Ejercicios nivel intermedio de Git

Para hacer este guión de prácticas vuelve a ponerte en pareja. Hasta que no se te indique seguirás trabajando en local, una vez que esté todo preparado trabajaremos con el servidor "github".

0. *Directorio de trabajo*

Al igual que en la práctica de ejercicios básicos con git, crearemos la carpeta "home/usuario/GIT/Intermedio". Sitúate dentro de la carpeta "Intermedio" para realizar esta lista de ejercicios con git.

1. *Preparación del entorno*

En "Intermedio" crea un fichero "READMEApellido1(alumno1)_Apellido1(alumno2)" que contenga "Pareja de prácticas formada por *alumno1* y *alumno2*". Crea un repositorio git de este directorio, añade y guarda el fichero (mensaje: "Entorno de trabajo").

2. *Creando una rama*

Crea una rama que se llame "*Apellido1(alumno1)_Apellido1(alumno2)*" *(haced que el alumno1 y el alumno2 aparezca en el mismo orden para ambos*[1]*)* y tras esto comprueba las ramas que existen. ¿Qué significa el * situado al lado de master? Ejecuta "`git checkout Apellido1(alumno1)_Apellido1(alumno2)`" (para navegar entre ramas) y luego vuelve a comprobar las ramas que existen ¿Qué ha ocurrido? ¿Existe algún comando para crear una rama e inmediatamente estar situado en dicha rama, cuál es? (busca en la ayuda checkout).

3. *Navegando y haciendo cambios en las ramas (solo alumno1)*

Estando en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" crea un fichero que se llame "README" que contenga: "Este es el entorno de trabajo de *Alumno1* y *Alumno2*, en el que se almacenarán los ficheros de la clase de prácticas de la asignatura Evolución del Software". Añade y guarda el fichero en el repositorio (mensaje: "README añadido"). Del mismo modo añade y guarda en esta rama el fichero pb.c del campus virtual (correspondiente a este guión de prácticas) (mensaje: "Fichero pb.c añadido"). Comprueba el contenido de las ramas.

4. *¿Qué ha sido cambiado?*

Cuando queremos visualizar los cambios que se han realizado escribimos git diff, esta opción nos compara lo que está en el área de preparación y qué se hizo en el último commit. Pero si queremos visualizar qué ficheros han sido modificados en el último commit, podemos usar las diferentes opciones de log o usar "`git whatchanged`". Busca en la ayuda de este comando para visualizar los últimos cambios realizados de tal modo que

[1] Para que luego no aparezcan repetidos en el repositorio los mismos alumnos.

aparezca la siguiente información: sha1, autor, el que hizo el commit, un título y el mensaje del commit.

5. *Pull y Push, trabajando con un repositorio remoto (alumno1)*

El alumno1 es el que va a realizar un push de la rama *"Apellido1(alumno1)_Apellido1(alumno2)"* al repositorio Github de la asignatura: https://github.com/lorgut/EvolSoft.git. Antes de esto hay que descargar el contenido de la rama **master** del repositorio remoto con el comando "`git pull <servidor_remoto> <rama>`", para utilizar esta orden, nos tenemos que situar en la rama master, ya que sólo le estamos indicando la rama de la cual nos vamos a traer los cambios (recuérdalo para el resto de la práctica), en caso contrario, tendremos que especificar la rama origen y destino. ¿Qué ha ocurrido? Observa el contenido de tu rama master. Cámbiate a la rama *"Apellido1(alumno1)_Apellido1(alumno2)"*, y haz un push al servidor de la rama. El formato del comando es exactamente el mismo pero indicando la rama que queremos subir al servidor.

6. *Actualizando y modificando directorio de trabajo*

El alumno2 se coloca en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" y la actualiza con el contenido de la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" del repositorio remoto: https://github.com/lorgut/EvolSoft.git. Comprueba el contenido de tu carpeta. Abre el fichero pb.c y realiza la siguiente modificación:

```
int main(int argc, char** argv)
{
    int balls[6];
    int count_balls = 0;
    int favorite = 0;

    for (int i=1; i<argc; i++)
    {
        const char* arg = argv[i];

        if ('-' == arg[0])
        {
            if (0 == strcmp(arg, "-favorite"))
            {
                favorite = 1;
            }
            else
            {
                goto usage_error;
            }
        }
        else
        {
            char* endptr = NULL;
            long val = strtol(arg, &endptr, 10);
            if (*endptr)
            {
                goto usage_error;
            }
            balls[count_balls++] = (int) val;
        }
```

```
    }

    if (6 != count_balls)
    {
        goto usage_error;
    }

    int power_ball = balls[5];

    int result = calculate_result(balls, power_ball);

    if (result < 0)
    {
        goto usage_error;
    }

    if (LUCKY_NUMBER == power_ball)
    {
        result = result * 2;
    }

    if (favorite)
    {
        result = result * 2;
    }

    printf("%d percent chance of winning\n", result);

    return 0;

usage_error:
    fprintf(stderr, "Usage: %s [-favorite] (5 white balls)
power_ball\n", argv[0]);
    return -1;
}
```

Guarda los cambios en el repositorio (mensaje: "Añadiendo argumento -favorite"). Tras esto crea un makefile para compilar el fichero con la instrucción: "`gcc -std=c99 -Wall -Wextra -Werror pb.c -o pb`". Añádelo al repositorio y guarda los cambios de tu directorio de trabajo (mensaje: "Makefile para compilar pb.c"). Tras esto realiza un push en la rama en la que estáis trabajando el alumno1 y tú.

7. *Más modificaciones con ramas*

El alumno1 actualiza su rama "*Apellido1(alumno1)_Apellido1(alumno2)*". Tras esto crea una nueva rama que se llama "*RamaAlumno1*", abre el fichero pb.c de la nueva rama y realiza los siguientes cambios:

```
#define MAX_WHITE_BALL 59
#define MAX_POWER_BALL 39

 static int my_sort_func(const void* p1, const void* p2)
 {
     int v1 = *((int *) p1);
     int v2 = *((int *) p2);

     if (v1 < v2)
```

```
    {
        return -1;
    }
    else if (v1 > v2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int calculate_result(int white_balls[5], int power_ball)
{
    for (int i=0; i<5; i++)
    {
        ...
        return -1;
    }

    qsort(white_balls, 5, sizeof(int), my_sort_func);

    return 0;
}
```

Guarda los cambios (mensaje: "Función My_sort_func") y realiza un push de la rama "*RamaAlumno1*" al repositorio github.

8. *Merge (sin conflictos)*

Actualmente en "*RamaAlumno1*" está la última versión del programa pb.c, donde el alumno1 ha añadido la función "my_sort_func". El alumno2, para seguir con la última versión, y no modificar el contenido del fichero del alumno1, va a fusionar ambos ficheros y luego seguirá trabajando con él. Para ello tendrá que crear (al igual que hizo el alumno1) una rama llamada "*RamaAlumno1*" estando en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" y actualizar su contenido de la misma rama del repositorio. Una vez actualizada la rama "*RamaAlumno1*", comprueba que efectivamente en el fichero pb.c está la función "my_sort_func". El alumno2 situado en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" realiza un merge (una fusión) de la rama "*RamaAlumno1*" a la rama en la que se encuentra actualmente. Comprueba que se ha fusionado correctamente (no debería de dar conflicto) y realiza los siguientes cambios:

```
    int balls[6];
    int count_balls = 0;
    int favorite = 0;  // this should be a bool

    for (int i=1; i<argc; i++)
    {
        goto usage_error;
    }

    // the power ball is always the last one given
    int power_ball = balls[5];

    int result = calculate_result(balls, power_ball);
```

```
      // calculate result can return -1 if the ball numbers
      // are out of range


      if (result < 0)
      {
          goto usage_error;
[…]
```

En calculate_result():

```
[…]
       return -1;
      }
     }

      // lottery ball numbers are always shown sorted
      qsort(white_balls, 5, sizeof(int), my_sort_func);

      return 0;
}
```

Guarda los cambios (mensaje: "Comentarios") y haz un push a la rama compartida "*Apellido1(alumno1)_Apellido1(alumno2)*".

9. *Merge (sin conflictos) El alumno2 sigue con el siguiente ejercicio*

El alumno1 actualiza la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" con los cambios realizados por el alumno2. Realiza un merge a la rama "*RamaAlumno1"* ya que los comentarios realizados por el alumno2 son interesantes para el código. Tras la fusión se dispone a cambiar la variable "favorite" a tipo bool. Para ello, además de cambiar el tipo de la variable y asignarle los valores adecuados, añade la librería `<stdbool.h>`. Guarda los cambios (mensaje: "Usando el tipo bool). Finalmente realiza un push en la rama "*RamaAlumno1"*.

10. Merge (conflictos)

El alumno2 trabaja en la última versión de pb.c de la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" y cambia a inglés británico la palabra *favorite* a *favourite* en todas las ocurrencias del código. (mensaje: "Arreglando error ortográfico a inglés británico"). Realiza un push en su rama "*Apellido1(alumno1)_Apellido1(alumno2)*".

El alumno1 actualiza la versión de la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" e intenta hacer un merge de los cambios realizados por el alumno2 a su rama "*RamaAlumno1"* (saldrá un mensaje de conflicto). Intenta resolver el conflicto con `git mergetool`, escogiendo la aplicación `tkdiff`. Si hay algo que no puedas solventar, tendrás que hacerlo de forma manual (abriendo el editor).

Una vez resueltos los conflictos, añade de nuevo el fichero pb.c, realiza un commit (mensaje: "Arreglados los conflictos tras la fusión") y haz un push en la rama "*RamaAlumno1*".

11. Merge y ¿cuántos commits has hecho tú más que yo?

Es ahora cuando el alumno2 actualiza la rama "*RamaAlumno1"* y realiza una fusión de ésta con la rama "*Apellido1(alumno1)_Apellido1(alumno2)*". Añade un comentario al final de la función "calculate_result":

```
    return -1;
    }
   }

    // lottery ball numbers are always shown sorted
    qsort(white_balls, 5, sizeof(int), my_sort_func);
    // Here should be typed a function to calculate the probability
    return 0;
```

Añade, guarda los cambios (mensaje: "Comentario sobre la función de probabilidad") y haz un push en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*". Ahora el alumno2 siente la curiosidad por saber la cantidad de commits que han estado haciendo ambos (el alumno1 también si quiere puede sentir esa curiosidad), para ello está la orden "`git shortlog`", acude a la ayuda para saber qué opciones tienes que usar para obtener una salida como esta (obviamente solo estaréis el alumno1 y tú):

```
$ git shortlog <opciones>
  135  Tom Preston-Werner
  15  Jack Danger Canty
  10  Chris Van Pelt
  7  Mark Reid
  6  remi
  3  Mikael Lind
  3  Toby DiPasquale
  2  Aristotle Pagaltzis
  2  Basil Shkara
  2  John Reilly
  2  PJ Hyett
  1  Marc Chung
  1  Nick Gerakines
  1  Nick Quaranto
  1  Tom Kirchner
```

12. *Eliminando ramas remotas*

El alumno1 viendo que todo el código está concentrado en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*", decide que es mejor borrar la rama "*RamaAlumno1*". Antes de realizar el borrado, visualiza las ramas locales y luego las ramas remotas. Para las ramas remotas tiene que hacer uso del siguiente comando: "`git ls-remote --heads servidor_remoto`". Borra la rama del repositorio y vuelve a comprobar las ramas locales y las ramas remotas. ¿Qué comando usarías para eliminar las ramas locales? (todavía no las borres, solo comenta cuál sería)

13. ¿Qué commits no son de esta rama?

Tanto el alumno1 como el alumno2 comprueban en sus entornos de trabajo qué commits **no** se han realizado en la rama master, en la rama "*Apellido1(alumno1)_Apellido1(alumno2)*" y en la rama "*RamaAlumno1"*. ¿Cómo lo haríais? Busca en la página web la orden `cherry`.

14. Información sobre nuestro entorno de trabajo

Para ver la información sobre el entorno de trabajo con respecto el repositorio con el que está trabajando tanto el alumno1 como el alumno2, ejecutan la siguiente orden: `git remote show <servidor_remoto>` Tras escribir el comando, comprueban que la rama master está desactualizada con respecto la rama master del servidor. Actualizan y comprueban el contenido de la rama.

15. Exportar el repositorio

¿Qué tendremos que hacer si queremos en un archivo comprimido de formato zip el contenido actual de nuestro entorno de trabajo? (busca en la ayuda `git archive`).

Fuentes utilizadas para la redacción de algunos de los ejercicios:

http://gitimmersion.com/index.html

http://gitref.org/index.html

http://gitready.com/