

IX

Comandos y usos

Llegó la hora de enumerar los comandos más importantes y su modo de empleo. XSE enumera un total de 232 comandos que se pueden ver junto con una escueta descripción en inglés pulsando *F1*. Pero es importante conocer bien los comandos principales.

Antes de nada, debemos diferenciar entre **dos tipos de comandos**:

- **Comandos estrictos:** Son los comandos que no requieren de ningún parámetro. Su función es estricta. Por ejemplo, el comando *faceplayer* hace que la última persona con la que interactuamos nos mire. Ese es el resultado del comando, ni más ni menos.
- **Comandos con parámetros:** Son comandos que necesitan una serie de parámetros que debemos configurar. Es por ejemplo el caso del comando *pause*, que hace que la ejecución del script se detenga durante los frames que queramos. La cantidad de frames, en este caso, sería el parámetro (puede haber uno o más dependiendo del comando). Por otro lado, todo es compilado en hexadecimal, como ya sabemos, pero XSE nos ofrece la posibilidad de poner los parámetros directamente en hexadecimal o de ponerlos en decimal y la propia herramienta se encarga de pasarlo a hexadecimal. Veámoslo con ejemplos si por ejemplo queremos usar un *pause* durante 10 frames:
 - **Parámetros en hexadecimal:** Se ponen precedidos por un "0x". Como queremos que la ejecución pare durante 10 frames, pondremos *pause 0xA*, ya que 10 es A en hexadecimal.
 - **Parámetros en decimal:** Se ponen tal cual, sin estar precedidos por nada. En nuestro ejemplo sería *pause 10*. En este caso, el programa se encargará de convertirlo a hexadecimal al compilarlo.

•Givepokemon:

Sirve para recibir un pokémon. El funcionamiento es sencillo:

```
Givepokemon 0x(pokémon) 0x(nivel) 0x(objeto) 0x0 0x0 0x0
```

Nota: En la carpeta de XSE podréis encontrar los archivos *stdpoke* y *stditems*, que incluyen la lista de pokémon y objetos respectivamente con el valor que corresponde a cada uno.

- **Pokémon:** El pokémon que queramos recibir.
- **Nivel:** El nivel que queremos que tenga el pokémon que vamos a recibir.
- **Objeto:** El objeto que queremos que lleve equipado el pokémon que vamos a recibir. Si no queremos que lleve objetos, pondremos *0x0*.

Por ejemplo, si queremos recibir un Pikachu en el nivel 20 que lleve equipado una poción lo pondremos así:

```
Givepokemon 0x19 0x14 0xD 0x0 0x0 0x0
```

Nota: Recordad que por defecto el *menú pokémon* está desactivado. Para activarlo hay que activar la flag correspondiente. Tenéis la lista de flags especiales en el apéndice del manual.

Items:

Giveitem:

Si el anterior comando era para recibir un pokémon, este es para recibir un objeto. Recordad que tenéis la lista en la carpeta de XSE. El funcionamiento es el siguiente:

```
Giveitem 0x(objeto) 0x(cantidad) 0x(mensaje)
```

- **Objeto:** Es el objeto que recibiremos.
- **Cantidad:** Es la cantidad de ese objeto que recibiremos.
- **Mensaje:** Pondremos *0x0* para los objetos que sean recibidos (que alguien te lo dé) o *0x1* para los que sean encontrados (los objetos que encuentras por las rutas). La diferencia será que en uno te dirá que has recibido un objeto y en la otra, que lo has encontrado.

En todo caso, sonará el sonido típico de obtener/encontrar un objeto y aparecerá un *msgbox* informándote de lo que has recibido/encontrado.

Giveitem2:

Es casi igual que el anterior, pero en este caso no mostrará ningún mensaje automático, pero se podrá poner el sonido que queramos que se reproduzca:

```
Giveitem2 0x(objeto) 0x(cantidad) 0x(sonido)
```

Additem:

Simplemente se añadirá un objeto a tu mochila. No mostrará mensajes, ni reproducirá sonidos, ni nada. Su funcionamiento es el siguiente:

```
Additem 0x(objeto) 0x(cantidad)
```

Removeitem:

Si los anteriores te daban un objeto, este te lo quita. Funciona de la siguiente manera:

```
Removeitem 0x(objeto) 0x(cantidad)
```

Por ejemplo, si ponemos *removeitem 0xD 0x5*, se entiende que se eliminarían de nuestra mochila 5 pociones.

Checkitem:

Comprueba si tienes un objeto específico en una cantidad específica. Este es su funcionamiento:

Checkitem 0x(objeto) 0x(cantidad)

Es muy parecido a un *checkflag*. Imaginemos que queremos que al hablar con una persona compruebe si tenemos un rubí y, en tal caso, nos diga algo referente a él:

```
(...)  
Checkitem 0x175 0x1  
Compare LASTRESULT 0x1  
If 0x1 goto @rubi  
(...)  
  
#org @rubi  
Msgbox @texto  
Release  
End  
  
#org @texto  
= ¡Tienes un rubí!\n¡Qué pasada!
```

Nota: Se comprueba de la misma forma que las variables (explicado más adelante).

Wildbattle:

Este comando se utilizará para iniciar un combate contra un pokémon salvaje. El funcionamiento es el siguiente:

Wildbattle 0x(pokémon) 0x(nivel) 0x(objeto)

- **Pokémon:** Es el pokémon contra el que lucharemos.
- **Nivel:** Es el nivel que tendrá el pokémon.
- **Objeto:** Si el pokémon llevará equipado algún objeto.

Trainerbattles:

Hay diferentes tipos de *trainerbattles*. En esencia todas son iguales, pero cambiando el tipo de batalla, cambiarán ligeramente los resultados y el comando podrá tener más o menos parámetros.

Cabe destacar que, una vez que ganemos una trainerbattle, no se podrá volver a repetir otra vez, ya que cada entrenador cuenta con una flag propia. Esto es en términos generales, ya que hay excepciones. Mediante el estado de esta flag se puede entender si le hemos derrotado ya o no. Pero esto lo podemos cambiar mediante los comandos *settrainerflag* para activar al entrenador y el comando *cleartrainerflag* para desactivarlo. Además, podemos comprobar si una flag está activada o desactivada con el comando *checktrainerflag*.

Trainerbattle 0:

Sigue la siguiente estructura:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1) @(texto2)
```

- **Tipo:** En este caso será *0x0*, ya que estamos hablando de las de tipo 0.
- **Nº entrenador:** Es el número que corresponde al entrenador con el que queramos luchar. Se pueden ver y editar en herramientas de edición de entrenadores como *PET* o *Advance Trainer*.
- **Texto1:** Pointer al texto que dirá antes de empezar a luchar.
- **Texto2:** Pointer al texto que dirá el oponente si ganas la batalla.

Un ejemplo sería el siguiente, teniendo en cuenta que lucharemos contra el entrenador número *0x54*:

```
(...)  
Trainerbattle 0x0 0x54 0x0 @lucha @ganas  
(...)
```

```
#org @lucha  
= ¡Luchemos!
```

```
#org @ganas  
= ¡¿Me has Ganado?!
```

Es el tipo de trainerbattle que se asigna a los entrenadores en las rutas. Al ejecutarse, el entrenador mirará al player cambiando la música a la típica de encuentro con un entrenador, nos dirá lo que asignamos a *@lucha* (en el ejemplo anterior) y comenzará la batalla. Si ganas, aparecerá el otro texto. Si pierdes, no aparecerá nada.

Este tipo de trainerbattle finaliza el script tras la ejecución de la batalla, por lo tanto, funcionarán los comandos que pongamos antes de ella, pero no los que pongamos después. Lo comandos que estén después sólo funcionarán si se vuelve a ejecutar el script estando ya la batalla ganada.

Trainerbattle 1:

En este caso, el oponente te dirá algo si le ganas pero no lo hará si pierdes. Además, se introducen varios cambios en los parámetros:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1) @(texto2) @(pointer)
```

- **Tipo:** En este caso usamos el *0x1*.

- **Nº entrenador:** El entrenador contra el que lucharemos.
- **Texto1:** Lo que nos dirá antes de empezar la batalla en un *msgbox* normal.
- **Texto2:** Lo que nos dirá si ganamos la batalla.
- **Pointer:** El pointer al que irá después de la batalla para seguir ejecutándose el script.

Al ejecutarse la batalla y ser ganada, te llevará al pointer que elijas. Si una vez ganada la batalla se vuelve a ejecutar el script, el *trainerbattle* no se ejecutará y, por tanto, no te llevará al offset en cuestión, sino que seguirá leyendo debajo.

Trainerbattle 2:

Aparentemente igual que el *Trainerbattle 1*. La diferencia podría radicar en que el juego original utiliza el tipo 1 para batallas como las realizadas contra líderes de gimnasios y utiliza el tipo 2 para batallas contra otros entrenadores.

Trainerbattle 3:

Este responde a la siguiente estructura:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1)
```

- **Tipo:** En este caso usamos el *0x3*.
- **Nº entrenador:** El entrenador contra el que lucharemos.
- **Texto1:** Lo que nos dirá cuando ganemos la batalla.

En este tipo de *trainerbattle* no se nos mostrará nada más que el mensaje de cuando ganamos al entrenador. Si queremos poner mensajes antes de luchar, deberemos utilizar el comando *msgbox*. En este caso, el entrenador no nos mirará al ejecutarse el *trainerbattle*, por lo que, si queremos que nos mire, deberemos utilizar el *faceplayer*. Además, una vez ganada la batalla, se sigue ejecutando el script.

Es importante tener en cuenta que este tipo de *trainerbattle* se ejecutará indefinidamente, es decir, si ejecutamos el script de nuevo, se volverá a ejecutar la batalla.

Trainerbattle 4:

Es muy similar al *trainerbattle 0*, ya que se utiliza para los típicos entrenadores que te encuentras en las rutas. La diferencia es que este tipo se utiliza para batallas dobles, es decir, de dos contra dos pokémons. En este caso, sólo podrás luchar si llevas en el equipo dos o más pokémons. Esta es la estructura:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1) @(texto2) @(texto3)
```

- **Tipo:** En este caso usamos el *0x4*.
- **Nº entrenador:** El entrenador contra el que lucharemos.
- **Texto1:** Lo que nos dirá antes de empezar la batalla en un *msgbox* normal.
- **Texto2:** Lo que nos dirá si ganamos la batalla.
- **Texto3:** Lo que nos dirá si no llevamos dos o más pokémons (no se producirá la lucha).

Al igual que en el de tipo 0, la ejecución de la batalla pondrá fin al script y sólo se ejecutarán los comandos posteriores al trainerbattle si volvemos a ejecutar el script habiendo ganado anteriormente.

Trainerbattle 5:

Atiende a la misma estructura que la de tipo 0. Este tipo de trainerbattle se utilizará para las revanchas, es decir, la pondremos para volver a luchar con un entrenador que ya hemos derrotado antes. Si no hemos derrotado al entrenador previamente, no se ejecutará la batalla. Estando derrotado el entrenador, sí se ejecutará indefinidamente, por lo que podrá ser necesaria la utilización de alguna condición (flags o variables) para que no se repita siempre que se ejecute el script. Tras la realización de la batalla, el script se seguirá leyendo, es decir, este tipo de trainerbattle no pone fin a la ejecución del script.

Trainerbattle 6:

Es muy similar a los tipos 1 y 2, ya que después de la batalla serás enviado al pointer que elijas. Sin embargo, este tipo de trainerbattle será utilizado para los combates dobles y, por tanto, llevará un parámetro más:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1) @(texto2) @(texto3)
@(pointer)
```

- **Tipo:** En este caso usamos el 0x6.
- **Nº entrenador:** El entrenador contra el que lucharemos.
- **Texto1:** Lo que nos dirá antes de empezar la batalla en un *msgbox* normal.
- **Texto2:** Lo que nos dirá si ganamos la batalla.
- **Texto 3:** Lo que nos dirá si no llevamos dos o más pokémons (la lucha no se realizará).
- **Pointer:** El pointer al que irá después de la batalla para seguir ejecutándose el script.

Trainerbattle 7:

Es muy similar al de tipo 5. Se utiliza para las revanchas en batallas dobles. Su estructura es la siguiente:

```
Trainerbattle 0x(tipo) 0x(nº entrenador) 0x0 @(texto1) @(texto2) @(texto3)
```

- **Tipo:** En este caso usamos el 0x7.
- **Nº entrenador:** El entrenador contra el que lucharemos.
- **Texto1:** Lo que nos dirá antes de empezar la batalla en un *msgbox* normal.
- **Texto2:** Lo que nos dirá si ganamos la batalla.
- **Texto 3:** Lo que nos dirá si no llevamos dos o más pokémons (la lucha no se realizará).

Parece ser que este tipo de trainerbattle sólo funcionaría utilizando el *buscapeleas*.

Trainerbattle 8:

Aparentemente tiene el mismo efecto que el tipo 6.

Trainerbattle 9:

Aparentemente tiene el mismo efecto que el tipo 3. La diferencia radica en que si cambiamos el `0x0` por `0x1`, `0x2` o `0x3`, se ejecutará el tutorial de batalla. En el caso del tutorial, ganes o pierdas continuará el script (no irás a casa o al CP). El problema principal que tiene es que no está pensado para que lo hagas con más de un pokémon, por lo que podrás cambiar de pokémon durante la batalla, pero su imagen dará problemas (no se verá, aparecerá intermitentemente en algunas ocasiones, etc).

Trainerbattle A-FF:

El máximo que se puede utilizar es el tipo `0x9`. No hay más tipos disponibles. Por lo que si ponemos otro número desde `0xA` hasta `0xFF` se ejecutará la batalla del tipo `0x0`.

·Warp

Sirve para transportarnos a otro mapa. Podrá llevarnos a otro warp o a unas coordenadas concretas del mapa. Para **transportarnos a un warp** lo haremos así:

```
Warp 0x(banco) 0x(mapa) 0x(warp) 0x0 0x0
```

Es importante recordar que *Advance Map* nos da los números de los bancos de mapas y de los mapas en decimal, por eso, si los ponemos precedidos de "0x", debemos introducir el valor en hexadecimal (si no recuerdas esto, puedes mirar el principio del tema).

Si por el contrario quieres **aparecer en unas coordenadas concretas**:

```
Warp 0x(banco) 0x(mapa) 0xFF 0x(coordenada x) 0x(coordenada y)
```

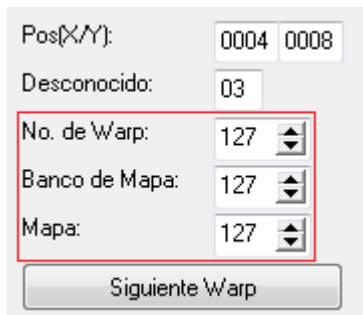
Existen varios tipos de warp. Son los siguientes:

- **Warp:** Es un warp normal. Se escuchará el sonido propio de los mismos.
- **Warpmuted:** No se reproducirá ningún sonido.
- **Warpwalk:** El jugador da un paso hacia adelante, se reproduce un sonido y se ejecuta el warp.
- **Warphole:** Este sólo usará dos parámetros: el banco y el mapa. Aparecerás en las mismas coordenadas que las que tengas cuando se ejecute el warp. Caerás del cielo en el nuevo mapa.
- **Warpteleport:** En este caso se producirá un efecto teletransporte.

Warp 127:

Esta es una configuración especial de los warps que ponemos en *Advance Map*. Lo normal es enlazar un warp con otro. Pero en este caso, podremos cambiar el warp al que está enlazado.

Un ejemplo claro son los ascensores. Podemos viajar por varias plantas y el warp cambia. Eso es porque utiliza la configuración 127. Consiste en configurar el warp de la siguiente manera:

A screenshot of a game configuration menu. It has several input fields: 'Pos(X/Y):' with values '0004' and '0008'; 'Desconocido:' with value '03'; 'No. de Warp:' with a dropdown menu showing '127'; 'Banco de Mapa:' with a dropdown menu showing '127'; and 'Mapa:' with a dropdown menu showing '127'. These four fields are grouped together and enclosed in a red rectangular box. Below this box is a button labeled 'Siguiente Warp'.

Ahora podemos determinar la salida del warp mediante el siguiente comando:

```
Setwarpplace 0x(banco) 0x(mapa) 0x(warp) 0x0 0x0
```

O si preferimos usarlo mediante coordenadas:

```
Setwarpplace 0x(banco) 0x(mapa) 0xFF 0x(coordenada x) 0x(coordenada y)
```

En el caso de que no hayamos ejecutado un *setwarpplace* antes de pasar por el warp, este nos llevara al último warp que utilizamos. Si no utilizamos ninguno, nos llevará al 0.0.

·Pokemart:

Sirve para crear una tienda. El comando apuntará un offset donde tendremos la lista de objetos. Funciona de la siguiente manera:

```
Pokemart @(pointer)
```

Los objetos se ponen mediante *#raw word*. Veamos un ejemplo:

```
(...)  
Pokemart @tienda  
(...)
```

```
#org @tienda  
#raw word 0x4  
#raw word 0xD  
#raw word 0xE  
#raw word 0x12  
#raw word 0x11  
#raw word 0xF  
#raw word 0x55  
#raw word 0x56  
#raw word 0x0
```

Lo más importante es saber que la lista de objetos se Cierra con un *#raw word 0x0*.

·Giveegg:

Es algo así como un *givepokemon* pero en este caso recibiremos un huevo:

```
Giveegg 0x(pokemon)
```

Simplemente hay que especificar el pokémon que saldrá del huevo y ya está. Se añadirá al equipo y, en caso de que esté completo, se enviará al PC.

·Dinero:

Con esta serie de comandos podrás trabajar con el dinero que tenga el jugador. Puedes darle dinero, quitárselo, comprobar el dinero que tiene...

Givemoney:

Con este comando podremos darle dinero al jugador:

```
Givemoney 0x(dinero) 0x0
```

Paymoney:

En este caso, pagaremos:

```
Paymoney 0x(dinero) 0x0
```

Showmoney:

Muestra una caja con el dinero que tenemos:

```
Showmoney 0x(coordenada x) 0x(coordenada y) 0x0
```

Updatemoney:

Si hacemos un cambio en el dinero (damos o quitamos) , la caja que ya tenemos puesta seguirá mostrando el mismo dinero. Este comando sirve para actualizarla:

```
Updatemoney 0x(coordenada x) 0x(coordenada y) 0x0
```

Hidemoney:

Ocultar la caja de dinero:

Hidemoney 0x(coordenada x) 0x(coordenada y)

Checkmoney:

Funciona de una manera muy simple. Un posible problema es que se comprueba el resultado con el mismo método que las variables. Si no sabes hacerlo, espera hasta haber leído y entendido el tema XI (*Scripts de gatillo y de nivel*) y luego vuelve a este apartado.

El funcionamiento es el siguiente:

Checkmoney 0x(dinero) 0x0

Ahora debemos comprobarlo mediante un *if*. Lo normal es simplemente comprobar si no se tiene el dinero poniéndole *0x0* al *if* (que significa que la cantidad de dinero que tenemos es mejor a la que hemos comprobado y, por tanto, no tenemos suficiente dinero para lo que queramos hacer).

Ejemplo:

Vamos a entenderlo mejor con un ejemplo. Queremos que si pagamos \$1.000 (0x3E8 en hexadecimal), nos den una contraseña, por ejemplo. Veámoslo:

```
#org @inicio
showmoney 0x0 0x0 0x0      'Muestra el dinero
msgbox @1 0x5
compare LASTRESULT 0x1
if 0x1 goto @si
hidemoney 0x0 0x0          'No queremos pagar, oculta el dinero
end

#org @si
checkmoney 0x3E8 0x0      'Comprueba si tenemos suficiente dinero
if 0x0 goto @nodinero      'Si no lo tenemos, va a @nodinero
paymoney 0x3E8 0x0        'Si lo tenemos, lo paga
updatemoney 0x0 0x0 0x0    'Actualiza la caja del dinero
msgbox @2 0x6
hidemoney 0x0 0x0          'Finalmente oculta el dinero
end

#org @nodinero
msgbox @3 0x6
hidemoney 0x0 0x0          'Oculta el dinero
end

#org @1
= Si me pagas $1.000 te doy la\ncontraseña.

#org @2
= Muy bien, esta es la contraseña:\n"8742".

#org @3
= ¿Intentas tomarme el pelo?\nNo tienes suficiente dinero.
```

·Special:

Ejecuta una función especial del juego (ASM). Si abrimos el rom con un editor hexadecimal y buscamos, podemos localizar una tabla de *specials*. La tabla está compuesta simplemente por punteros, unos seguidos de otros. Cada puntero apunta hacia un *special* y, dependiendo de la posición que ocupe, tendrá un número u otro. Es decir, el primer puntero apuntaría al *special 0x0*, el segundo apuntaría al *special 0x1*, el tercero apuntaría al *special 0x2*...

Pero eso no es lo importante. Lo que nos interesa es saber qué *special* realiza una función concreta que nosotros quisiéramos utilizar. En *XSE*, aunque incompleta, podemos encontrar la lista de *specials* y para lo que sirven.

Respecto al funcionamiento es tan simple como *special 0x(número)*. Por ejemplo, el *0x0* sirve para curar a todos los pokémons del equipo. Pues si en un script ejecutamos el *special 0x0*, nuestros pokémons se curarán por completo.

·Special2:

Es una variante del *special*, pero en este caso añadiremos una variable de la siguiente forma:

Special2 0x(variable) 0x(special)

Este comando se utiliza para *specials* que devuelven un valor a una variable. En el caso del *special 0x0*, que se limita a curar a los pokémons, no devuelve ningún valor que nos interese. Pero pueden existir *specials* que nos devuelvan un valor que sí nos interesa a nuestra variable. Es el caso del *special 0xBA*, que sirve para determinar qué pokémon elegimos de nuestro equipo. Un ejemplo de esto:

```
(...)  
Special 0xBC      'Abre el menú pokémon  
Waitstate  'Espera a que seleccionemos el pokémon  
Special2 0x7FFF 0xBA  'Guarda el pokémon en la variable 0x7FFF  
(...)
```

De esta forma, con el *special 0xBA* podremos determinar qué pokémon elegimos del equipo.

Es importante reiterar que el *special2* devuelve un valor que se guarda en la variable y no que el *special* tome un valor que nosotros le demos de la variable, que sería justo lo contrario y se trata de un error bastante común.

·Sonidos:

Hay distintos comandos para trabajar con el sonido del juego. Son los siguientes:

Sound:

Reproduce un sonido, nunca debe reproducir una canción. En este caso, el sonido se ejecutará sin perjuicio de la música que esté sonando. Se utiliza como *sound 0x(sonido)*.

Checksound:

Sirve para comprobar si hay algún sonido, canción o fanfare reproduciéndose en ese momento. No tiene parámetros y devuelve *0x0* si no hay nada sonando, o *0x1* si lo hay. Se comprueba con un *if*.

```
(...)  
Checksound  
If 0x1 goto @haysonido  
(...)
```

Fanfare:

En español diríamos que se “reproduce una fanfarria”, pero para qué nos vamos a engañar, en inglés suena bastante mejor. Simplemente reproduce un sonido y se utiliza igual que el *sound: fanfare 0x(sonido)*. El *fanfare* se suele utilizar muy a menudo para reproducir el sonido de cuando recibes algo utilizándolo como *fanfare 0x13E*.

Waitfanfare:

No lleva ningún parámetro y su función es detener la ejecución del script hasta que se reproduce completamente el *fanfare*. Este comando es importante, porque es de esos comandos que parece que no sirven para nada, pero son los detalles que le añaden una profesionalidad bastante elevada al hack.

Playsong:

Un comando muy importante, ya que nos permitirá cambiar en cualquier momento la música que suena. Esto es el típico momento en el que vas avanzando por una ruta y de pronto cambia la música y aparece el rival para enfrentarse a nosotros.

Se utiliza de la siguiente manera:

```
Playsong 0x(canción) 0x1
```

El *0x1* se pone siempre por lo general. Respecto a las canciones, podéis escucharlas todas con *Sappy*.

Adicionalmente, existe el comando *playsong2* que cambia la canción pero elimina el último parámetro (*0x1*). Con lo cual, se utilizaría así: *playsong2 0x(canción)*.

Fadedefault:

Cuando hemos cambiado la canción, este comando sirve para que ésta deje de sonar y se vuelva a reproducir la música por defecto del mapa. Este comando no tiene ningún parámetro. Sin embargo, suele dar errores. El más común es que vuelva a sonar la canción por defecto

pero, tras una lucha con un pokémon salvaje, vuelva a sonar la canción que habíamos ejecutado con el *playsong*.

Para evitar este error, lo que haremos será utilizarlo de esta manera:

```
Playsong2 0x0  
Fadedefault
```

Fadesong:

Es igual que el *playsong* pero, en vez de empezar a sonar de momento, lo hace con un efecto de fundido. Se utiliza como *fadesong 0x(canción)*.

Fadeout:

Deja de reproducir la canción que se esté reproduciendo con un efecto fundido para quedar en silencio. Es decir, no habrá música. Se utiliza como *fadeout 0x(velocidad)*.

Importante: Es un comando que puede producir errores con los warps. Acompañarlo de un *playsong2 0x0* puede solucionar el error.

Fadein:

Igual que el anterior, pero esta vez será para volver a reproducir la canción. Se utiliza como *fadein 0x(velocidad)*. Si en el *fadeout* utilizamos el *playsong2*, en este caso no podríamos utilizar el *fadein*, sino el *fadedefault*.

Cry:

Reproduce el grito de un pokémon. Se utiliza como *cry 0x(pokémon) 0x(efecto)*. Lo normal es poner *0x0* en el efecto para que se reproduzca el grito en condiciones normales. Si quisiéramos, por ejemplo, reproducir el grito de un Pikachu, pondríamos *cry 0x19 0x0*.

Pero como he dicho, existen varios efectos:

- **0x0:** Normal.
- **0x1:** Se corta más rápido.
- **0x2:** Se ejecuta en un tono más fuerte.
- **0x3:** Se ejecuta en un tono más fuerte con un cierto eco.
- **0x4:** Sonido lejano y reservado.
- **0x5:** Sonido bajo.
- **0x6:** Grito con una ligera mayor fuerza.
- **0x7:** Grito corto y ligeramente bajo.
- **0x8:** Ligeramente más alto de lo normal.
- **0x9:** Sonido lejano, reservado y corto.
- **0xA:** Aparentemente normal.
- **0xB:** Ligeramente más bajo.
- **0xC:** Sonido ligeramente más corto de lo normal.
- **0xD - 0xFF:** Sonido normal.

Waitcry:

Igual que el *waitfanfare*. Este comando no lleva parámetros y sirve para que el script espere a que se haya reproducido por completo el *cry*.

Countpokemon:

Sirve para comprobar el número de pokémons que llevamos en el equipo. Se comprueba como una variable, por lo que, si aún no sabes hacerlo, espera a leer el tema XI (scripts de gatillo y de nivel) antes de intentar comprender este comando.

Su funcionamiento es simple, ya que no requiere parámetros. Si, por ejemplo, quisiéramos que no nos dejaran entrar a algún sitio sin llevar, al menos, 3 pokémons, lo pondríamos así:

```
(...)  
Countpokemon      'Guarda el número de pokémons en LASTRESULT  
Compare LASTRESULT 0x3      'Compara el número con 3  
If 0x0 goto @nolleva3      'Si lleva menos de 3, va a @nolleva3  
(...)
```

Comandos de overworlds:

En *Advance Map* ponemos un overworld y le configuramos la posición y el comportamiento. Pero esto se puede alterar mediante script con algunos comandos.

Spriteface:

Hay veces que queremos que, durante un script, un overworld mire hacia otro lado. Si además lo tenemos que mover o algo así, deberíamos hacerlo mediante un *applymovement*. Pero si sólo queremos que mire para otro sitio sin nada más, podemos ahorrarlo. Este comando funciona así: *spriteface 0x(persona) 0x(dirección)*. Por ejemplo, si quisiéramos que la persona número 3 mire hacia la derecha, lo pondríamos así:

```
Spriteface 0x3 0x4
```

Si queremos que el efecto se aplique al jugador, en el número de persona pondremos *0xFF*. Las direcciones son las siguientes:

- **0x1:** Mira abajo.
- **0x2:** Mira arriba.
- **0x3:** Mira a la izquierda.
- **0x4:** Mira a la derecha.

Spritebehave:

Cambia el comportamiento del overworld. Por ejemplo, si tenemos una persona que está “mirando alrededor”, podemos poner que “mire hacia arriba” o lo que queramos. Funciona así:

```
Spritebehave 0x(persona) 0x(comportamiento)
```

Para saber qué valor corresponde al comportamiento, vamos a *Advance Map* y desplegamos la lista de comportamientos del mini. Tendremos que contar hasta llegar al comportamiento que queramos. Se empieza a contar desde cero, es decir, el *0x0* sería “Sin movimiento”, el *0x1* sería “Mirar alrededor” y así sucesivamente.

Importante: Sólo puede ser ejecutado mediante scripts de nivel de tipo 03 o 05.

Movesprite:

Mueve un sprite a una posición determinada del mapa. Funciona de la siguiente manera:

```
Movesprite 0x(persona) 0x(coordenada x) 0x(coordenada y)
```

Este comando funciona con scripts asignados a postes, a gatillos, a personas y scripts de nivel de tipo 02 y 04.

Movesprite2:

Es exactamente igual que el anterior y se configura mediante los mismos parámetros. La diferencia radica en que, en este caso, no se podrá utilizar para scripts normales, sino que este comando está orientado a su uso en scripts de nivel de tipo 03 o 05.

Setmaptile:

Sirve para cambiar el mapeado del juego. Este comando nos permite cambiar un determinado bloque del mapa por otro. Podemos cambiar tantos bloques como queramos, pero tendremos que usar un *setmaptile* para cada bloque. Es decir, si quisiéramos cambiar cuatro bloques, tendríamos que usar cuatro *setmaptiles*. El cambio se mantendrá incluso si recargamos la partida. La única forma de volver al estado anterior será mediante un *warp* (o utilizando *setmaptiles* para volver a colocar los bloques originales).

Funciona de la siguiente manera:

```
Setmaptile 0x(coordenada x) 0x(coordenada y) 0x(bloque) 0x(paso)
```

- **Coordenadas:** Indican las coordenadas del mapa donde efectuaremos el *setmaptile*.
- **Bloque:** El nuevo bloque que pondremos en lugar del original. Para saber cuál es el bloque, vamos a la pestaña “Ver mapa” en *Advance Map* y, en la zona de la izquierda (donde se muestran todos los bloques del tileset), ponemos el ratón encima del que

queramos poner (sin pinchar). Mientras tengamos el ratón encima, aparecerá el número en la esquina inferior izquierda del programa.

- **Paso:** Determina si se puede pasar o no. Si ponemos *0x0*, el jugador podrá atravesar el bloque. Si ponemos *0x1*, no podrá atravesarlo.

Además, tenemos que ejecutar un *special* para que se vean los cambios, ya que si no lo hacemos, no funcionará. Se trata del *special 0x8E*. Simplemente hay que ponerlo después de los *setmaptiles*. Por ejemplo:

```
(...)  
Setmaptile 0x3 0xD 0x2 0x1  
Setmaptile 0x5 0x8 0x5 0x1  
Special 0x8E  
(...)
```

·Clima:

Se trata de dos comandos que nos servirán para cambiar el tiempo que hace en el mapa donde estemos. El tiempo cambiará tras la ejecución de los comandos y volverá al original cuando salgamos del mapa y volvamos a entrar. Es decir, si por ejemplo tenemos un mapa con tiempo normal y con un script hacemos que llueva, lloverá sólo mientras permanezcamos en el mapa, si salimos y volvemos a entrar, de nuevo tendremos el tiempo que pusimos en la configuración del cabezal del mapa.

Los comandos son los siguientes:

- **Setweather:** Sirve para definir el nuevo clima del mapa, pero no lo ejecuta. Se utiliza como *setweather 0x(clima)*.
- **Doweather:** Sirve para ejecutar el cambio de clima que definimos en el comando anterior. No requiere ningún parámetro.

Si por ejemplo queremos que llueva, pondríamos lo siguiente en el script:

```
(...)  
Setweather 0x3  
Doweather  
(...)
```

Estos son los climas que hay (también los puedes encontrar en el apéndice):

- **0x0:** Ninguno.
- **0x1:** Reflejo de nubes.
- **0x2:** Normal.
- **0x3:** Lluvia.
- **0x4:** 3 snowflakes.
- **0x5:** Tormenta.
- **0x6:** Niebla.
- **0x7:** Nieve.

- **0x8:** Tormenta de arena.
- **0x9:** Niebla (diagonal).
- **0xA:** Thin fog.
- **0xB:** Oscuro.
- **0xC:** Sobrecalentamiento.
- **0xD:** Tormenta 2.
- **0xE:** Bajo el agua.
- **0xF:** Ninguno.

En el caso de que quisiéramos que el clima cambiara aunque saliéramos del mapa, tendríamos que poner un script de nivel de tipo 03 o 05.

·Fadescreen:

Se trata de un efecto fundido en blanco o en negro que afecta a toda la pantalla, es decir, si se ejecuta, la pantalla se pondrá en blanca o en negra y no se verá nada. El funcionamiento es *fadescreen 0x(efecto)*. Los efectos son los siguientes:

- **0x0:** Efecto de salida del fundido en negro.
- **0x1:** Efecto de entrada del fundido en negro.
- **0x2:** Efecto de salida del fundido en blanco.
- **0x3:** Efecto de entrada del fundido en blanco.

Si ponemos un *fadescreen 0x0* o *0x2*, al ser el efecto de salida, la pantalla se tornará al color que hayamos elegido y volverá de paso al estado normal. En algunos casos es suficiente con eso, pero lo normal es poner primero el de entrada y luego el de salida. Veamos un ejemplo. Hablamos con una persona y, cuando acabos de hablar, ésta desaparecerá. Lo haremos con un fundido para que no se vea como desaparece. La persona tiene el número 2 y la flag 0x200 asignada. Lo haríamos así:

```
(...)
Fadescreen 0x1    'Vuelve la pantalla negra
Setflag 0x200     'Activa la flag para que el mini no vuelva a aparecer
Hidesprite 0x2    'Hace desaparecer al mini
Pause 0x30        'Espera 30 frames antes de continuar
Fadescreen 0x0    'Vuelve la pantalla a la normalidad
(...)
```

Adicionalmente, existe el commando *fadescreendelay*, que además del efecto, permite poner un tiempo de espera antes de que se ejecute. Funciona como *fadescreendelay 0x(efecto) 0x(tiempo)*. En este caso nos podríamos ahorrar el *pause*, ya que es algo así como un *pause* integrado en el propio comando. Introduciéndolo en el script anterior, quedaría así:

```
(...)
Fadescreen 0x1
Setflag 0x200
Hidesprite 0x2
Fadescreendelay 0x0 0x30 'Espera 30 frames y ejecuta el fundido
(...)
```

Cuidado: El *fadescreen* puede provocar errores si se usa cuando el mapa tiene asignado un clima que oscurece la pantalla (oscuro, lluvia, tormenta...). En ese supuesto, cada vez que se ejecute un *fadescreen* se oscurecerá un poco más la pantalla llegando, en caso de ejecutar varios, a quedar incluso totalmente negra. Utiliza un *fadescreen* con ese tipo de climas si es necesario, pero no abuses de ellos.

·Showpokepic:

Muestra un pokémon. Se usa como *showpokepic 0x(pokémon) 0x(coordenada x) 0x(coordenada y)*. En Rubí y Zafiro, este comando no funciona correctamente. En dichas ediciones, las paletas de la imagen saldrán mal. Por lo que tendremos que poner dos *showpokepic* para que el primero corrija las paletas.

En el caso de *Rojo Fuego y Verde Hoja*, lo usaremos tal cual. Por ejemplo, si queremos que salga en la esquina superior derecha mostrando un Pikachu, lo pondremos así:

```
(...)  
Showpokepic 0x19 0x0 0x0  
(...)
```

En el caso de *Rubí y Zafiro*, será, como hemos dicho antes, algo distinto. Tendremos que poner el *showpokepic* configurado con el pokémon y las coordenadas. Luego utilizaremos un *pause* para conseguir que cargue correctamente. Para acabar, ocultaremos esa imagen que aparecería mal y mostraremos de nuevo la imagen que ahora saldrá totalmente bien. Quedaría algo así utilizando el ejemplo anterior:

```
(...)  
Showpokepic 0x19 0x0 0x0  
Pause 0x1  
Hidepokepic  
Showpokepic 0x19 0x0 0x0  
(...)
```

Saldrá la imagen mal por unas décimas de segundo. Pero al ser tan poco tiempo, casi no nos hará preocuparnos por ello.

Para ocultar el pokepic usaremos el comando *hidepokepic*, que no requiere ningún parámetro.

·Multichoice:

Vamos a hacer un ejercicio de memoria y vamos a recordar el *Yes/No*. En aquel script, nos aparecía una caja donde podíamos escoger entre dos opciones: *Sí* o *No*. Y dependiendo de lo que escogiéramos, pasaría una cosa u otra. En este caso es lo mismo, la diferencia es que la caja contendrá hasta 6 opciones. Un ejemplo de esto es cuando vamos al PC en un *Centro Pokémon* y nos aparecen las opciones (pokémon, objetos...). Eso sería un *multichoice*. La lista de *multichoices* se puede encontrar en la guía de *XSE* para cada una de las ediciones. Los

diferentes *multichoices* se pueden editar, existen herramientas para ello como *Multichoice Editor* (desarrollada por mí) o *Multichoice Manager* (desarrollada por Gut_Bro).

En esta ocasión tenemos 3 comandos que hacen lo mismo, es decir, muestran una caja con diferentes opciones entre las que elegir, pero nos brindan ciertas variaciones. Estos son los tres comandos:

```
Multichoice 0x(coordenada x) 0x(coordenada y) 0x(multichoice)
0x(B)
```

```
Multichoice2 0x(coordenada x) 0x(coordenada y) 0x(multichoice)
0x(opción seleccionada) 0x(B)
```

```
Multichoice3 0x(coordenada x) 0x(coordenada y) 0x(multichoice)
0x(opciones por columna) 0x(B)
```

- **Coordenada x:** Coordenadas en el eje x donde aparecerá la caja.
- **Coordenada y:** Coordenadas en el eje y donde aparecerá la caja.
- **Multichoice:** Número del *multichoice* que se mostrará.
- **B:** Determina si se puede pulsar el botón *B* para salir. 0x0 determina que se puede usar, 0x1 determina que no.
- **Opción seleccionada:** En el caso del *multichoice2*, se puede determinar qué opción saldrá marcada por defecto. Es decir, si ponemos 0x1, por ejemplo, el *multichoice* aparecerá con la opción número 2 seleccionada por defecto. Ojo, seleccionada no quiere decir elegida, sino que es la opción desde la que se parte.
- **Opciones por columna:** Por defecto, aparecen todas las opciones en una misma columna, pero se puede hacer en el caso del *multichoice3*, que aparezca un determinado número de opciones por cada columna. Por ejemplo, si ponemos un *multichoice* con 4 opciones y en este parámetro ponemos que haya 0x2 opciones por columna, aparecerán dos opciones en la primera columna y las otras dos en la segunda columna.

Recuerdo que los *multichoices* se pueden editar, pero en este caso, vamos a hacer el ejemplo con uno original. En Fire Red, el multichoice 0x12 tiene las opciones *Yes*, *No* e *Info*. Si queremos que pase una cosa u otra dependiendo de la opción elegida, pondríamos algo así:

```
(...)
Multichoice 0x0 0x0 0x12 0x1      'Muestra el multichoice
Compare LASTRESULT 0x0          'Comprueba si elegimos la opción 1
If 0x1 goto @yes                'Si lo hicimos, va a @yes
Compare LASTRESULT 0x1          'Comprueba si elegimos la opción 2
If 0x1 goto @no                 'Si lo hicimos, va a @no
Compare LASTRESULT 0x2          'Comprueba si elegimos la opción 3
If 0x1 goto @info               'Si lo hicimos, va a @info
(...)
```

La opción seleccionada se guarda en la variable *LASTRESULT*. Es importante tener en cuenta que la primera opción es representada por 0x0, no por 0x1. La 0x0 es la primera, la 0x1 es la segunda, la 0x2 es la tercera... Y así sucesivamente.

·Random:

Sirve para generar un valor aleatorio. Es muy útil si queremos que algo pase precisamente de esa forma: aleatoria. Un ejemplo sería que al hablar con una persona, tuviéramos un 10% de posibilidades de que nos diera o dijera algo concreto. Luego volveremos a este ejemplo.

El comando *random* lo que hace es generar un valor entre los que nosotros queramos y guardarlo en *LASTRESULT*. Funciona como *random 0x(opciones)*. Es decir, si ponemos lo siguiente:

```
Random 0x4
```

El comando generará un número entre 0 y 3. 0 es siempre el primer valor que se puede generar, por lo tanto, como hemos dicho que genere 4 valores, lo generará entre 0, 1, 2 y 3 (es decir, 4 valores). Si ponemos que genere 6, generará del 0 al 5, si ponemos 9, generará del 0 al 8... Y así con cualquier valor. Ese valor se guarda en *LASTRESULT* y así podremos consultar cuál ha sido el valor generado. Lo comprobaremos con un *compare*. Puesto que *LASTRESULT* es una variable (concretamente la variable 0x800D), podemos comprobarlo como una variable, pero eso es explicado en el tema IX (Scripts de gatillo y de nivel), por lo que si lo crees conveniente, puedes esperar a leer dicho tema antes de seguir con esto.

Volviendo al ejemplo. Sabemos que el 10% es 10 posibilidades entre 100. O lo que es lo mismo, 1 entre 10. Luego entonces, vamos a generar 10 valores:

```
Random 0xA
```

Como hemos puesto que se genere un valor entre 10 posibilidades, el valor generado estará entre el 0 y el 9. Como hemos dicho, el valor generado aleatoriamente se guardará en *LASTRESULT*. En nuestro ejemplo queríamos un 10% y eso hemos dicho que es 1 entre 10. Luego lo único que tenemos que hacer es que únicamente uno de los diez valores sea el que nos lleve a otro pointer. Por ejemplo, el 5 (en este caso da igual el valor que escojamos entre los posibles, ya que con coger uno, estaríamos cubriendo el 10%). Quedaría así:

```
(...)  
Random 0xA          `Genera un valor entre 10 (0xA) posibilidades  
Compare LASTRESULT 0x5      `Comprueba si el valor generado es 0x5  
If 0x1 goto @asdf      `Si es 0x5, va a @asdf  
(...)
```

Este es uno de los mil ejemplos que se podrían poner. Puedes hacer cosas más complejas, como que dependiendo del valor que genere vaya a un pointer determinado y pasen cosas distintas, por ejemplo.

·Buffers:

Se trata de una memoria que nos permitirá mostrar cosas en un *msgbox*. Disponemos de tres buffers distintos que se introducen en los textos como *[buffer1]*, *[buffer2]* y *[buffer3]*. Podemos guardar distintas cosas en ellos, pero los comandos más importantes que trabajan con buffers son los siguientes:

Bufferpokemon:

Carga el nombre de un Pokémon en el buffer que queramos. Funciona como *bufferpokemon 0x(buffer) 0x(pokemon)*. El *buffer* se empieza a contar desde 0, es decir, si ponemos que el *buffer* sea el 0x0, se guardará en *[buffer1]*. Por ejemplo, guardaremos un Pikachu:

```
Bufferpokemon 0x0 0x19
```

Y en el texto pondremos algo así:

```
= Mi favorito es [buffer1].
```

Como en *[buffer1]* tenemos guardado un Pikachu (o el Pokémon que quisiéramos poner), el texto real que se mostrará en el juego será: "Mi favorito es PIKACHU".

Bufferfirstpokemon:

Guarda el nombre del primer Pokémon que tengamos en el equipo. Funciona como *bufferfirstpokemon 0x(buffer)*. Si después de haber ejecutado dicho comando, ponemos en el texto lo siguiente:

```
= Me encanta tu [buffer1].
```

En el juego nos dirá (si por ejemplo el primer Pokémon es Pidgey: "Me encanta tu PIDGEY".

Bufferpartypokemon:

Igual que el anterior, guarda el nombre de un Pokémon del equipo en un buffer, pero en este caso, no del primero, sino del que queramos. Es importante saber que se empieza a contar desde 0, luego el primer Pokémon del equipo sería el 0x0, el segundo el 0x1, el tercero el 0x2... Si por ejemplo queremos guardar el nombre del tercer Pokémon lo haremos así:

```
Bufferpartypokemon 0x0 0x2
```

En el texto se muestra igual que en los anteriores supuestos e igualmente se mostrará en los siguientes.

Bufferitem:

Como si de un *bufferpokemon* se tratara. En este caso guardará el nombre de un objeto con la forma *bufferitem 0x(buffer) 0x(objeto)*.

Bufferatack:

Guarda el nombre de un ataque en el buffer. Funciona de la misma manera: *bufferattack 0x(buffer) 0x(ataque)*.

Un ejemplo claro de este comando es cuando usamos *corte*. Primero comprueba qué pokémon usaremos y lo guarda en un buffer con el comando *bufferpartypokemon*. Después guarda en otro buffer con el comando *bufferattack* el ataque que usamos (*corte* en este caso). Así se muestra luego tanto el pokémon como el ataque en el *msgbox*.

Buffernumber:

Guarda el valor de una variable en un buffer y lo muestra en un *msgbox*. El valor se mostrará automáticamente en decimal. Es muy útil si queremos, por ejemplo, crear un banco con mi sistema.

**El sistema de bancos puede ser encontrado como un archivo adicional de este manual.*

Bufferstring:

Sirve para guardar un texto en un buffer para ser mostrado en un *msgbox*. Puede ser útil por ejemplo si queremos para casos en los que sólo cambien una determinada parte del texto, así evitamos compilar varios *msgbox* prácticamente iguales y ahorramos mucho espacio. Funciona como *bufferstring 0x(buffer) 0x(offset del texto)*. Por ejemplo, si tenemos el texto “Hola” en el offset 0x8019A0 y pusiéramos lo siguiente:

```
(...)  
Bufferstring 0x0 0x8019A0  
Msgbox @1 0x6  
End
```

```
#org @1  
= [buffer1]
```

De esta forma, en el *msgbox* se mostraría: “Hola”. Es un ejemplo muy simple, pero con buenas ideas puede dar mucho juego.

“Comandos y usos”

Manual redactado por **Javi4315**.

Queda estrictamente prohibida su distribución.