

Git: mooc_git-entrega1_merge-v2

Objetivo

Practicar con repositorios locales y remotos, creación de commits, además de la creación, uso e integración con merge de ramas de desarrollo.

Resumen de la práctica

Crear una cuenta en Github, si no se tiene. Crear un nuevo repositorio de nombre **my_calculator** en dicha cuenta, con un pequeño desarrollo de software. En la rama **master** se desarrollará en dos commits una calculadora con 2 botones (x^3 y x^4), un botón en cada commit. El primer commit debe incluir además un fichero README.md.

También se desarrollará otra versión de la calculadora en una rama lateral **sine**. Esta comenzará después del primer commit (calculadora con botón x^3) de la rama **master**. En esta rama se crea un commit que añade un nuevo botón $\sin(x)$, que calcule el seno del número.

A continuación se debe integrar con "git merge" la rama **master** en **sine** y luego **sine** en **master**.

Para terminar se debe subir el todo repositorio local, con sus dos ramas **master** y **sine**, a un repositorio vacío, creado en su cuenta de GitHub con el mismo nombre: **my_calculator**.

Prueba de la práctica

Para comprobar que la práctica ha sido realizada correctamente hay que utilizar el validador de este repositorio

https://github.com/practicas-ging/mooc_git-entrega1_merge

Para utilizar el proyecto validador, se debe instalar antes node.js (y npm) y por supuesto Git. El software y las instrucciones están accesibles en: <https://nodejs.org/es/>. Una vez instalados todos ellos, el proyecto se descarga, instala y ejecuta en el ordenador local con estos comandos:

```
$ ## El proyecto debe clonarse en el ordenador local
$ git clone https://github.com/practicas-ging/mooc_git-entrega1_merge
$
$ cd mooc_git-entrega1_merge ## Entrar en el directorio de trabajo
$
$ npm install ## Instala el programa de test
$
$ npm run checks [nombre_de_la_cuenta] ## Pasa los tests sobre el repositorio en github
..... ## indicando que partes están correctamente
..... ## implementadas y cuales no.
... (resultado de los tests)
$
```

Debe cambiar [nombre_de_la_cuenta] por el nombre de su cuenta en GitHub.

Instrucciones para la Entrega y Evaluación

Una vez finalizado el desarrollo debe entregar en MiriadaX solo el **nombre de su cuenta** en GitHub donde ha subido el repositorio my_calculator.

¡Cuidado! Compruebe que el nombre de la cuenta subido es el correcto y que el repositorio esta actualizado con la última versión.

El evaluador debe comprobar que la entrega es correcta buscando en GitHub el nombre de cuenta entregado y comprobando que contiene el repositorio pedido con las características solicitadas.

RUBRICA: La resolución de cada uno de estos puntos dará un el % indicado de la nota total:

- 10%: Existe el repositorio my_calculator
- 20%: El primer commit de la rama master es "x^3 button" y contiene lo pedido
- 20%: El segundo commit de la rama master es "x^4 button" y contiene lo pedido
- 20%: Existe la rama sine con el commit "sin(x) button" con el contenido y origen pedidos
- 30%: La integración de las ramas master y sine se ha realizado correctamente y la calculadora funciona bien.

El objetivo de este curso es sacar el máximo provecho al trabajo que están dedicando, por lo que les recomendamos que utilicen la evaluación para ayudar a sus compañeros enviando comentarios sobre la corrección del código, su claridad, legibilidad, estructuración y documentación.

Dado que es un curso para principiantes, ante la duda les pedimos que sean benevolentes con sus compañeros, porque muchos participantes están empezando y los primeros pasos siempre son difíciles.

¡Cuidado! Una vez enviada la evaluación, está no se puede cambiar. Piensen bien su evaluación antes de enviarla.

Pasos a seguir en el desarrollo de la práctica

Paso 1) Crear un repositorio de trabajo local vacío, llamado my_calculator, para realizar el desarrollo que se describe en los siguientes pasos.

Sugerencia de comandos a utilizar

\$ git init (Crea un repositorio git en el directorio actual)

También se puede utilizar

\$ git init my_calculator (Crea un nuevo directorio llamado my_calculator con su repositorio git)

Paso 2) Añadir al directorio de trabajo creado los dos ficheros siguientes:

1) Un primer fichero llamado README.md con el siguiente contenido:

"Mi primer fichero en mi primer repositorio, <nombre apellidos>"

2) Un segundo fichero de nombre calculator.html que implemente una calculadora con el botón de x^3 con el siguiente código:

```
=====
<!DOCTYPE html><html><head>
<title>Calculator</title><meta charset="utf-8">
<script type="text/javascript">

function cube() {
    var num = document.getElementById("n1");
    num.value = Math.pow(num.value, 3);
}
</script>
</head>
<body>
    <h1>Calculadora de .....su nombre y apellidos.....</h1>
    Number:
    <input type="text" id="n1">
    <p>
        <button onclick="cube()"> x^3 </button>
    </p>
</body>
</html>
=====
```

Debajo de la marca <body> debe rellenar en el título (entre marcas <h1>) su nombre y apellidos.

Sugerencia de comandos a utilizar además del editor:

```
$ cd mi_repo          # Para entrar en el directorio de trabajo
$ git status          # Para conocer el estado de los ficheros
$ git log --oneline    # Para mostrar que no hay commits en la historia (formato corto)
```

Paso 3) Registrar los ficheros en el índice y crear el primer commit en la rama master. Recordar que antes de crear un commit hay que probar siempre que el programa que se va a guardar funciona correctamente.

Sugerencia de comandos a utilizar:

```
$ git add <fichero1> <fichero2> ...
$ git commit          # abre el editor. Introducir entonces el msg x^3 button y
                      # cerrar editor para finalizar creación de commit "x^3 button".
```

También se pueden utilizar

```
$ git commit -m "x^3 button"
$ git status -s      # Para conocer el estado de los ficheros antes y después de crear commits
$ git log --oneline   # muestra la historia de commits en formato corto
```

Paso 4) Crear un segundo commit en la rama master. El commit debe añadir a la calculadora (fichero calculator.html) un segundo botón que eleve un número a la cuarta potencia (x^4).

Una vez añadido el código del nuevo nuevo botón a la calculadora y después de probar que funciona correctamente, registrar los cambios en el índice y crear el nuevo commit.

Sugerencia de comandos a utilizar:

```
$ git status -s      # muestra el estado de los ficheros respecto al índice
$ git add <fichero1> <fichero2> ... # añadir ficheros al índice
$ git commit          # abre el editor. Introducir entonces el msg x^4 button y
                      # cerrar editor para finalizar creación de commit "x^4 button".
```



Abre el editor automaticamente para añadir un mensaje 3

También se pueden utilizar

```
$ git commit -m "x^4 button" # cierra el commit, añadiéndole el mensaje indicado
$ git add . # registra todos los fich. nuevos o mod. en índice, Peligroso! usar con cuidado
$ git log --oneline # muestra la historia de commits en formato corto
$ git diff .... # muestra las diferencias con el commit anterior antes y después de cerrarlo
```

Paso 5) Crear una rama de nombre **sine** que salga del primer commit (con mensaje "x^3 button") y restaurar la rama **sine** en el directorio de trabajo, para poder trabajar sobre ella.

Sugerencia de comandos a utilizar:

```
$ git branch -v # muestra las ramas existentes en el repositorio
$ git checkout -b sine <commit_id> # Crea rama en commit y realiza checkout a rama
$ git log --oneline # historia de commits (incluye <commit_id>, p.e. 71e69ce)
```

Paso 6) Crear un nuevo commit en la rama **sine** que añada un segundo botón **sin(x)** a la calculadora en el fichero **calculator.html** (además del botón **x^3** que ya existe). El nuevo botón debe calcular el seno de un número utilizando la función JavaScript: **Math.sin(x)**.

→ Estando en la rama **sine**, se añade el código del nuevo nuevo botón a la calculadora y una vez comprobado que funciona correctamente, registrar los cambios en el índice y crear el nuevo commit.

Sugerencia de comandos a utilizar:

```
$ git branch -v # muestra las ramas existentes en el repositorio
$ git status -s # muestra el estado de los ficheros
$ git add <fichero1> <fichero2> ...
$ git commit -m "sin(x) button" # cierra el commit, añadiéndole el mensaje indicado
```

También se pueden utilizar

```
$ git branch -v # muestra las ramas existentes en el repositorio
$ git log --oneline # muestra la historia de commits en formato corto
$ git log --oneline --all # Para mostrar la historia de todas las ramas en formato corto
$ git diff .... # muestra las diferencias con el commit anterior antes y después de cerrarlo
$ git add . # registra todos los fich. nuevos o mod. en índice, Peligroso! usar con cuidado
```

los commit que hay en la rama master meterlo dentro de la rama sine

Paso 7) Integrar la rama **master** en la rama **sine** con el comando "git merge master", para crear una calculadora con tres botones: **x^3**, **x^4** y **sin(x)**. La integración tiene conflictos, que se deben resolver con el editor. Una vez resueltos y después de comprobar que el programa integrado funciona correctamente, se debe finalizar la integración (merge) creando el commit de integración con git commit.

Sugerencia de comandos de git a utilizar:

```
$ git merge master # ejecutar este comando estando en la rama sine, para integrar master
$ git merge --continue # continuar merge después de resolver conflictos e indexar cambios
$ git status -s # para ver los ficheros con conflictos despees de integrar las ramas
$ git add <fichero1> <fichero2> ... # para registrar ficheros modificados en el índice
$ git commit -m "integrate master" # para finalizar la integración cerrando el commit
```

También se pueden utilizar

```
$ git merge --abort # deshacer merge en curso y volver al estado anterior
$ git log --oneline --graph # Para mostrar la historia de integraciones de la rama
$ git log --oneline --graph --all # Para mostrar la historia de integraciones de todas las ramas
$ git diff xxx # para ver diferencias en el fichero xxx
$ git add . # registra todos los fich. nuevos o mod. en índice, Peligroso! usar con cuidado
```

Paso 8) Añadir un nuevo commit a la rama **master** que integre la calculadora con tres botones de la rama **sine**, es decir integrar con **merge** la rama **sine** en la rama **master**. Como el grafo de commits indica que la integración se ha realizado ya en la rama **sine**, git realiza la integración con **Fast-Forward**, reutilizando el commit ya generado en la integración del punto anterior y simplemente avanza el puntero de rama a dicho commit, sin crear uno nuevo.

Sugerencia de comandos de git a utilizar:

```
$ git merge sine # ejecutar este comando estando en la rama master
$ git log --oneline --graph --all # Para mostrar la historia de integraciones de todas las ramas
```

```
1 git merge sine
Updating 9fffc2c..b345bff
Fast-forward
 calculator.html | 66 +++++++++++++++++++++++++++++++++++++
 1 file changed, 41 insertions(+), 25 deletions(-)
```

→ **Paso 9)** Crear una cuenta en **GitHub**. Si ya tiene una cuenta puede utilizarla.

Paso 10) Crear en la cuenta de **GitHub** anterior un repositorio vacío de nombre **my_calculator**.


Paso 11) Subir todas las ramas del repositorio local al nuevo repositorio en **GitHub**.

Sugerencia de comandos a utilizar:

```
$ git push --all https://github.com/<su_cuenta>/my_calculator
```

La opción **--force** o **-f** permite subir un repositorio incompatible, pero ¡Cuidado borra el existente!

Partes de un archivo



Metadatos:

Los metadatos (*metadata*) son campos de texto que van incrustados en casi todos los tipos de archivos que añaden información adicional como la fecha de creación, resolución, tamaño, fecha de modificación, autor, etc.

Ejemplo de metadatos de un archivo de Word:

```
language - U.S. English
paragraph count - 11
line count - 40
title - SUMMARY
word count - 858
page count - 2
creator - Sales Account Company, S.L.
date - 2008-01-17T13:04:00Z
character count - 4891
generator - Microsoft Office Word
last saved by - Jimmy
creation date - 2008-01-10T10:31:00Z
template - Normal
```

¿Cómo ver los metadatos de un archivo? ▶ Opción **PROPIEDADES** del Menú Contextual de un archivo

SlidePlayer 9 / 13

Contenido de archivo y Metadatos - describe como es ese archivo

Anexo I: Algunas recomendaciones para UNIX/LINUX

Uso del terminal de comandos en UNIX/LINUX

Los comandos de **Git**, **node** o **npm** se deben ejecutar en un terminal de comandos. El terminal de comandos está atendido por un programa denominado **shell** (o variante), que suele indicar que esta preparado mostrando: **\$**. Un comando se ejecuta tecleando su nombre y parámetros asociados, seguidos de retorno de línea.

En UNIX/LINUX el terminal de comandos suele tener un icono asociado en algún lugar del escritorio. El terminal de comandos se abre al hacer clic en el icono.

Un terminal de comandos tiene siempre un **directorio de trabajo** (**working directory** o **wd**) asociado, que es el **directorio** sobre el cual se ejecutan los comandos. Por ejemplo

```
$ ls                ## lista ficheros del wd (directorio de trabajo)
$ ls dir1           ## lista ficheros del directorio dir1, contenido en wd
$ ls ../dir2        ## lista ficheros del directorio dir2, contenido en el directorio padre de wd
$ ls /home/eva/dir3 ## lista ficheros de identificado por la ruta absoluta dir3

$ nano fichero.ext  ## edita, si existe, o crea, si no existe, el fichero.js del directorio de trabajo
$ vi fichero.ext    ## edita, si existe, o crea, si no existe, el fichero.js del directorio de trabajo

$ pwd              ## muestra la ruta absoluta al directorio de trabajo del terminal de comandos

$ cd dir1          ## cambia el directorio de trabajo por el directorio dir1 contenido en él
$ cd ..            ## cambia el directorio de trabajo por su directorio padre (el primero en la ruta a la raíz)
```

El sistema de ficheros y directorios de UNIX/LINUX tiene estructura de árbol y los ficheros se identifican en los comandos con rutas absolutas, que comienzan en el **directorio raíz del árbol** (identificado con **/**), o relativas, que comienzan en el **wd** (directorio de trabajo). Las rutas son caminos en el árbol de directorios. Por ejemplo **dir1** y **../dir2** son rutas relativas a **wd**, mientras que **/home/eva/dir3** es una ruta absoluta que parte del directorio raíz que identifica el directorio **dir3**, contenido en el directorio **eva**, **eva** a su vez está contenido en el directorio **home**, que está contenido en el directorio raíz **/**.

Los ficheros y directorios de un ordenador pueden verse y modificarse de 2 formas:

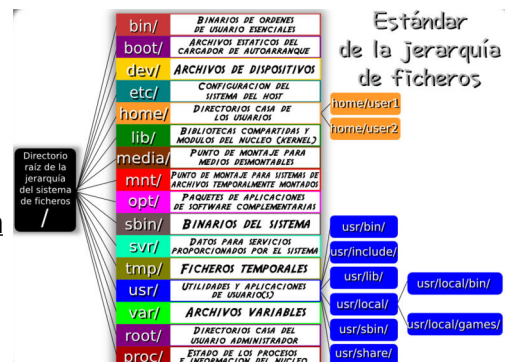
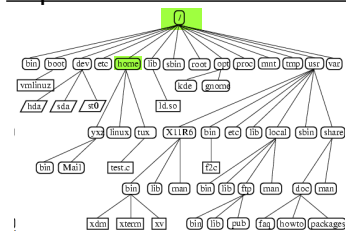
1. A través del terminal de comandos invocando comandos.
2. A través de ventanas con iconos que muestran gráficamente el contenido de un directorio. Los comandos se ejecutan al hacer clic con el ratón en el icono.

- Directorio actual .
- Directorio padre ..
- Directorio home ~
- Ej:
cp fich1 ..
cp fich1 ~
cp fich2 ~/datos

Ambas formas de ver y procesar ficheros son equivalentes y muestran la misma información. Por ejemplo: El comando **ls** muestra los mismos ficheros que la ventana con iconos asociada al mismo directorio. O si borramos o modificamos el nombre de un fichero con comandos del terminal de comandos, la ventana de iconos quitará el icono o mostrará el nuevo nombre.

Listado de Comandos UNIX/LINUX mas habituales:
<http://www.cheat-sheets.org/saved-copy/fwunixref.pdf>

Pequeño tutorial de de UNIX/LINUX:
<http://www.cheat-sheets.org/saved-copy/A.very.brief.guide.to.Linux.htm>



Edición de programas en UNIX/LINUX

Los ficheros que contienen programas deben editarse con editores de texto plano, tales como nano, vi o vim, notepad o notepad++, sublime text, ATOM, Brackets, ... Estos editores solo incluyen los caracteres que muestran y permiten que los programas editados con ellos sean ejecutados sin problemas.

¡Cuidado! Los editores de documentos como **Word o Pages** estropean el texto de un programa porque añaden caracteres especiales que impedirán que se pueda ejecutar. No utilizarlos nunca para editar programas ejecutables, porque los dejan inservibles.

Si ya está utilizando algún editor o IDE que permite editar texto plano y lo conoce bien, lo más recomendable es que lo siga utilizando. Sino le recomendamos uno de estos:

Sublime-text: editor de tipo wysiwyg potente y fácil de usar, existente para UNIX, Windows, ... Se puede descargar una versión de prueba para uso ilimitado de <https://www.sublimetext.com>.

ATOM: editor de tipo wysiwyg potente y fácil de usar, existente para UNIX, Windows, ... Tiene funciones de IDE y permite inspeccionar y gestionar repositorios Git. Es gratuito (patrocinado por GitHub) y se descarga de <https://atom.io/>.

nano: editor muy sencillo, auto-explicativo y fácil de aprender, aunque muy limitado. Está en todos los UNIX o Linux. Se invoca en modo comando:

```
$ nano fichero.ext    ## abre para edición, si existe, o crea, si no existe, el fichero.ext en wd
```

vi o vim: vi es el editor tradicional existente de UNIX y está en todos los UNIX o Linux. Es el editor que suele abrir Git por defecto. Es potente, pero orientado a comando y requiere aprender los comandos. vim es una evolución de vi. Se invoca en modo comando:

```
$ vi fichero.ext      ## abre para edición, si existe, o crea, si no existe, el fichero.ext en wd
```

Lista de comandos:

http://www.atmos.albany.edu/daes/atmclasses/atm350/vi_cheat_sheet.pdf

Además están los editores de los IDEs (Integrated Development Enviroments) mas profesionales. Son muy convenientes, porque permiten tanto editar programas, como realizar muchas tareas de gestión de un proyecto software. Destacan **Visual Studio** (<https://visualstudio.microsoft.com/es/>), que es gratuito, o **Webstorm** (<https://www.jetbrains.com/webstorm/>), que es de pago pero da licencias gratuitas para actividades educativas de profesores y estudiantes (<https://www.jetbrains.com/student/>).

Anexo II: Algunas recomendaciones para Windows

Uso del terminal de comandos en Windows

Los comandos de Git, node o npm se deben ejecutar en un terminal de comandos. El terminal de comandos está atendido por un programa, que suele indicar que esta preparado mostrando: **>**. Un comando se ejecuta tecleando su nombre y parámetros asociados, seguidos de retorno de línea.

En Windows hay que hacer clic en el icono de Windows y teclear “powershell” seguido de retorno de línea para abrir el terminal de comandos. Alternativamente se puede utilizar “cmd” (terminal por defecto anterior) o “git bash” (terminal que nos provee Git cuando lo instalamos en Windows).

Un terminal de comandos tiene siempre un directorio de trabajo (working directory o wd) asociado, que es el directorio sobre el cual se ejecutan los comandos. Por ejemplo en Windows

```
> dir                ## lista ficheros del wd. ls funciona también en “powershell”, pero no en “cmd”
> dir dir1           ## lista ficheros del directorio dir1, contenido en wd
> dir ..\dir2        ## lista ficheros del directorio dir2, contenido en el directorio padre de wd
> dir /home/eva/dir3 ## lista ficheros de identificado por la ruta absoluta dir3

> pwd               ## muestra la ruta absoluta al directorio de trabajo del terminal de comandos

> cd dir1           ## cambia el directorio de trabajo por el directorio dir1 contenido en él
> cd ..             ## cambia el directorio de trabajo por su directorio padre (el primero en la ruta a la raíz)
```

Las últimas versiones de Windows suelen aceptar también comandos de UNIX.

El sistema de ficheros y directorios de Windows tiene estructura de árbol y los ficheros se **identifican** en los **comandos con rutas absolutas**, que **comienzan** en el **directorio raíz del árbol** (identificado con **C:** o ****), o **relativas**, que comienzan en el **wd (directorio de trabajo)**. Las rutas son caminos en el árbol de directorios. Por ejemplo **dir1** y **..\dir2** son rutas relativas a wd, mientras que **C:\home\eva\dir3** o **\home\eva\dir3** es una **ruta absoluta** que parte del directorio raíz que identifica el directorio **dir3**, contenido en el directorio **eva**. **eva** a su vez está contenido en el directorio **home**, que está contenido en el directorio raíz **C:** o ****.

Los ficheros y directorios de un ordenador pueden verse y modificarse de 2 formas:

1. A través del terminal de comandos invocando comandos.
2. A través de ventanas con iconos que muestran gráficamente el contenido de un directorio. Los comandos se ejecutan al hacer clic con el ratón en el icono.

Ambas formas de ver y procesar ficheros son equivalentes y muestran la misma información. Por ejemplo: El comando **dir** muestra los mismos ficheros que la ventana con iconos asociada al mismo directorio. O si borramos o modificamos el nombre de un fichero con comandos del terminal de comandos, la ventana de iconos quitará el icono o mostrará el nuevo nombre.

Listado de comandos Windows mas habituales:

<http://simplyadvanced.net/blog/cheat-sheet-for-windows-command-prompt/>

Edición de programas en Windows

Los ficheros que contienen programas deben editarse con editores de texto plano, tales como nano, vi o vim, notepad o notepad++, sublime text, ATOM, Brackets, ... Estos editores solo incluyen los caracteres que muestran y permiten que los programas editados con ellos sean ejecutados sin problemas.

¡Cuidado! Los editores de documentos como **Word** estropean el texto de un programa porque añaden caracteres especiales que impedirán que se pueda ejecutar. No utilizarlos nunca para editar programas ejecutables, porque los dejan inservibles.

Si ya está utilizando algún editor o IDE que permite editar texto plano y lo conoce bien, lo más recomendable es que lo siga utilizando. Sino le recomendamos uno de estos:

Sublime-text: editor de tipo wysiwyg potente y fácil de usar, existente para UNIX, Windows, ... Se puede descargar una versión de prueba para uso ilimitado de <https://www.sublimetext.com>.

ATOM: editor de tipo wysiwyg potente y fácil de usar, existente para UNIX, Windows, ... Tiene funciones de IDE y permite inspeccionar y gestionar repositorios Git. Es gratuito (patrocinado por GitHub) y se descarga de <https://atom.io/>.

Notepad: es el editor wysiwyg de texto plano de windows. Es recomendable instalar **Notepad++** que soporta análisis sintáctico del lenguaje de programación y se puede descargar de <https://notepad-plus-plus.org/>.

vi o vim: vi es el editor tradicional existente de UNIX y en Windows se instala cuando se instala Git, pero está accesible solo en el terminal de comandos "git bash". Además es el editor que suele abrir Git por defecto. Es potente, pero orientado a comando y requiere aprender los comandos. vim es una evolución de vi. Se invoca en modo comando:

```
$ vi fichero.ext      ## abre para edición, si existe, o crea, si no existe, el fichero.ext en wd
```

Lista de comandos:

http://www.atmos.albany.edu/daes/atmclasses/atm350/vi_cheat_sheet.pdf

Además están los editores de los IDEs (Integrated Development Enviroments) mas profesionales. Son muy convenientes, porque permiten tanto editar programas, como realizar muchas tareas de gestión de un proyecto software. Destacan **Visual Studio** (<https://visualstudio.microsoft.com/es/>), que es gratuito, o **Webstorm** (<https://www.jetbrains.com/webstorm/>), que es de pago pero da licencias gratuitas para actividades educativas de profesores y estudiantes (<https://www.jetbrains.com/student/>).