Ejercicios nivel avanzado de Git

Para hacer este guión de prácticas conservarás a tu pareja de prácticas, pero trabajaréis con otra (es decir, serán 4 alumnos, como máximo, los que estarán en el grupo de la práctica). Hasta que no se os indique seguiréis 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 nivel intermedio con git, crearemos la carpeta "home/usuario/GIT/Avanzado" ¿Cómo?, haremos un clonado desde la carpeta ya creada "GIT" de la rama "Avanzado" del repositorio Github de la asignatura: https://github.com/lorgut/EvolSoft.git. Sitúate dentro de "Avanzado" para realizar esta lista de ejercicios con git.

1. *Preparación del entorno*

Cada pareja se inventará un nombre *que la identificará. En "Avanzado" crea un fichero "READMENombreEquipo1_NombreEquipo2"* (haced que el *NombreEquipo1* y el *NombreEquipo2* aparezcan en el mismo orden[1]) que contenga "*Grupo de prácticas formado por el equipo NombreEquipo1*: *alumno1* y *alumno2* y por el equipo *NombreEquipo2*: *alumno3* y *alumno4*". *Añade y guarda el fichero (mensaje: "Entorno de trabajo"). Tras esto crea la rama (situándote directamente en ella) "NombreEquipo1vsNombreEquipo2"* (todos con el mismo orden). ***Sólo uno de los alumnos (alumno 4 del equipo 2)*** *Añade y guarda en esta rama los ficheros pb.c y makefile del campus virtual (correspondientes a este guión de prácticas) (mensaje: "Ficheros pb.c y makefile añadidos").*

2. *No queremos tantos commits (alumno 4)*

Vamos a unir los dos commits anteriores, ya que los cambios realizados anteriormente podrían ser indicados con un único mensaje (mensaje: "Rama de los equipos *NombreEquipo1, NombreEquipo2* versión inicial"). Para ello, utiliza el comando rebase, para fusionar estos dos commits en uno, con el mensaje anterior. Tras esto mira el historial y comprueba que ya no están los commits anteriores y está el nuevo commit. *Realiza un push de la rama "NombreEquipo1vsNombreEquipo2" dentro de la rama "Avanzado"* del repositorio remoto.

3. *Creando la rama de tu equipo (todos)*

Antes de crear la rama para tu equipo, realiza un pull del contenido que ha subido el alumno 4 al repositorio (el alumno 4 no lo hace) en tu rama "*NombreEquipo1vsNombreEquipo2*". *Crea una rama (cuyo padre sea la rama "NombreEquipo1vsNombreEquipo2"*) que se llame como el nombre de tu equipo "*NombreEquipo*" y comprueba las ramas que existen. Crea un fichero (en la rama de tu equipo) que se llame "READMENombreEquipo" que contenga: "Este es el entorno de trabajo del equipo *NombreEquipo*, donde trabajan los alumnos:

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

alumnoA y *alumnoB*, en el que 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"). Comprueba el contenido de las ramas.

4. *Trabajando con un repositorio remoto*

Para trabajar de forma independiente entre los grupos, uno de los miembros (alumno1 del equipo1), realiza un push de la rama de su equipo. Del mismo modo, el alumno3 del equipo 2 hará otro push de la rama de su equipo. Tras esto, todos los alumnos comprueban las ramas que existen en el repositorio remoto: lista las ramas remotas por consola y/o visualiza las ramas en la web del repositorio Github de la asignatura.

5. *Clave pública SSH*

Antes de seguir avanzando, todos los alumnos van a crearse su clave ssh. Para ello, seguimos las instrucciones que aparecen en https://help.github.com/articles/generating-ssh-keys

6. *Generando números*

El alumno3 actualiza el directorio de trabajo de la rama de su equipo y modifica el fichero pb.c para incluir el siguiente código que permitirá generar los números para las bolas blancas y la "power ball", cuando el usuario no introduzca ningún valor:

```
int whiteballs_computer_generated()
{
        return rand()%59+1;
}

int powerball_computer_generated()
{
        return rand()%39+1;
}

[…]

int main(int argc, char** argv)
{
    […]

    if (6 != count_balls)
    {

      for (int i = 0; i < 5; i++){
           balls[i] = whiteballs_computer_generated();
      }

        balls[5] = powerball_computer_generated(); // Power ball

      printf("Your numbers are: ");

        for (int i = 0; i < 5; i++){
           printf("%d ", balls[i]);

      }

      printf("\nAnd the power ball:");
      printf(" %d\n", balls[5]);
```

```
    }
    […]
}
```

Tras esto compila y ejecuta para comprobar que funciona correctamente. Guarda los cambios en el repositorio (mensaje: "Generando números aleatorios para las bolas blancas y la power ball"). Tras esto realiza un push en la rama de su equipo. (En este momento el otro equipo puede empezar la siguiente pregunta).

Tras subir los cambios, el alumno4 (del mismo equipo del alumno3), actualiza la rama de su equipo. Tras ver el código se da cuenta que la modificaciones hechas por su compañero tienen un fallo, no ha tenido en cuenta que los números no se repitan, así que elabora una función (llamada "checkwhiteballs()") que comprueba que todos los números de las bolas blancas sean diferentes entre ellos. Tras esto compila, comprueba que funciona, guarda los cambios en el repositorio (mensaje: "Función para controlar los números repetidos"), y realiza un push en la rama de su equipo.

```
void checkwhiteballs(int balls[5], int control)
{
  int last = balls[control];

  for (int i = 0; i < control; i++){
    if (last == balls[i]){
      balls[control] = whiteballs_computer_generated();
      break;
    }
  }
}
```

En la llamada de la función en main:

```
[…]

if (6 != count_balls)
{
   for (int i = 0; i < 5; i++){
        balls[i] = whiteballs_computer_generated();
        checkwhiteballs(balls, i);
   }

[…]
```

7. *Merge sin conflictos*

El alumno2 comprueba en el repositorio remoto (lista las ramas o ve a la web), que al igual que ellos, el otro equipo se ha creado una rama, así que crea una rama (estando situados en la rama *"NombreEquipo1vsNombreEquipo2"*) con el mismo nombre que la del otro equipo y actualiza esta rama con el contenido de la rama del otro equipo del repositorio. Comprueba las diferencias que hay entre el fichero pb.c que está en la rama *"NombreEquipo1vsNombreEquipo2"* con la rama que acaba de actualizar del otro equipo y las fusiona en la rama compartida. No debería dar conflicto, si los hay, utiliza mergetool para solventarlos y guarda los cambios (mensaje: "Fusión con los cambios realizados por el equipo NombreEquipo") ¡Ojo! Si no hubo conflictos, no hay que guardar los cambios.

Como los cambios que el alumno2 va a realizar son en otra parte del código y se centra en la función "calculate_result" del fichero pb.c de la rama de su equipo. Comprueba que tras ordenar las bolas blancas, no se imprimen, así que para comprobar que se ordenan realiza los siguientes cambios:

```
[…]

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

  printf("Your sorted numbers: \n");

  for (int i = 0; i < 5; i++){
    printf("%d ", white_balls[i]);
  }

  printf("The power ball: %d \n", power_ball);

  return 0;
 }
```

Compila y ejecuta el programa. Guarda los cambios (mensaje: "Imprimiendo los números tras ordenarlos"). Finalmente decide que sus cambios pueden mejorarse, así que no va a fusionarlos con la rama *"NombreEquipo1vsNombreEquipo2"*, por lo que hace un push de la rama "*NombreEquipo1vsNombreEquipo2"* y otro push de la rama de su equipo.

El alumno1 actualiza su directorio de trabajo, tanto la rama de su equipo, como la rama "*NombreEquipo1vsNombreEquipo2*". Comprueba las diferencias que hay en ambas ramas y decide fusionar los cambios del fichero "pb.c". Aunque antes de hacerlo, piensa que los cambios realizados por su compañero, deberían ir en una función aparte, así que todo lo que implementó su compañero (en la rama de su equipo), lo pone en una función que llama: "showing_results", con su correspondiente llamada en la función main (después de la llamada a la función "calculate_result"). Compila y comprueba que todo está correcto.

```
void showing_results(int white_balls[5], int power_ball)
{
  printf("Your sorted numbers: \n");

  for (int i = 0; i < 5; i++){
    printf("%d ", white balls[i]);
  }

  printf("The power ball: %d \n", power_ball);
}
```

Guarda los cambios (mensaje: "Función showing_results"). Ahora es cuando realiza la fusión en la rama *"NombreEquipo1vsNombreEquipo2"* de los cambios de la rama de su equipo con la rama *"NombreEquipo1vsNombreEquipo2"*, no debería de haber ningún conflicto, si los hubiese, utilizamos mergetool y guardaríamos los cambios (mensaje: "Fusión con rama *NombreEquipo1vsNombreEquipo2*"). Y realiza un push de la rama *"NombreEquipo1vsNombreEquipo2".*

8. *Merge*

El alumno4 actualiza de su directorio de trabajo la rama "*NombreEquipo1vsNombreEquipo2*", y comprueba que el código que había implementado, no está contemplado en esta rama, así que decide fusionar sus cambios para que se contemplen en la rama común. Si hubo conflictos, resuélvelos con mergetool y guarda los cambios (mensaje: "Función checkwhiteballs fusionada"). Compila y comprueba la aplicación y hace un push de la rama "*NombreEquipo1vsNombreEquipo2".* Para tener la última versión en la rama de su equipo, decide actualizarla con la rama *"NombreEquipo1vsNombreEquipo2".* Por otro lado, piensa que los "defines" se declararon por algo, y en vez de utilizar en las funciones "whiteballs_computer_generated" y "powerball_computer_generated" los números 59 y 39, cambia estos valores por los defines (estos cambios los realiza directamente en la rama *"NombreEquipo1vsNombreEquipo2"*). Guarda los cambios (mensaje: "Usando defines") y hace un push de la rama *"NombreEquipo1vsNombreEquipo2"* en la rama "*NombreEquipo1vsNombreEquipo2"* del repositorio. Y como son pocos los cambios, los realiza en su rama del equipo, guarda los cambios (mismo mensaje) y realiza un push de su rama.

9. *Alumno1 vs alumno3 (conflictos)*

Ambos alumnos actualizan la rama de su equipo y la rama que tienen en común. Para poder trabajar con la última versión en las ramas de sus respectivos equipos, la actualizan con *"NombreEquipo1vsNombreEquipo2"*.

Tras hacer esto, ambos alumnos se disponen a implementar (en la rama de su equipo) una función que simule la generación de los números premiados por el sorteo (mostrándolos posteriormente por pantalla) y su correspondiente llamada en la función main, que estará antes de la llamada a la función "calculate result". La función se llama *"void lottery_numbers_simulation()"* y genera las 5 bolas blancas y la power ball. En la función se generan los números (bolas blancas y powerball), se comprueban que las bolas blancas sean diferentes y se muestran los números por pantalla (no queremos todavía que se ordenen), (ayuda: puedes usar las funciones que ya existen). Declara todas las variables locales que consideres necesarias para su implementación.

Cuando los alumnos finalizan, guardan los cambios (mensaje: "Simulación de los números premiados por NombreEquipo"), fusionan con la rama *"NombreEquipo1vsNombreEquipo2"*, y hacen un push de la rama *"NombreEquipo1vsNombreEquipo2"*.

Si a la hora de hacer un push, hay conflicto, tendrás que solventarlo... El otro equipo terminó la implementación antes. A la hora de hacer la fusión quédate con la función que consideres que está mejor implementada. Una vez resueltos los conflictos con mergetool, guarda los cambios (mensaje: "Función lottery_numbers_simulation tras conflicto") y haz un push en la rama común.

10. *Ficheros a ignorar (alumnos 2 y 4)*

El alumno2 y el alumno4 actualizan en su directorio de trabajo, la rama *"NombreEquipo1vsNombreEquipo2".* Comprueban si en el fichero pb.c de la rama de su

equipo hay alguna diferencia que deba contemplarse con respecto la rama compartida, en caso afirmativo, la fusionará para tener en la rama de su equipo la última versión.

Tras tener todo en orden, (el alumno4 puede seguir en el párrafo siguiente) el alumno2 piensa que sería una buena práctica, y para que git no se quejase, tener controlados los ficheros que en el proyecto se quieren ignorar: ejecutable, .o (si es que existe) y .orig (creado tras hacer las fusiones). Para ello configura en la rama "NombreEquipo1vsNombreEquipo2" el fichero .gitignore para que estos ficheros queden ignorados y de este modo todos los usuarios pueden trabajar tranquilos. Una vez hecho esto, guarda los cambios (mensaje: "Fichero .gitignore para ambos equipos"), realiza un push y comprueba tu estado.

El alumno4, quiere ir guardando los resultados de diferentes pruebas en un fichero que se llama "pruebas", pero sabe que en cuanto lo cree, git va a "quejarse" de que éste no se encuentra bajo el control de versiones. Para que no ocurra esto, y ya que sólo él quiere almacenar estos datos, configura en su directorio de trabajo el fichero "info/exclude" para que ignore el fichero "pruebas" donde almacenará los resultados. Tras esto realiza (por ejemplo) las siguientes pruebas: ./pb >> pruebas, ./pb 3 4 15 6 32 28 >> pruebas, ./pb 22 16 37 42 9 10 >> pruebas ... Y comprueba que ignora el fichero.

11. *Fetch + merge (alumnos 1 y 3)*

El alumno1, actualiza la rama de su equipo y la rama compartida, en el caso de ver alguna diferencia entre la rama compartida y la de su equipo, fusionará en la de su equipo los cambios para tener la última versión. En caso contrario, seguirá trabajando en el código en la función "showing_results". Ya que esta función se está usando para mostrar tanto los números del usuario, como los números premiados, decide realizar las siguientes modificaciones (ojo en las modificaciones en el main hay que mover la llamada de la función "lottery_numbers_simulation()" y añadir código):

```
void showing_results(int white_balls[5], int power_ball)
{
  printf("The numbers of the white balls sorted: \n");

  for (int i = 0; i < 5; i++){
    printf("%d ", white_balls[i]);
  }

  printf("The power ball: %d \n", power_ball);
}

[…]

int main(int argc, char** argv)
{

[…]

   printf("\n--- The lottery numbers ---\n");
   lottery_numbers_simulation();
   printf("%d percent chance of winning\n", result);

   return 0;
```

```
[…]

}
```

Compila y ejecuta el programa para comprobar el resultado. Guarda los cambios (mensaje: "Función showing_results"), y realiza un push en la rama de *su equipo.*

El alumno3 (espera a que el alumno1 termine), se da cuenta que hace tiempo que no actualiza la ramas de su equipo. Para la rama que tienen en común, como no está seguro de querer fusionar directamente los cambios, realiza un "`git fetch`". La rama común apunta a una nueva rama (que se eliminará en cuanto fusionemos) que se llama "`FETCH_HEAD`", comprueba las diferencias y fusiona los cambios. Por otro lado, también siente curiosidad por saber qué es lo que está haciendo el otro equipo, así que se crea una rama (con el mismo nombre que el otro equipo) y se trae sus cambios del repositorio… no está seguro de los cambios que ellos han subido, así que prefiere primero descargar y si les interesa los cambios ya los fusionará con la rama compartida de su directorio de trabajo. Tras ver las diferencias, decide fusionar en la rama compartida (no debería de haber conflictos). Y realiza un push de la rama compartida.

12. *Alumno2 vs alumno4 (conflictos)*

El alumno2 y el alumno4 actualizan en su directorio de trabajo, la rama de su equipo y la rama *"NombreEquipo1vsNombreEquipo2".* Comprueban si en el fichero pb.c de la rama de su equipo hay alguna diferencia que deba contemplarse con respecto la rama compartida, en caso afirmativo, la fusionará para tener en la rama de su equipo la última versión.

Una vez actualizados los contenidos, ambos alumnos se disponen (en las ramas de sus equipos) a crear una función que compruebe que los números del usuario coincidan con los premiados. Para ello primero tendrán que hacer los siguientes cambios en la función main (recuerda borrar las líneas que ya no te hacen falta):

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

    printf("\n--- The lottery numbers ---\n");
    lottery_numbers_simulation(lott);
    int result = calculate_result(balls, power_ball);
    showing_results(balls, power_ball);

    if (result < 0)
    {
        goto usage_error;
    }

[…]
```

Ahora la función "lottery_numbers_simulation" ha cambiado, haz los cambios necesarios para que concuerde con las últimas modificaciones. Compila y ejecuta para comprobar que

has realizado los cambios correctamente. Guarda los cambios (mensaje: "Modificaciones en la entrada de la función lottery_numbers_simulation").

Tras esto, se centra en la función "calculate_result", que es donde van a realizarse las comparaciones. Pasa el vector "lott" como parámetro de la función (modifica la llamada y la propia función), guarda los cambios (mensaje: "Adaptando la función calculate_result para calcular probabilidades").

Los cálculos se realizarán del siguiente modo: se comparan las 5 bolas blancas (las de la lotería frente las del usuario), si éstas son todas iguales, has acertado el 100% (se devuelve 1), si son iguales 4, el 80% (se devuelve 0,8)... Guarda los cambios (mensaje: "Cálculos para las bolas blancas"). Para la power ball, al ser un premio extraordinario, se le suma al valor que se ha calculado antes de las bolas blancas un 0,1. Guarda los cambios (mensaje: "Cálculos para la power ball"). Realiza las modificaciones necesarias en esta función para realizar estos cambios y guarda los cambios en el repositorio (con los mensajes que quieras) tantas veces como necesites. Compila y ejecuta para comprobar que has realizado los cambios correctamente ¡Ojo con el tipo de las variables!

¡ATENCIÓN NO TIENES RED PARA SUBIR LOS CAMBIOS!

El primer alumno que termine, le mandará en un correo el fichero generado al ejecutar el comando `git bundle create <nombrefichero.bundle> <NombreEquipo> <HEAD>` (pon el nombre que desees al fichero).

Si eres el alumno que ha recibido el correo donde se adjunta un fichero y se indica que se te notificará cómo han de subirse los cambios a la rama común, sal del repositorio clonado "Avanzado" y en su mismo nivel, coloca el fichero que has recibido. Con la orden `git bundle verify <nombrefichero.bundle>` verifica que el fichero es correcto. Y con la orden `git clone <nombrefichero.bundle> -b master <nombrerepositorio>` (pon al nuevo repositorio el nombre que quieras) se clonará el árbol que guardó el compañero en el fichero con el historial completo. Entra en el nuevo repositorio y comprueba el historial. Sube los cambios al repositorio.

13. Dividiendo commits

El alumno3 quiere finalizar la aplicación ordenando los números en la función "lottery_numbers_simulation(...)". Ordena, utilizando la misma función que en "calculate_result", los números generados que simulan los dados por la lotería. Antepón un comentario en esta llamada con el siguiente contenido: `//Sort the lottery numbers`.

Añade los siguientes comentarios en la función calculate_result:

```
// Percent white balls
for (int i = 0; i < 5; i++){
  for (int j = 0; j < 5; j++){
    if (white_balls[i] == lott[j])

  }
```

```
  }
  // Percent power ball
  if (power_ball == lott[5])
    result += 0.1;
```

Del mismo modo, añade en la función main, antes de la llamada a la función "calculate_result" las líneas:

```
// Head for the lottery numbers
printf("\n--- The lottery numbers ---\n");
// Head for my numbers
printf("\n--- Your lottery numbers ---\n");
```

Como son diferentes cambios los realizados, quieres hacer varios commits:

1. mensaje: "Ordenando los números de la lotería"
2. mensaje: "Output para los números del usuario"
3. mensaje: "Comentarios"

Utiliza git add -i para dividir los cambios en esos 3 commits y haz un push.

14. *Rebase manejo de ramas*

Los cuatro alumnos, visualizan el estado de su directorio de trabajo utilizando la orden gitk, o gitweb, comparando las tres ramas (o dos ramas) existentes en su repositorio local. Para que el historial no sea un caos, deciden unificar (localmente) en la rama común los cambios. Si existe algún conflicto al unir se usará mergetool.

15. *¿¡Quién hizo eso!?*

El jefe de los equipos ve el resultado del programa y quiere ver quién hizo la función que calcula los resultados. Hay algo de disputa entre ambos equipos porque cada uno se atribuye los cambios. Para que se solvente la duda, actualiza la rama compartida (a no ser que ya la tengas actualizada) y haz un `git blame <fichero>` del fichero pb.c y observa el resultado (también podemos hacerlo desde la web del repositorio remoto en github) ¿Solventa esto la duda? ¿Quién es ascendido, o despedido?