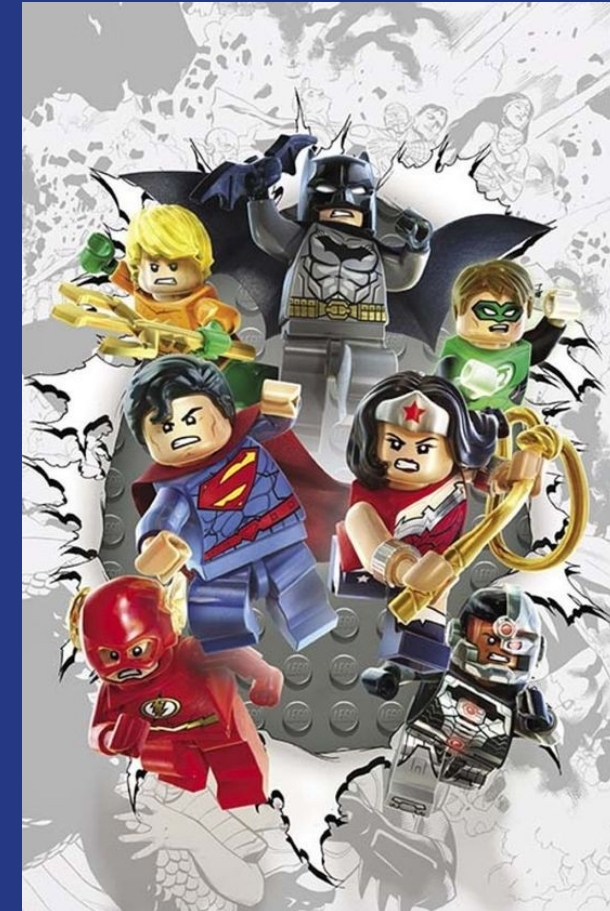
- Se agregan cambios
- Se confirman los cambios
- Se sube a la nube

```
/**  
 * @author Sergio Arboleda  
 * @since 1.0.0  
 * @see https://www.udacity.com/course/ud900  
 */  
package com.udacity.ud900.controller;  
  
import com.udacity.ud900.model.*;  
import com.udacity.ud900.service.*;  
import org.springframework.web.servlet.mvc.annotation.annotation.AnnotationMethodMapping;  
import org.springframework.web.servlet.mvc.annotation.annotation.AnnotationMethodHandlerMethod;  
import org.springframework.web.servlet.mvc.annotation.annotation.AnnotationMethodHandlerException;  
  
@AnnotationMethodMapping  
public class LoginController {  
  
    private final UserService userService;  
  
    public LoginController(UserService userService) {  
        this.userService = userService;  
    }  
  
    @RequestMapping(method = RequestMethod.POST)  
    public ModelAndView login(@RequestBody LoginRequest loginRequest) {  
        ModelAndView modelAndView = new ModelAndView();  
        User user = userService.login(loginRequest.getUsername(), loginRequest.getPassword());  
        if (user != null) {  
            modelAndView.addObject("user", user);  
            modelAndView.setViewName("login_success");  
        } else {  
            modelAndView.setViewName("login_failure");  
        }  
        return modelAndView;  
    }  
}
```



Al finalizar el proceso

Al finalizar el proceso, **todo el equipo** hará
git pull
Y tendremos el código actualizado



Qué es un ambiente de trabajo

- Muchas veces, cuando estamos desarrollando utilizamos algunos valores que dependen EXCLUSIVAMENTE de nuestra máquina.
- La máquina en la que trabajamos, tiene un sistema operativo, puede tener un servidor de base de datos y nuestra configuración puede ser distinta. Por ejemplo, puede que hayamos establecido una contraseña o un usuario distinto para la base de datos.
- A este conjunto de configuraciones dependientes de la máquina es a lo que llamamos AMBIENTE.

Necesidad de separar ambientes

Muchas veces en producción se generan errores debido a que no se modificó el valor de una variable, Una URL apunta a una máquina local o configuraciones semejantes. La solución inicial es dejar escritas todas las posibilidades y comentar aquellas no se utilizan.

```
spring.datasource.url=jdbc:mysql://localhost:3306/carsDB
spring.datasource.username=root
spring.datasource.password=micontraseñasecreta!
server.port=8080
#spring.datasource.url=jdbc:mysql://10.0.0.37:3306/carsDB
#spring.datasource.username=admin
#spring.datasource.password=lacontraseñadeproduccion!
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto = update
```

Cómo separar ambientes

Para soportar perfiles, la configuración y variables de entorno, crearemos tantos archivos `application.properties` como sea necesarios. Este archivo que es donde reside la configuración se encuentra en la carpeta `src/main/resources`

Así se verán los archivos, cuyo nombre inicia con `application` y luego de un guión escribimos el nombre del ambiente, terminamos con `.properties`.

La configuración común a todos los ambientes se puede quedar en el archivo inicial `application.properties`

`application-dev.properties`
`application-qa.properties`
`application-prod.properties`

`.`
`.`
`.`

`application-myenv.properties`

Cómo separar ambientes

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.mode
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto = update
```

application-dev.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/carsDB
spring.datasource.username=root
spring.datasource.password=micontraseñasecreta!
server.port=8080
```

application-prod.properties

```
spring.datasource.url=jdbc:mysql://10.0.0.37:3306/carsDB
spring.datasource.username=admin
spring.datasource.password=lacontraseñadeproduccion!
server.port=80
```

Cómo separar ambientes

Una vez separados los archivos, en el momento de compilar y ejecutar bastará con llamar maven de la siguiente manera

```
mvn -Dspring.profiles.active=dev spring-boot:run
```

Se escribirá el nombre del perfil que se quiera compilar y ejecutar!

Ejercicio

Se propone en esta sesión de revisión de la unidad, realizar completo el ejercicio de compartir un repositorio en equipo, desarrollar una mínima funcionalidad y separar en ambientes.