Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

Práctica 1a:

Uso de git

1. Objetivos

- Aprender a usar **git** de manera individual y de forma local
- Aprender a crear repositorios, commits, tags, etc.
- Deshacer acciones
- Aprender a crear ramas y trabajar con ellas
- Realizar nuestro primer programa en C#

2. Requisitos técnicos

Sigue los pasos indicados, **respeta el uso de mayúsculas y minúsculas** así como el **nombre de** las **carpetas, archivos, clases y métodos** que se te indique que has de crear. Requisitos que tiene que cumplir este trabajo práctico para ser evaluado (si no se cumple alguno de los requisitos la calificación será **cero**) son:

- La solución entregada no contiene archivos compilados, por lo que, **antes de comprimir, debes limpiar la solución (Compilar > Limpiar solución).**
- El archivo entregado se llama `hada-p1.zip` (**todo en minúsculas**).
- Al descomprimir el archivo `hada-p1.zip` se crea un directorio de nombre `hada-p1` (**todo en minúsculas**), el cual contiene a su vez otro directorio de nombre `hada-p1` (**todo en minúsculas**).
- El directorio de la solución `hada-p1` contiene la copia de trabajo y el directorio **`.git`**.
- Los archivos que hay dentro del directorio del proyecto `hada-p1` se llaman como se indica en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).
- Las etiquetas, nombres de ramas, clases C# y métodos implementados se llaman como se indica en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).

3. Guía de evaluación

Esta práctica contará un 1.25% de la nota final.

La práctica se puntuará de acuerdo a los siguientes criterios:

- Los distintos *commits* pedidos a lo largo del enunciado supondrán hasta el 25% de la nota.
- La creación correcta de las etiquetas pedidas supondrá hasta el 15% de la nota.
- La creación y trabajo correctos con las ramas pedidas supondrá hasta el 50% de la

nota.
- La correcta contestación a las preguntas marcadas en negrita **(P1-P4)**, que se incluirán en un archivo readme.md (más detalles abajo), supone un 10% de la nota.

4. Entrega

Se entregará el directorio de la solución `hada-p1`, junto con todo su contenido, comprimido en un fichero llamado `hada-p1.zip`.
La entrega se realizará en http://pracdlsi.dlsi.ua.es, no se admite ningún otro método.
Fecha límite: 16/02/2020

5. Descripción

En esta primera práctica vamos a hacer uso del sistema de control de versiones **Git**.

Parte 0: primeros pasos con git

Vamos a trabajar desde consola. Puedes utilizar la consola de windows (busca cmd) o la consola de git (busca git cmd - para una consola windows - o git bash - para una consola con comandos unix). Nosotros te recomendamos utilizar la consola de git.

Comprueba la versión instalada:

```
$git --version
```

Parte 1: crear nuestro primer repositorio y añadir archivos

Crea una solución llamada **`hada-p1`** que contenga un proyecto de tipo "Aplicación de consola (.NET Framework)" con el mismo nombre y dentro de éste crea un archivo C# llamado **`HadaP1.cs`** el cual tendrá una clase llamada **HadaP1** (Agregar > Nuevo Elemento > Clase).

En la consola, muévete al directorio de trabajo (la carpeta que contiene la solución) e inicializa el repositorio git.

```
$cd C:\VisualStudioProjects\hada-p1
```

```
$git init
```

Añádele el archivo `HadaP1.cs` sólo con la clase, sin ningún método.

```
$git add hada-p1/HadaP1.cs
```

Comprobamos el estado después de añadirlo.

```
$git status
```

Se te indica que hay cambios pendientes, haz el *commit* correspondiente

```
$git commit -m "Added HadaP1 class"
```

Confirma que no hay más cambios preparados para hacer commit con git status. Verás que hay archivos que no se han añadido al repositorio, inclúyelos con los commits correspondientes (uno por archivo). Recuerda que no es conveniente subir todos los archivos, sólo aquellos que no se puedan generar a partir de otros ya incluidos (extensiones a ignorar .csproj y .sln).

Puedes comprobar qué archivos están bajo el control de versiones, porque los has añadido al repositorio, así:

```
$git ls-files
```

Parte 2: modificar y renombrar archivos

Queremos crear un conversor de unidades, para ello en la clase **`HadaP1.cs`** añade un método con una interfaz como esta:

```
public static double Seconds2Minutes(double s)
```

Como su nombre indica, dado una número "s" de segundos nos devolverá su equivalente en minutos (1 minuto = 60 segundos). Haz el *commit* correspondiente para incluir este método.

Modifica el método anterior para que compruebe si el número "s" es cero, en ese caso que devuelva cero directamente. Haz el *commit* correspondiente que recoge esta modificación.

Crea un archivo autores.txt para incluir tu nombre (Agregar > Nuevo Elemento > Elemento de Visual c# > Archivo de texto). Haz el *commit* correspondiente para incluir este archivo bajo control de versiones.

Renombra el archivo autores.txt a readme.txt

```
$git mv hada-p1/autores.txt hada-p1/readme.txt
```

Ahora para que Visual Studio tenga en cuenta el nuevo archivo llamado readme.txt tendríamos que excluir del proyecto (Botón derecho > Excluir del proyecto) el antiguo autores.txt y añadir readme.txt (Agregar > Elemento Existente > readme.txt).

Figura 1. Excluir del proyecto archivo autores.txt.

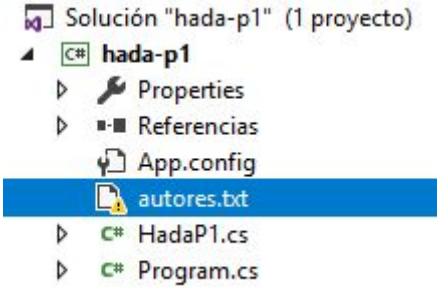
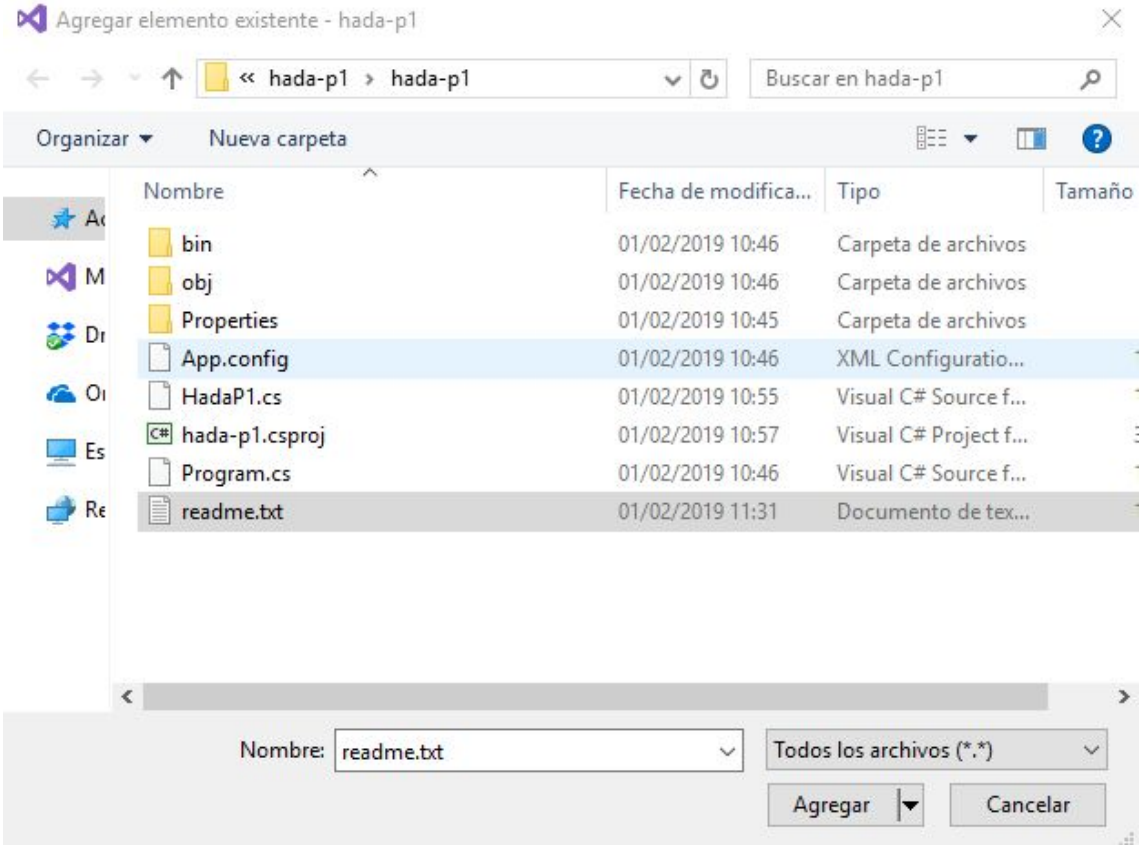


Figura 2.Añadir al proyecto archivo readme.txt.



¿Qué habría que hacer ahora? Utiliza el comando `git status` para saber si es necesario hacer un nuevo *commit* que refleje este cambio.

Parte 3: borrar archivos y vuelta atrás

Decidimos que no queremos el archivo readme.txt y lo eliminamos

```
$git rm hada-p1/readme.txt
```

Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

Haz el *commit* correspondiente para eliminar este archivo del control de versiones.

Te das cuenta que realmente no lo querías borrar, solo querías cambiar la extensión de `txt` a `md` e incluir el título de la práctica. ¿Cómo podemos recuperar este archivo? Lo primero es consultar el log del repositorio

```
$git log
```

Este comando nos devuelve todos los commits que hemos hecho en el repositorio. Cada uno tiene un número SHA-1 propio que permite distinguir cada commit realizado. Verás que el primero de esta lista es el último que *commit* realizado, donde se había eliminado el archivo.

```
$git log
commit    8ecb5b2606bd5f1d0ec32668c0bea238a5734a4f    (HEAD    ->
master)
Author: hada <hada@dlsi.ua.es>
Date:   Sat Nov 10 19:41:39 2018 +0100
      remove readme.txt
commit 7f31764a387ac259dabfd6ec29a42d110de0fae3
Author: hada <hada@dlsi.ua.es>
Date:   Sat Nov 10 19:39:56 2018 +0100
      rename autores.txt to readme.txt
:... skipping ...
```

Para borrar este commit y recuperar el archivo eliminado, puedes utilizar el comando `revert`, para ello le indicas el commit a borrar. En lugar de utilizar el código SHA-1, puedes utilizar el alias `HEAD`:

```
$git revert -n HEAD
```

¿Qué habría que hacer ahora? Utiliza el comando `git status` para saber si es necesario hacer un nuevo *commit* que refleje este cambio.

P1. ¿Qué crees que significa la opción -n en el comando revert? Pista: Usa la ayuda del comando (`$git revert -help`). *Recuerda: las preguntas se contestarán en el archivo readme que se encuentra actualmente en tu repositorio.*

Vuelve a eliminar el archivo e intenta solucionar el problema anterior con el comando `git reset --hard`.

P2. ¿Qué cambios se han producido al ejecutar el comando `reset`?

P3. ¿Qué diferencias observas entre `revert` y `reset`?

Recuerda cambiar la extensión de `txt` a `md` e incluir el título de la práctica.

Parte 4: trabajar con etiquetas y ramas

Antes de continuar implementando nuestro conversor, queremos etiquetar la versión

actual (`HEAD)` como la versión 0.1:

```
$ git tag v0.1 HEAD
```

Cada etiqueta marca un punto específico de la historia como importante, normalmente se corresponden con versiones. En nuestro caso marca la inclusión del proyecto inicial y la conversión de de segundos a minutos.

Para listar las etiquetas disponibles:

```
$ git tag -l
```

Para visualizar todo esto gráficamente utiliza: `$ gitk`

Todo repositorio tiene, mínimo, una rama llamada `master` por defecto. Es una buena práctica dejar en la rama `master` la última versión estable de tu programa y tener una rama `devel` sobre la que ir trabajando en nuevas funcionalidades.

Para ver las ramas disponibles en local:

```
$git branch
```

A partir del último commit de la rama `master`, crea ahora una nueva rama llamada `devel` y cámbiate a ella.

```
$git branch devel
```

```
$git checkout devel
```

Puedes comprobar que estás en la rama `devel` con el siguiente comando:

```
$ git branch
* devel
  master
```

Modifica la clase **`HadaP1.cs`** para añadir un método como una interfaz como esta:

```
public static double Minutes2Seconds(double m)
```

Como su nombre indica, dado una número "m" de minutos nos devolverá su equivalente en segundos (1 minuto = 60 segundos). Haz el *commit* correspondiente para incluir este método y crea una etiqueta con versión v0.2.

Parte 5: combinar ramas

Ahora tienes dos ramas, para saber cuáles son las diferencias entre ellas a nivel de archivo utiliza el comando:

```
$ git diff devel master
```

También podemos ver las diferencias a nivel de commits con git log e indicando la rama origen y la destino:

```
$ git log ramaorigen..ramadestino
```

Por ejemplo, para saber cuáles son los commits incluidos en devel y no en master: `$ git log master..devel`

```
commit 7caad75792e599ff353245e1692b291d14a06e9c (HEAD -> devel)
Author: hada <hada@dlsi.ua.es>
Date:   Sat Nov 10 21:03:03 2018 +0100
      add method that converts minutes to seconds
```

Otra opción es utilizar el comando `show-branch`:

```
$ git show-branch master devel
! [master] rename readme.txt to readme.md
 * [devel] add method that converts minutes to seconds
--
 * [devel] add method that converts minutes to seconds
+* [master] rename readme.txt to readme.md
```

Ahora queremos mezclar los cambios de ambas ramas. Primero debes cambiarte a la rama `master`.

```
$ git checkout master
```

Tenemos dos opciones: mezclarlas o rebasarlas. La más sencilla es mezclarlas, para eso usamos el comando `merge` e indicamos la rama:

```
$ git merge devel
Updating ac0928d..7caad75
Fast-forward
 hada-p1/HadaP1.cs | 8 ++++++++
 1 file changed, 8 insertions(+)
```

A partir de la rama `devel`, crea una nueva llamada `devel-usuario` y cambiate a ella. Modifica el archivo Program.cs para que le pregunte al usuario la unidad de la que parte (segundos o minutos) y la cantidad. Muéstralo lo por pantalla y realiza la conversión utilizando las funciones disponibles en **`HadaP1.cs`**. Confirma los cambios de Program.cs.

Algunas pistas: utiliza la función `Console.WriteLine` para mostrar mensajes por pantalla y `Console.ReadLine` para leer la respuesta del usuario. Para convertir la respuesta (un string) a double, utiliza la función `double.Parse`. Por último, es recomendable emplear la estructura do-while para preguntarle al usuario si quiere realizar más conversiones, además, así evitarás que se cierre la consola.

Fusiona la rama `devel-usuario` con `master` y crea una etiqueta de la versión v0.3. Esta vez, en lugar de mezclar las ramas, las rebasarás, utilizando el comando `rebase` desde la rama master e indicando la rama:

```
$ git rebase devel-usuario
```

P4. ¿Qué diferencias observas entre `merge` y `rebase`?

Referencias:

[1] https://git-scm.com/book/es/v2
[2] http://ndpsoftware.com/git-cheatsheet.html
[3] https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Reorganizar-el-Trabajo-Realizado