Introducción a Maven, GIT, GitHub

Introducción a Maven, GIT, GitHub
Autor: Luis Daniel Benavides Navarro

Herramientas para el Taller

- Maven: Automatiza y estandariza el flujo de vida de la construcción de software
- Git: Administrador descentralizado de configuraciones
- Heroku: Plataforma como servicio (PaaS) de computación en la nube

MAVEN

¿Qué es Maven?

- Estandariza la estructura física de los proyectos de software (Estructura de archivos)
- Maneja dependencias (Librerías) automáticamente desde repositorios
- Administra el flujo de vida de construcción de software:
  - Descargar dependencias
  - Compilar
  - Ejecutar pruebas
  - Generar reportes
  - Empaquetar, e.g., Jar, War
  - Desplegar
- Usa el principio de Convención sobre configuración

Instalando MAVEN

- **Aprenda a ejecutar Maven desde la linea de comandos**
- Descargar Maven en http://maven.apache.org/download.html
- Necesitan terner Java Instalado (7 o 8)
  - java -version
  - ```
    java version "1.8.0"
    Java(TM) SE Runtime Environment (build 1.8.0-b132)
    Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
    ```
- Siga las instrucciones en http://maven.apache.org/download.html#Installation
- Si quedó bien instalado al ejecutar mvn -version se desplegará:
    ```

```
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-14T12:29:23-05:00)
Maven home: /Users/dnielben/Applications/apache-maven-3.2.5
Java version: 1.8.0, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "mac os x", version: "10.10.1", arch: "x86_64", family: "mac"
```

Creando un proyecto

- Abra un Shell del sistema operativo
- Ubíquese en el directorio donde almacenará sus proyectos
- `mvn archetype:generate -DgroupId=edu.escuelaing.arsw.ASE.app -DartifactId=mi-primera-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
- `cd mi-primera-app/`
- Estructura de archivos

```
.
|____pom.xml
|____src
| |____main
| | |____java
| | | |____edu
| | | | |____escuelaing
| | | | | |____app
| | | | | | |____App.java
| |____test
| | |____java
| | | |____edu
| | | | |____escuelaing
| | | | | |____app
| | | | | | |____AppTest.java
```

Código fuente de `App.java`:

```
package edu.escuelaing.app;
/**
 * Hello world!
 *
 */
public class App
```

```
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

El pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>edu.escuelaing.app</groupId>
   <artifactId>mi-primera-app</artifactId>
   <packaging>jar</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>mi-primera-app</name>
   <url>http://maven.apache.org</url>
   <dependencies>
     <dependency>
       <groupId>junit</groupId>
       <artifactId>junit</artifactId>
       <version>3.8.1</version>
       <scope>test</scope>
     </dependency>
   </dependencies>
 </project>
```

- Las coordenadas del proyecto son:
  - <groupId>edu.escuelaing.app</groupId>
    <artifactId>mi-primera-app</artifactId>
    <version>1.0-SNAPSHOT</version>

Arquitectura de plugins de MAVEN

- Los plugins son los que hacen las cosas en Maven
- archetype es un plugin que permite crear proyectos a partir de un plugin
- Maven tiene una arquitectura de plugins
- Hay plugins para:
  - compilar,
  - ejecutar tests,
  - generar documentación,

- empaquetar, etc.

Construir el proyecto

- mvn package
  - ```
    [INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mi-primera-app ---
    [INFO] Building jar: /Users/dnielben/NetBeansProjects/mi-primera-app/target/mi-primera-app-1.0-SNAPSHOT.jar
    [INFO] ------------------------------------------------------------------------
    [INFO] BUILD SUCCESS
    [INFO] ------------------------------------------------------------------------
    [INFO] Total time: 17.029 s
    [INFO] Finished at: 2015-01-19T13:51:56-05:00
    [INFO] Final Memory: 16M/59M
    [INFO] ------------------------------------------------------------------------
    ```
- "package" no es un plugin, es una fase
- El comando ejecutará todas las fases hasta la fase package
- OJO: El directorio target está ahora en la estructura de archivos
- Ejecutar el applicativo
  - ```
    java -cp target/mi-primera-app-1.0-SNAPSHOT.jar edu.escuelaing.arsw.ASE.app.App
    ```
- Y la ejecución
  - ```
    Hello World!
    ```

Arquitectura de fases

- Las fases agrupan multiples objetivos (goal) sobre los plugins
- El comando mvn [fase], ejecuta todas las fases hasta la fase indicada
- Las fases del ciclo de vida por defecto que se usan comunmente
  - **validate:** Proyecto correcto y toda información disponible
  - **compile**
  - **test:** hacer pruebas unitarias usando un framework de pruebas concreto
  - **package:** Tomar el código compilado y empaquetarlo en un formato específico,e.g., jar
  - **integration-test:** procesar y desplegar el software en un entorno donde las pruebas de integración se puedan ejecutar.
  - **verify:** ejecutar cualquier revisión para verificar los criterios de calidad del paquete
  - **install:** instalar el paquete el el repositorio local para que esté disponible como
    ```

dependencia.
    - **deploy:** En un ambiente de publicación de nuevas versiones o en un ambiente de producción, desplegar el software
- Otras fases
    - **clean:** Limpia los artefactos creados por construcciones previas
    - **site:** genera el sitio de documentación para este proyecto
- Encadenando fases y objetivos de plugin
    - mvn clean dependency:copy-dependencies package
    - Limpia, copia dependencias y ejecuta todas las fases hasta el empaquetado!
- Generando el sitio de la documentación
    - mvn site
- Para generar el Javadoc como parte del site debe adicionar el plugin de javadoc a la parte de reportes en el POM.
    -
```
<project>
...
<reporting>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-javadoc-plugin</artifactId>
<version>2.10.1</version>
<configuration>
...
</configuration>
</plugin>
</plugins>
...
</reporting>
...
</project>
```
- Si quiero generar el javadoc como un elemento independiente de la documentación, para empaquetarlo en el JAR debo agregar el plugin en la sección de build.
    -
```
<project>
...
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-javadoc-plugin</artifactId>
<version>2.10.1</version>
<configuration>
...
</configuration>
</plugin>
```

```
</plugins>
...
</build>
...
</project>
```

- Comandos para generarlos:
  - ```
    mvn javadoc:javadoc
    mvn javadoc:jar
    mvn javadoc:aggregate
    mvn javadoc:aggregate-jar
    mvn javadoc:test-javadoc
    mvn javadoc:test-jar
    mvn javadoc:test-aggregate
    mvn javadoc:test-aggregate-jar
    ```
- Esto genera los javadoc y los empaqueta con el Jar.

Arquitectura de repositorios

- Maven viene configurado con un repositorio por defecto
- http:// repo1. maven.org/ maven2
- Cada vez que se ejecuta el descarga los plugins o dependencias necesarias y las almacena en el repositorio local
- Una vez descargadas no se descargan nuevamente a menos que sean borradas
- Para librerías que no son libre o públicamente accesibles usted puede crear repositorios con los contenidos o desplegarlas localmente

Dependencias

- Las dependencias listan las dependencias que un proyecto necesita para compilar, probar o correr o para lo que sea
- Para cada dependencia usted necesita definir por lo menos:
  - groupid
  - artifactid
  - version
  - scope: puede tener valores como compile, test y runtime
- Maven primero busca en su directorio local y luego en su repositorio remoto para obtener las dependencias
- La mayoría de librerías open source están disponibles en el repositorio por defecto, hay que saber las coordenadas de la librería en le repositorio.
    ```

Manejando recursos adicionales

- los directorios de recursos pueden adicionar recursos adicionales
  - src/main/resources: Application/Library resources
  - src/test/resources: Test resources
  - src/site/resources: recurso de documentación adicionales que quieren que sean copiados en el target
  - **src/site/resources/PSPXX: LA DOCUMENTACIÓN PSP DEL PROYECTO.**

GIT

¿Qué es GIT?

- Un sistema de control de versiones
  - Un sistema que almacena cambios sobre un archivo o un conjunto de archivos
  - Un sistema que permite recuperar versiones previas de esos archivos
  - Permite otras cosas como el manejo de ramas (Branches)

Tipos de sistemas de control de versiones

- Sistema Local de manejo de versiones (e.g., RCS aún distribuido en los MACs)
- Sistema Centralizado de manejo de versiones (Subversion)
- Sistema Distribuido de manejo de versiones (GIT)

Sistema Local de manejo de versiones (e.g., RCS aún distribuido en los MACs)

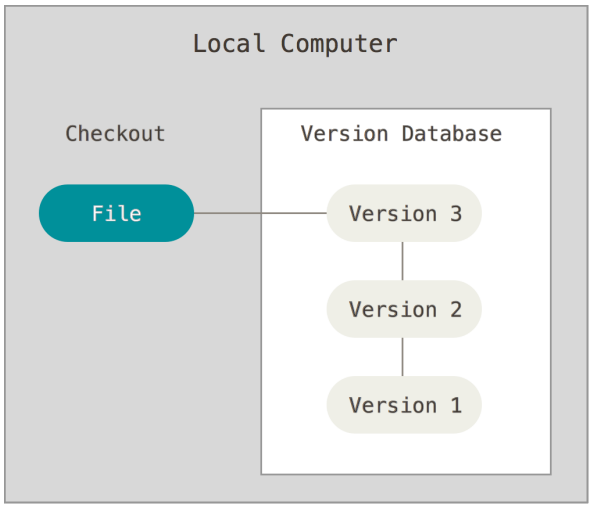


Imagen extraída del sitio web de Git (www.git-scm.com)

Sistema Centralizado de manejo de versiones (Subversion)

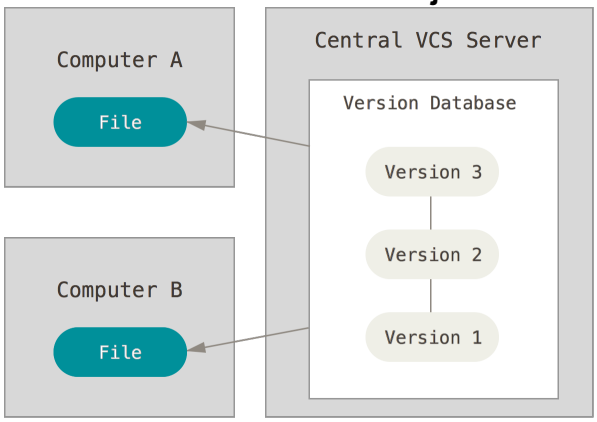


Imagen extraída del sitio web de Git (www.git-scm.com)

Sistema Distribuido de manejo de versiones (GIT)

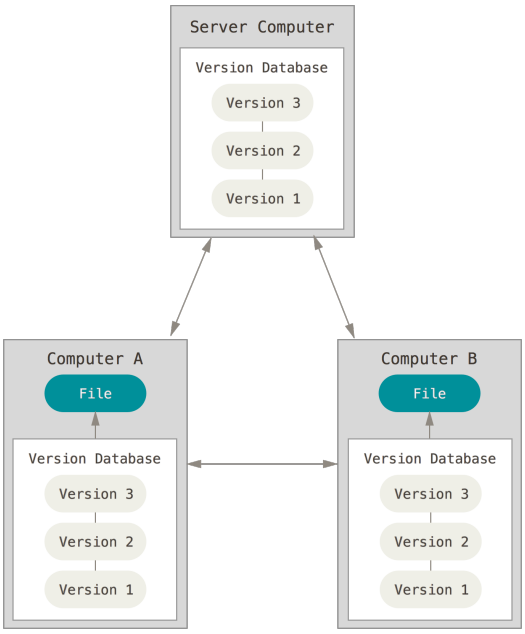


Imagen extraída del sitio web de Git (www.git-scm.com)

¿Cómo almacena datos GIT?

Los VCS tradicionales, manejan los datos como el archivo original y una serie de diferencias que son almacenadas en la base de datos

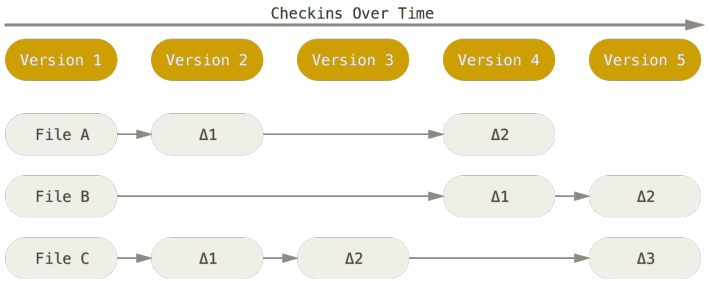


Imagen extraída del sitio web de Git (www.git-scm.com)

GIT almacena cada versión como una fotografía de todos los archivos, y el archivo que no haya cambiado lo crea como un link a la versión anterior.

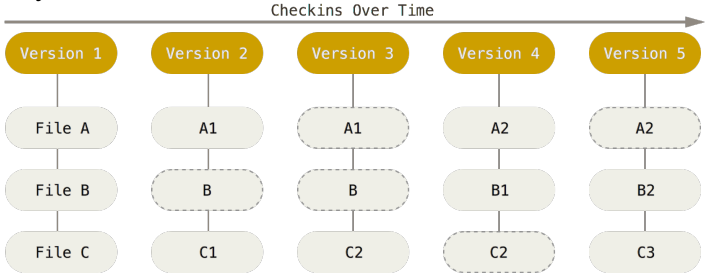


Imagen extraída del sitio web de Git (www.git-scm.com)

Otras características

- La mayoría de operaciones son locales
- Git tiene integridad: todo es revisado por un hash antes de ser almacenado
- Git solo adiciona datos,i.e., es muy difícil hacer algo que no se pueda deshacer

Los tres estados de un archivo en GIT (IMPORTANTISIMO)

- Commited: su archivo esta seguro almacenado en su repositorio LOCAL
- Modified: Archivo modificado pero aun no Commited
- Staged: Archivo marcado para ser cometido en la siguiente versión del sistema

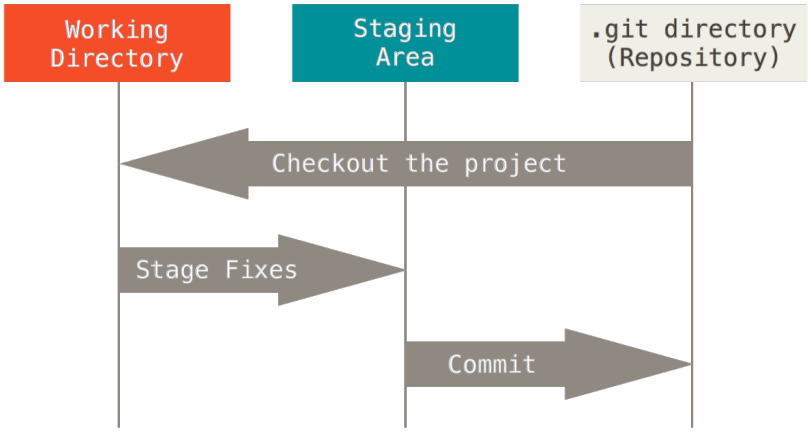


Imagen extraída del sitio web de Git (www.git-scm.com)

El flujo de trabajo típico de GIT

1. Modificar archivos en su directorio de trabajo
2. Marcar los archivos como staged
3. Hacer un commit para pasar los archivos de Staged a Commited.

Instale GIT

- Siga las instrucciones en el sitio web de GIT
  - http://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- ```
  mbp-de-luis-2:~ dnielben$ git --version
  git version 2.2.1
  ```
  ```

Configure su identidad y su editor por defecto

- $ git config --global user.name "John Doe"
- $ git config --global user.email johndoe@example.com
- git config --global core.editor emacs
- Revise su configuración
    - `git config --list`

      ```
      ...
      user.name=Daniel Benavides
      user.email=dnielben@gmail.com
      core.editor=emacs
      ```

¿Cómo obtener ayuda?

- git help <verb>
  $ git <verb> --help
  $ man git-<verb>
- Ejemplo
    - $ git help config
- El libro en línea en el sitio web de GIT

Crear un repositorio GIT

- En el directorio de su proyecto escriba
    - `$ git init`
- Obtendrá el siguiente mensaje
    - `Initialized empty Git repository in /Users/dnielben/NetBeansProjects/mi-primera-app/.git/`
- *Ha creado un repositorio local de Git!*

Agregar archivos al control de versiones de su repositorio

- Use el comando "add"
    - `$ git add pom.xml`
      `$ git status`

- Obtendrá la siguiente salida

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   pom.xml

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        src/
        target/
```

Confirmar los cambios en GIT y crear una nueva versión (commit)

- Para crear una nueva versión y confirmar los cambios (commit) escriba
  - `$ git commit -m 'Primera versión del proyecto'`
- Obtendrá la siguiente salida
  - `[master (root-commit) b75ab28] Primera versión del proyecto`
  - `1 file changed, 40 insertions(+)`
  - `create mode 100644 pom.xml`
- Si utiliza el comando de “git status” obtendrá algo similar a

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        src/
        target/

nothing added to commit but untracked files present (use "git add" to track)
```

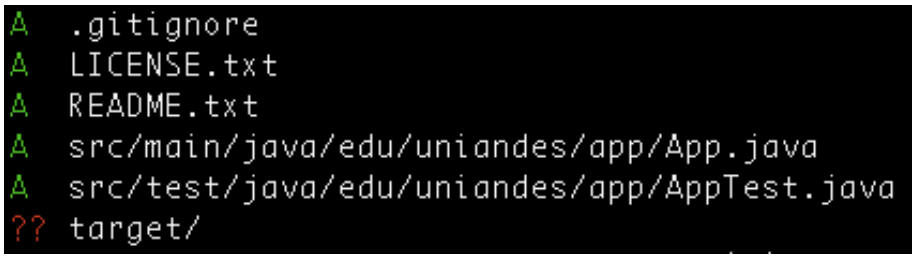
Trabajando con repositorios remotos (remotes)

- Para saber el listado de directorios remotos con los que ha estado trabajando digite
  - `$git remote`
- Por ahora no hay ningún repositorio configurado
- **Diríjase al sitio web de Github y cree un repositorio nuevo en GIthub**

- En su su proyecto git local adicione el repositorio remoto que acaba de crear con el nombre “origin"
    - `$ git remote add origin https://github.com/dnielben/miprimerrep.git`
- Revise que el repositorio de nombre origin esta en su listado con “git remote"
- Revise que las urls de los repositorios remotos están correctas con “git remote -v"
- Ahora empuje la ultima versión del proyecto que tiene almacenada en su computador
    - `git push -u origin master`
- Obtendrá una salida similar a:
    - `Username for 'https://github.com': suusuario`
    - `Password for 'https://suusuario@github.com':`
    - `Counting objects: 3, done.`
    - `Delta compression using up to 4 threads.`
    - `Compressing objects: 100% (2/2), done.`
    - `Writing objects: 100% (3/3), 624 bytes | 0 bytes/s, done.`
    - `Total 3 (delta 0), reused 0 (delta 0)`
    - `To https://github.com/dnielben/miprimerrep.git`
    - `* [new branch] master -> master`
    - `Branch master set up to track remote branch master from origin.`
- Ahora ya esta sincronizado su repositorio local con su repositorio remoto!
- Su rama (brach) de trabajo por defecto es “master” y su repositorio remoto por defecto es “origin"

Agregar más archivos a su proyecto de Git y su repositorio remoto

- Es una práctica recomendada crear un archivo README, otro LICENSE y otro .gitignore
    - `$ echo 'Mi primer proyecto' > README.txt`
    - `$ echo 'TODO: Copiar el texto de la licencia http://www.gnu.org/licenses/gpl.html' > LICENSE.txt`
    - `$ echo '# TODO: Copiar los ontenidos de https://github.com/github/gitignore/blob/master/Java.gitignore' > .gitignore`
- Adicione los archivos
    - `$ git add *.txt`
    - `$ git add .gitignore`
- OJO: Git ignore puede incluir expresiones glob, para indicarle a git que archivos ignorar
- Adicione el directorio src y recursivamente todos los contenidos de este
    - `$ git add src`

- Revise el estado de su proyecto (la opción “-s” muestra una lista compacta)
  - $ git status -s
- Obtendrá algo así:
  - 

Ignorando algunos archivos

- Modifique su archivo .gitignore para que no considere el directorio target
  - agregue las siguientes líneas
    - `# Ignora todos los archivos en el directorio target en los proyectos creados con maven`
    - `target/`
- Revise nuevamente el status de su proyecto
  - `$ git status -s`
- Obtendrá la siguiente salida
  - 

- Mire que el archivo “.gitignore” que está en el área staged es diferente al modificado
- Entonces debemos agregarlo nuevamente al area staged
  - `git add .gitignore`
- y obtendrá
  - 

Realice ahora el commit y el push remoto

- Ahora realice el commit

    - `git commit -m 'Segunda versión del proyecto adicionando e ignorando más archivos'`
- Ahora empuje los nuevos cambios al repositorio remoto
    - $ git push
- Si alguien modificara el repositorio, usted puede obtener los últimos cambios con
    - $ git pull
- Este comando trae los cambios y los combina con los cambios que usted tenga

Obtener una copia de un repositorio remoto

- OJO, intente esto en otro directorio diferente a su proyecto
- Puede obtener una copia de un repositorio usando el comando "clone"
- Intente bajar su proyecto en otro directorio
    - `$ git clone https://github.com/dnielben/miprimerrep copiademirep`
- Ahora tiene una copia de su repositorio totalmente independiente, el trabajo en ada una será independiente y tendrá que sincronizarlos por medio del servidor central en github.
- En este proyecto usted ya podría ejecutar los comandos de maven
    - `$ mvn package`

Otros comandos útiles

- Ver exactamente que cambios(lineas) hizo y aun no ha enviado al area *staged*
    - `$ git diff`
- Ver exactamente que cambios(lineas) hizo y aun no ha enviado al repositorio local (commit)
    - `$ git diff --staged`
- Cuando quiero hacer commit de todo lo que he modificado y ya esta siendo seguido por git pero sin parar por el area staged
    - `$ git commit -a -m 'commit directo sin pasar por staged'`
- Remover archivos del proyecto y del control de Git
    - Remover un archivo del directorio pero no del control de Git
        - $ rm [archivo]
    - Remover archivo del control de Git, primero “rm” lo pasa a staged y el commit lo confirma
        - $ git rm [archivo]
        - $ git commit -m ‘Confirmar la eliminación del archivo'
    - Remover un archivo del area staged
        - $ git rm —cached [archivo]
    - Puede usar patrones glob para remover cosas

        - `$ git rm log/ \*.log` (Remueve los archivos que terminan con log en el directorio log)
        - `$ git rm \*~` (Remueve todos los archivos que terminan con ~)
    - Renombrar o mover archivos
        - `$ git mv file_from file_to`
    - Git también se da cuenta si mueve un archivo de la siguiente manera, se da cuenta que es el mismo archivo
        - `$ mv README.md README`
        - `$ git rm README.md`
        - `$ git add README`
    - Ver el historial de commits
        - `$ git log`

Crear la estructura de la tarea y el repositorio remoto

- Borre los repositorios locales y remotos que ha credo durante la práctica
- **Cree el proyecto para su primera tarea usando maven**
    - `$ mvn archetype:generate -DgroupId=edu.escuelaing.arem -DartifactId=psp0 -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
    - `$ cd psp0`
- Verifique que todo esté funcionando
    - `$ mvn package`
- Cree los archivos recomendados
    - `$ echo 'Mi primer proyecto' > README.txt`
    - `$ echo 'TODO: Copiar el texto de la licencia http://www.gnu.org/licenses/gpl.html' > LICENSE.txt`
    - `$ echo '# TODO: Copiar los ontenidos de https://github.com/github/gitignore/blob/master/Java.gitignore' > .gitignore`
- Cree el repositorio local de Git
    - `$ git init`
- Adicione los archivos al control de Git
    - `$ git add *.txt pom.xml .gitignore src`
    - `git status -s`
- agregue las siguientes líneas a .gitignore
    - `# Ignora todos los archivos en el directorio target en los proyectos creados con maven`
    - `target/`
    - `git status -s`
- agregue .gitignore nuevamente al area staged

    - `$ git add .gitignore`
- Cree la primera versión de su proyecto
    - `$ git commit -m 'Primera versión de proyecto PSP0 de Daniel Benavides'`
- **`Cree un repositorio en GitHub`**
- Adicione al repositorio remoto
    - `git remote add origin https://github.com/dnielben/psp0.git`
- Empuje su proyecto y haga que Git rastree los cambios en el repositorio remoto
    - `git push -u origin master`

Para entregar el proyecto

- Coloque los informes en el directorio src/site/resources
- Cree un archivo ZIP con los contenidos de su proyecto (Intente no empacar su directorio target)
- El README debe traer las instrucciones para obtener su repositorio desde github
- Al desempacar el proyecto y correr "mvn package" todo debería compilar.