P00: Entorno de pruebas

Para realizar las prácticas de la asignatura, tendremos que:

- Descargar y poner en marcha la máquina virtual de la asignatura
- Crear un repositorio Git remoto, en GitHub, y otro local, en nuestra máquina virtual
  - Crearemos un Token para acceder a nuestro repositorio remoto
  - Podemos guardar las credenciales de acceso al repositorio en nuestra máquina
  - Modificaremos el fichero .gitignore
  - Configuraremos git con las propiedades *user.name* y *user.email*
  - Siempre trabajaremos con los comandos git *add*, *git commit*, y *git push*
- Crear una cuenta educacional en JetBrains
- Configurar IntelliJ para usar maven de forma externa e instalar un plugin para pruebas
- Primeros pasos con Maven:
  - Estructura de directorios de un proyecto Maven
  - Construcción del proyecto

- Adicionalmente, proporcionamos un listado de programas instalados en la máquina virtual y que usaremos en la primera parte de la asignatura por si alguno de vosotros prefiere trabajar en nativo en lugar de usar el vdi

Máquina virtual de la asignatura

Para las prácticas en el laboratorio utilizaremos una Máquina Virtual en la que tendréis instalado el software que vamos a necesitar. En el laboratorio estará accesible desde el escritorio cuyo icono lleva el nombre “PPSS-2024-25".

El *login* y *password* de la máquina virtual es **ppss** en ambos casos.

Para trabajar con la máquina virtual desde vuestro ordenador necesitaréis tener instalado previamente ***VirtualBox*** (incluyendo VirtualBox ***Extension Pack***) y crear una nueva máquina virtual a partir del fichero *"ppss-2024-25.vdi"*.

En el siguiente enlace tenéis los ficheros necesarios para crear el vdi: enlace_descarga_fichero_vdi. Para poder acceder necesitáis acceder con vuestro **usuario de "gcloud"**. Aquí veréis cuatro ficheros con extensión .zip más un fichero adicional con las instrucciones para crear el fichero vdi a partir de ellos.

Para CREAR la nueva máquina virtual desde VirtualBox, a partir del fichero vdi, seleccionaremos la opción **Nueva.** Usaremos los siguientes valores:

- *Nombre: Podéis poner cualquier nombre arbitrario*
- *Tipo:* Linux
- *Version*: Ubuntu (64 bits)
- (en la siguiente pantalla): Procesadores: usad más de 1
- (en la siguiente pantalla) *Usar un archivo de disco duro virtual existente*. A continuación seleccionamos el fichero ***ppss-2024-25.vdi*** de nuestro disco duro.

Una vez creada la máquina virtual, y antes de iniciarla, es posible que necesites cambiar la configuración de la pantalla de VirtualBox para que no se muestre en un tamaño demasiado pequeño

(dependerá de tu ordenador). Seleccionamos la nueva máquina que hemos creado, y desde ***Configuración→ Pantalla*** puedes probar a ponerla al 200% si lo necesitas.

Finalmente elegimos la opción **Iniciar**.

IMPORTANTE: Antes de comenzar a hacer nada, recuerda que debes instalar ***GuestAdditions*** para no tener problemas al redimensionar la ventana de la máquina virtual. Cada vez que actualices la versión de VirtualBox, deberás volver a instalar *GuestAdditions* (desde el menú Devices).

Os recomendamos que activéis la compartición del portapapeles entre la máquina anfitrión y vuestra máquina virtual desde: *Devices→**Shared Clipboard**→Bidirectional*. De esta forma, podremos copiar/pegar el contenido del portapapeles entre ambas máquinas.

También podéis activar la opción para copiar ficheros entre ambas máquinas desde: Devices→***Drag and Drop***→*Bidirectional.*

Cuando termines el trabajo, siempre debes apagar la máquina pulsando en el icono azul de la esquina inferior izquierda, y seleccionar ***Salir → Apagar***.

Git: repositorio remoto, en *GitHub*

Es una muy buena práctica (indispensable para futuros trabajos profesionales en grupo) utilizar una **herramienta de gestión de versiones**, para así tener todo nuestro trabajo accesible desde cualquier lugar, y con el historial de versiones, por si nos interesa recuperar alguna versión anterior, crear nuevas versiones a partir de ellas, etc.

Aunque nosotros no vamos a trabajar en grupo, es interesante habituarnos a utilizar un repositorio remoto para, por ejemplo, tener siempre una copia de seguridad del trabajo realizado, continuar el trabajo iniciado desde cualquier otro ordenador en el mismo punto que lo dejamos, ..., entre otras cosas.

GitHub es un servicio que nos permite trabajar con repositorios Git (Git es una herramienta de gestión de versiones). Lo primero que haremos será **crearnos una cuenta** (gratuita) en GitHub, (https://github.com.

Para crear la cuenta seleccionamos "Sign up" (en la parte superior derecha). Os pedirá el correo electrónico (tenéis que usar vuestro correo institucional!).

Recibiréis un e-mail en la dirección de correo que hayáis especificado para confirmar vuestra dirección. Una vez verificada, se os pedirá un nombre de usuario y password de vuestra elección.

Debéis indicar que el grupo de trabajo está formado sólo por vosotros (1 sólo miembro). Marcaréis que sois estudiantes.

Una vez creada la cuenta, tenéis que **crear UN ÚNICO repositorio** que contendrá TODO el trabajo de prácticas que realicéis durante el curso: Al entrar en nuestra cuenta de GitHub, lo que se muestra es el "dashboard", en donde pulsaremos el botón "New" para crear el repositorio,.

El repositorio tendrá como nombre: **ppss-2025-Gx-apellido1-apellido2**, en donde:

- **Gx** es el identificador del grupo de prácticas al que vais a asistir durante todo el curso (si no realizáis ningún cambio de grupo, éste será el que tenéis oficialmente asignado.

  Los valores posibles son: G1..G9, G40, según la siguiente tabla:

Grupo de prácticas	Identificador
Miércoles de 9 a 11h	G1
Lunes de 9 a 11h	G2
Martes de 9 a 11h	G3
Lunes de 11 a 13h	G4
Martes de 17 a 19h	G5

Grupo de prácticas	Identificador
Martes de 19 a 21h	G6
Miércoles de 19 a 21h	G7
Miércoles de 17 a 19h	G8
Martes de 13 a 15h	G9
Martes de 11 a 13h	G40

- **apellido1** es el primer apellido (todo en minúsculas y SIN acentos ni eñes)
- **apellido2** es el segundo apellido (todo en minúsculas y SIN acentos ni eñes). Si algún alumno no tiene segundo apellido, entonces omitiremos esta parte

Una vez que hayáis puesto el nombre (ppss-2025-..., en el campo *Repository name*), debéis aseguraros de que:

- La casilla "***Private***" esté MARCADA, puesto que queremos crear un repositorio privado.
- La casilla: "***Add a README?"*** podéis marcarla.
- El desplegable "***Add .gitignore***" debe tener seleccionada la opción ***"Maven"***.

Después de crear el repositorio, deberéis añadir un **colaborador** . Para ello accedemos a la opción "***Settings***" de vuestro proyecto (icono de la rueda dentada que verás en la parte superior de la página en donde se muestra el proyecto).

A continuación seleccionamos "***Collaborators***", y a través del botón "***Add people***" debéis enviar una invitación a vuestro profesor de prácticas:

- abotia@gcloud.ua.es (antonio)
- rfb8@gcloud.ua.es (raúl)
- jviche@gcloud.ua.es (nacho)

Git: URL repositorio remoto y PAT

Cualquier repositorio remoto tiene una **URL** asociada, Para conocer dicha url accedemos desde la página de nuestro repositorio en GitHub, a la pestaña "***<> Code***", y podemos copiar la url **https** mostrada.

Desde nuestro repositorio local necesitaremos acceder al repositorio remoto para añadir y recuperar información y para ello necesitaremos identificarnos en GitHub. Un **Personal access token** (PAT) es una alternativa al uso de contraseñas para la autenticación en GitHub cuando accedemos a través de línea de comandos.

Vamos a crear un PAT para acceder a nuestro repositorio en GitHub. Cuando hagamos un *git push* o *git pull*, por ejemplo, nos pedirá nuestras credenciales. En lugar de introducir el password de nuestra cuenta en GitHub, usaremos el PAT en su lugar.

Desde el icono de tu perfil (en la esquina superior derecha de la página), accede a :

Settings→Developer Settings (al final del menú de la izqda)→Personal access tokens→Tokens (classic)→ Generate new token→ Generate new token(classic)

- El campo ***Note*** es para indicar una descripción opcional del token
- Opcionalmente, también podemos indicar una **fecha de expiración** del token.
- Marcaremos como mínimo el *scope* **repo** (para indicar qué permisos concedemos al uso de dicho token).
- Finalmente pulsaremos el botón ***Generate Token***.

A continuación configuramos los permisos del token. Como mínimo selecciona los permisos del *scope* **repo**.

El token generado se nos muestra en pantalla, pero SÓLO UNA VEZ. Asegúrate de copiarlo porque no podrás "volverlo a ver".

Si olvidas el token de acceso, puedes volver a generar otro tantas veces como quieras.

IMPORTANTE: Cuando desde el terminal os solicite la contraseña de GitHub para acceder a vuestro repositorio, ten en cuenta que el cursor NO se moverá al teclear los caracteres, ni veréis lo que estáis tecleando!!. Es normal.

Git: repositorio local, credenciales, fichero .gitignore y configuración

comando git clone

(para descargar una copia del repositorio)

Una vez creado el repositorio, vamos a "descargar y sincronizar" dicho repositorio remoto en un directorio de nuestra máquina virtual. A este proceso lo llamaremos CLONAR el repositorio remoto. Si trabajas en el laboratorio, este paso lo tendrás que hacer SIEMPRE.

Supongamos que nuestro directorio de trabajo (local) va a ser "**$HOME/practicas**". Para **CLONAR** el repositorio en dicho directorio, necesitamos la URL del repositorio remoto en GitHub. Usaremos el siguiente comando desde un terminal (desde nuestro directorio de trabajo).

```
git clone <url_de_nuestro_repositorio_en_GitHub>
```

Git nos solicitará nuestras credenciales, que serán nuestro usuario de la cuenta de GitHub y el token de acceso a dicho repositorio y tendremos que haber creado previamente.

Después de clonar el repositorio, verás que se crea un directorio con el nombre de tu repositorio:

```
$HOME/practicas/ppss-2025-Gx-apellido1-apellido2
```

Puedes comprobar que el nuevo directorio contiene a su vez un directorio oculto:

```
$HOME/practicas/ppss-2025-Gx-apellido1-apellido2/.git
```

El **directorio .git** contiene nuestro **repositorio local** Git (que está conectado con el repositorio remoto en GitHub). A partir de ahora, cualquier fichero que añadamos en el directorio *$HOME/practicas/ppss-2025-apellido1-apellido2/* podrá estar sujeto al control de versiones de Git.

GUARDAR LAS CREDENCIALES

Al intentar acceder a vuestro repositorio remoto desde el terminal, Git necesitará vuestras credenciales (login y PAT). Siempre que accedamos desde el terminal, usaremos el login de nuestra cuenta de GitHub, y un **token** (en lugar del **password** de nuestra cuenta de GitHub).

Una forma de evitar tener que introducir el *login* y *password* (token) cada vez que accedamos a nuestro repositorio remoto es utilizar el comando:

```
> git config credential.helper 'cache --timeout=3600'
```

Esta opción **almacena nuestras credenciales** de forma temporal en **memoria** durante un determinado periodo de tiempo. En este ejemplo establecemos que dicho almacenamiento tendrá lugar durante 3600 segundos (una hora). Por defecto el valor de "timeout" es de 900. (en cuyo caso el comando sería `git config credential.helper cache`). Podemos elegir el valor que más nos convenga.

Si queremos desactivar el temporizador antes de que finalice, utilizaremos el comando:

```
> git credential-cache exit
```

Otra opción es almacenar la contraseña (token) en un fichero de texto, pero debes tener en cuenta que se guarda en texto plano, con lo que puede leerse por cualquiera que acceda a vuestro ordenador.

Para ello usaremos el comando:

```
> git config --global credential.helper store
```

La siguiente vez que accedamos a GitHub (con *git push*, o *git pull*, por ejemplo). Nos pedirá el usuario y la contraseña (sólo la primera vez). Una vez introducidas se guardarán en el fichero ***.gitCredentials*** en nuestro HOME, de forma que las siguientes veces ya no nos volverá a preguntar.

FICHERO .gitignore

Nos situaremos en el directorio que contiene el fichero **.git** (nuestro repositorio local git). Este será, a partir de ahora, vuestro **directorio de trabajo**, en donde almacenaréis todo el trabajo de la asignatura.

Nuestro directorio de trabajo contiene el fichero oculto **`.gitignore`**, que hemos generado al crear el repositorio en GitHub

Se trata un archivo de texto en el que se indican todos aquellos ficheros y/o directorios que serán "ignorados" por git. Es decir, que no serán "guardados" en el repositorio remoto (ni en el local).

Necesitamos editar el contenido de **.gitignore**. Puedes hacerlo con cualquier editor de texto. Añadiremos las siguientes líneas (que mostramos a la derecha).

Las líneas que comienzan por # son comentarios. Ignoraremos los ficheros indicados relacionados con IntelliJ, además de los generados automáticamente por otros sistemas operativos como OSX o Windows.

```
# IntelliJ
.idea
*.iws
*.iml
*.ipr

# MacOS
*.DS_Store

# Windows
Thumbs.db
```

Cuando construyamos nuestros proyectos java con Maven, se creará sistemáticamente el directorio ***target***, que contendrá, entre otras cosas, los ficheros *.class* de nuestra aplicación. No tiene sentido guardar en GitHub información que podemos obtener en cualquier momento (volviendo a construir el proyecto), esto va a ocupar espacio en disco y consumir un tiempo innecesario (de subidas y descargas a/desde GitHub). Puedes comprobar que la primera línea del fichero **.gitignore** que acabas de editar es "target/".

CONFIGURAR GIT

A continuación vamos a **configurar Git**. Recuerda que este paso deberás hacerlo SIEMPRE cuando trabajes en los ordenadores del laboratorio.

comando git config

(ejecutar siempre en el laboratorio)

El comando git config nos permitirá guardar en el fichero .git/config algunos parámetros de configuración para trabajar con git. Ya lo hemos usado para guardar nuestras credenciales. Ahora lo usaremos para guardar nuestro nombre de usuario y *e-mail*. Cuando trabajes con tu ordenador, sólo será necesario hacerlo una vez (a menos que quieras cambiar de nombre de usuario y e-mail).

Configuraremos nuestra **identidad** con nuestro nombre y dirección de correo electrónico usando los siguientes comandos (desde la carpeta que contiene nuestro repositorio local, es decir, desde la carpeta que contiene el directorio .git):

> `git config user.name <nombreUsuario>`

Siendo `<nombreUsuario>` el nombre que mostrará Git cuando hagamos un *commit*. Poned vuestro nombre y apellidos. P.ej. "Luis Lopez Perez" (si usáis espacios tenéis que poner dobles comillas)

> `git config user.email <emailUsuario>`

Siendo `<emailUsuario>` el email del usuario en GitHub (no es necesario poner comillas)

comando git status

(para ver el estado de los ficheros)

El comando ***git status*** nos muestra el estado de los ficheros de nuestro directorio de trabajo (en **rojo** significa que el fichero todavía no está bajo el control de Git, o sea, que si lo borramos, lo perdemos).

Para que cualquier cambio realizado sobre un fichero de nuestro directorio de trabajo sea controlado por git, primero tenemos que "marcar" dicho fichero usando el comando git add. Y posteriormente guardaremos el fichero marcado en nuestro repositorio local con el comando ***git commit***. Para subir (copiar) nuestro repositorio local a GitHub usaremos el comando ***git push***.

Comandos para "guardar" el contenido del directorio de trabajo en nuestro repositorio local y en GitHub

(los ejecutaremos SIEMPRE)

```
> git add .
> git commit –m "Mensaje obligatorio"
> git push
```

IMPORTANTE: Siempre debes ejecutar los comandos git desde tu directorio de trabajo, es decir, desde el directorio que contiene el subdirectorio oculto .git

Si has hecho todo esto con **tu propio ordenador,** a partir de ahora, lo único que tendrás que hacer es utilizar los comandos git *add*, *commit* y *push* para sincronizar (en tu repositorio remoto) los cambios realizados en local (suponiendo que nunca trabajas con el ordenador del laboratorio). Recuerda que SIEMPRE deberás hacerlo desde $HOME/practicas/ppss-2025-Gx-apellido1-apellido2/ (aunque en cada práctica trabajemos en algún subdirectorio del mismo)

Si trabajas siempre utilizando los ordenadores del laboratorio, cuando llegues a casa y trabajes en tu ordenador, tendrás que (la primera vez):

Secuencia de trabajo en el ordenador de casa

(sólo la primera vez)

1. clonar el repositorio de GitHub (***git clone***)
2. configurar nuestra identidad (***git config***)
3. trabajar en el directorio de trabajo (***ppss-2025-Gx-apellido...***)
4. subir los cambios a GitHub (***git add + git commit + git push***)

Para las siguientes veces: supongamos que has trabajado desde los ordenadores del laboratorio, y luego quieres seguir trabajando en casa (y en casa ya habías clonado previamente el repositorio). En ese caso tendrás que hacer primero un ***git pull*** desde el ordenador de tu casa para sincronizar (y descargarte) los cambios que hiciste en el laboratorio antes de continuar trabajando:

Secuencia de trabajo en el ordenador de casa

(el resto de veces)

> **git pull**

2. trabajar en el directorio de trabajo (***ppss-Gx-2025-apellido...***)
3. subir los cambios a GitHub (***git add + git commit + git push***)

LICENCIA educacional IntelliJ

Para poder usar la versión Ultimate de IntelliJ necesitáis solicitar, cada uno de vosotros, una licencia educacional desde www.jetbrains.com (válida durante un año). Para ello tendréis que:

1. Crearos una cuenta. Para solicitar la licencia educacional necesariamente tendréis que proporcionar vuestro e-mail institucional. Recibiréis un correo con un enlace al que tendremos que acceder para confirmar la petición.
2. Una vez que entréis en vuestra cuenta, veréis una pantalla con el mensaje "No available Licenses", y varios enlaces. Seleccionamos el enlace *Apply for a free student or teacher license for educational purposes*. Rellenamos la petición indicando que sois estudiantes (lógicamente). Y a continuación recibiréis de nuevo un correo para activar la licencia educacional.

Una vez obtenida la licencia, ésta será necesaria para poder ejecutar IntelliJ. Si usáis los ordenadores de los laboratorios tendréis que activarla SIEMPRE, puesto que cuando apagáis la máquina NO se guarda ninguna información que hayáis introducido.

Para **activar la licencia** podemos hacerlo desde **IntelliJ** proporcionando nuestro e-mail y password de nuestra cuenta de JetBrains.

Configuración inicial de IntelliJ

Una vez que hemos accedido a la pantalla de bienvenida de IntelliJ, podemos cambiar el tema de la interfaz desde "***Customize→Color Theme***" (las opciones aparecen en la parte izquierda de la ventana) si no nos gusta el tema "Dark", que es el que aparece por defecto.

Vamos a **cambiar la ruta de maven**. Esto es importante porque vamos a usar la versión de maven que tenemos instalada, en lugar de usar la versión que viene con la instalación del IDE. Para ello, seleccionamos ***Customize → All Settings...→Build, Execution, Deployment→Build Tools→Maven***, en el cuadro de texto "***Maven home path***", y seleccionamos la ruta "***/usr/local/apache-maven-3.9.9*** (usando el icono con "...")

Adicionalmente, vamos a delegar en maven todas las acciones de construcción/ejecución del proyecto. Lo haremos desde ***Customize → All Settings...→Build, Execution, Deployment→Build Tools→Maven →Runner***, y desde aquí marcamos la casilla ***Delegate IDE building actions to Maven***

También vamos a ***instalar el plugin*** "***Maven test support***". Lo haremos seleccionando la opción ***Plugins***, y desde la pestaña ***Marketplace***, filtraremos por ejemplo por "maven". Una vez instalado el plugin tendremos que salir de IntelliJ y volver a entrar. Nos aseguraremos de que se ha instalado, accediendo de nuevo a ***Plugins***, y desde la pestaña ***Installed***, ahora nos aparecerá el plugin "*Maven test support*", y estará seleccionado.

Primeros pasos con Maven

Os hemos proporcionado, una carpeta **Plantillas-P00** con varios proyectos Maven. Maven es una herramienta de Construcción Automática de Proyectos. En un entorno profesional, es imprescindible utilizar alguna herramienta de construcción automática, y Maven es una de las opciones más usadas para Java.

Explicaremos con más detalle Maven en la sesión de teoría, pero la idea es comenzar a familiarizarnos con esta herramienta, ya que vamos a usarla durante todo el curso.

En esta sesión vamos a iniciarnos con Maven usando el TERMINAL. En sesiones posteriores también usaremos Maven, pero mayormente lo haremos desde IntelliJ, ya que necesitaremos modificar el código antes de la siguiente construcción del proyecto y es mucho más práctico tener todas las herramientas disponibles en una sola (el IDE) que ir alternando entre varias.

Crea la carpeta **P00-Maven** en tu directorio de trabajo (recuerda que tu directorio de trabajo es el que contiene la carpeta oculta .git) y copia las carpetas **P00-maven1**, y **P00-maven2** del directorio de plantillas en la carpeta P00-Maven.

Cada una de las dos carpetas que has copiado (P00-maven1 y P00-maven2) contienen un proyecto Maven.

Si no usas la máquina virtual y trabajas desde mac, abre el fichero **alias_tree.txt** y sigue las instrucciones para copiar la línea que contiene el script para emular el comando `tree` de linux.

ESTRUCTURA DE DIRECTORIOS DE UN PROYECTO MAVEN

Abre dos terminales y ejecuta el comando **`tree`** desde **P00-maven1** en uno de ellos, y desde **P00-maven2** en el otro. Tienes que comprobar que la estructura de directorios es idéntica en ambos casos (excepto por los paquetes java, que también se guardan como carpetas en el disco duro). TODOS los proyectos Maven tienen la misma estructura de directorios. Es importante que la memorices porque la usaremos en todas las prácticas en las que implementemos código. Copia las estructuras de directorios que has obtenido al ejecutar el comando **`tree`** en un fichero de texto con nombre **estructura_directorios_maven.txt** (en la carpeta **P00-Maven** de tu directorio de trabajo), y anota a la derecha de cada carpeta maven para el primer proyecto, cual es el propósito de dicha carpeta (si has tenido ya la clase de teoría, ya debes saberlo).

CONSTRUCCIÓN DEL PROYECTO

Ahora vamos a trabajar con el proyecto **P00-maven1**. Desde un terminal, sitúate en el directorio P00-maven1 y ejecuta el siguiente comando: **`mvn compile`**

> **`mvn`** es el comando que usamos para ejecutar Maven. IMPORTANTE: Este comando siempre hay que ejecutarlo desde la carpeta que contiene el fichero pom.xml.
>
> **`compile`** representa una FASE de un ciclo de vida Maven. Lo explicaremos en clase de teoría.

La ejecución del comando anterior desencadena la realización de una serie de acciones por parte de Maven. Observa que en pantalla se muestra en todo momento la traza de la ejecución de todas y cada una de las acciones realizadas por Maven. Es IMPORTANTE que tengas clara la **secuencia de acciones** realizadas al ejecutar un comando maven. Dicha secuencia de acciones representan una construcción del proyecto. Las acciones se muestran en pantalla precedidas de tres guiones **---**

Si todo ha ido bien, es decir, no se produce ningún error en ninguna de las acciones ejecutadas, entonces el proceso de construcción termina con un mensaje (lo verás con letras mayúsculas). ¿cuál es dicho mensaje? BUILD SUCCESS

Dependiendo del comando que usemos, provocaremos la ejecución de una secuencia de acciones u otra.

Vuelve a ejecutar el comando **`tree`** ¿qué ha cambiado? Ahora hay una carpeta Target ..

Ahora ejecuta el comando **`mvn clean`**, anota de nuevo las acciones realizadas por maven y vuelve a ejecutar el comando **`tree`**. De nuevo veras un cambio. ¿ya has averiguado qué hace este comando? Ya no hay target

Ejecuta el comando **`mvn test`** y compara la lista de acciones ejecutadas con las del comando **`mvn compile`**. Aunque la secuencia de acciones es diferente, en ambos casos, el proceso de construcción debe terminar mostrando el mismo mensaje, ya que no se va producir ningún error. En qué se diferencian ambas secuencias de acciones ejecutadas?

Vuelve a ejecutar **`tree`** y observa los cambios.

Ejecuta de nuevo **`mvn compile`**. ¿se ejecutan las mismas acciones que antes? ¿ves alguna diferencia?

Ahora vamos a cambiar de proyecto, y vamos a usar **P00-maven2**. Ejecuta el comando **`mvn compile`** que ya conoces y comprueba que se realizan las mismas acciones que antes. Comprueba también el mensaje que se muestra al finalizar la construcción.

Ejecuta el comando **`mvn test`**, y comprueba qué acciones se ejecutan. Verás que el mensaje mostrado al final del proceso de construcción es diferente porque tres de los tests han fallado.

Ahora, con un editor de textos, edita el fichero `Matricula.java`, y elimina el **`;`** del final de la línea 6.

Vuelve a ejecutar **`mvn test`** y verás que el proceso de construcción también resulta fallido, pero por una razón diferente. ¿por qué no se ejecutan los tests?

Vuelve a dejar el código como estaba, y ahora, de nuevo con un editor de textos, edita el fichero `MatriculaTest.java` y elimina el **`;`** del final de la línea 10. NO HAY PUTNO COMA EN LA 10. SIN HACER

Vuelve a ejecutar **`mvn test`** y explica lo que ocurre.

Es FUNDAMENTAL que tengas claro qué acciones se llevan a cabo (y en qué orden) cuando ejecutas cualquier comando maven. Y que sepas usar el comando maven adecuado para construir el proyecto en un momento dado. Para un determinado proyecto es posible que no se requieran ciertas acciones, por ejemplo, si no estamos desarrollando una aplicación web, no será necesario desplegar la aplicación en un servidor web.

Crea un fichero de texto con nombre **comandos_maven.txt** (en la carpeta **P00-Maven** de tu directorio de trabajo) y anota la secuencia de acciones que realiza maven para cada uno de los tres comandos que hemos practicado (cuando no se produce ningún error durante el proceso de construcción).

Finalmente sube tu trabajo de esta sesión a tu repositorio remoto en GitHub. Recuerda que tienes que ejecutar tres comandos:

```
> git add .
> git commit -m "Terminada la práctica P00-maven"
> git push
```

En las siguientes prácticas te aconsejamos que no dejes para el final el subir tu trabajo a GitHub (si tienes algún problema con la máquina virtual, o si trabajas en nativo y por ejemplo tienes algún problema con la escritura de los ficheros en el disco duro puedes perder todo el trabajo realizado). Es recomendable hacer "subidas" frecuentes para garantizar que nuestro trabajo está "a salvo" :)

Listado de programas que usaremos para la primera parte de la asignatura 

La razón de usar una máquina virtual es proporcionar el mismo entorno de trabajo para todos. De esta forma, ante cualquier problema con el software, lo solucionaremos una sola vez. Las prácticas que os proponemos están probadas en la máquina virtual (adicionalmente también hago la prueba en un mac (con procesador intel).

Si alguno de vosotros preferís trabajar en nativo, os proporcionamos la lista de programas y sus versiones, que hemos instalado en la máquina virtual.

Versión de linux: lubuntu 24.04.1 LTS (Noble Numbat) (https://lubuntu.me/downloads/)

Software que usaremos para la primera parte de la asignatura (sesiones S01..S06, P00..P06)	
openjdk 21.0.5 2024-10-15	
maven 3.9.9	
git 2.43.0	Podéis usar también versiones superiores a ésta
idealU-2024.3.1	Aunque haya actualizaciones disponibles, de momento en la máquina virtual no vamos a actualizar esta versión.