Troubleshooting
Solución de problemas

Now that our scripts become more complex, it's time to look at what happens when things go wrong. In this chapter, we'll look at some of the common kinds of errors that occur in scripts and examine a few useful techniques that can be used to track down and eradicate problems.
Ahora que nuestros guiones se vuelven más complejos, es hora de ver qué sucede cuando las cosas salen mal. En este capítulo, veremos algunos de los tipos comunes de errores que ocurren en los scripts y examinaremos algunas técnicas útiles que se pueden utilizar para rastrear y erradicar problemas.

Syntactic Errors
Sintaxis de errores

One general class of errors is syntactic. Syntactic errors involve mistyping some element of shell syntax. The shell will stop executing a script when it encounters this type of error.
Una clase general de errores es sintáctica. Los errores sintácticos implican escribir mal algún elemento de la sintaxis del shell. El shell dejará de ejecutar un script cuando encuentre este tipo de error.
In the following discussions, we will use this script to demonstrate common types of errors.
En las siguientes discusiones, usaremos este script para demostrar tipos comunes de errores.

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
```

As written, this script runs successfully.
Tal como está escrito, este script se ejecuta correctamente.

```
[me@linuxbox ~]$ trouble
Number is equal to 1.
```

Missing Quotes
Cotizaciones faltantes

Let's edit our script and remove the trailing quote from the argument following the first echo command.
Editemos nuestro script y eliminemos la cita final del argumento que sigue al primer comando echo.

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
```

Here is what happens:
Esto es lo que sucede:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 10: unexpected EOF while looking for matching
`"`
/home/me/bin/trouble: line 13: syntax error: unexpected end of file
```

It generates two errors. Interestingly, the line numbers reported by the error messages are not where the missing quote was removed but rather much later in the program. If we follow the program after the missing quote, we can see why. bash will continue looking for the closing quote until it finds one, which it does, immediately after the second echo command. After that, bash becomes very confused. The syntax of the subsequent if command is broken because the fi statement is now inside a quoted (but open) string.
Genera dos errores. Curiosamente, los números de línea informados por los mensajes de error no son donde se eliminó la cita que faltaba, sino mucho más tarde en el programa. Si seguimos el programa después de la cita que falta, podemos ver por qué. bash continuará buscando la cita de cierre hasta que encuentre una, lo que hace, inmediatamente después del segundo comando echo. Después de eso, bash se vuelve muy confuso. La sintaxis del comando if subsiguiente está rota porque la instrucción fi ahora está dentro de una cadena entre comillas (pero abierta).
In long scripts, this kind of error can be quite hard to find. Using an editor with syntax highlighting will help since, in most cases, it will display quoted strings in a distinctive manner from other kinds of shell syntax. If

a complete version of vim is installed, syntax highlighting can be enabled by entering this command:
En scripts largos, este tipo de error puede ser bastante difícil de encontrar. El uso de un editor con resaltado de sintaxis ayudará ya que, en la mayoría de los casos, mostrará cadenas entre comillas de una manera distinta a otros tipos de sintaxis de shell. Si se instala una versión completa de vim, el resaltado de sintaxis se puede habilitar ingresando este comando:

```
:syntax on
```

Missing or Unexpected Tokens
Tokens faltantes o inesperados

Another common mistake is forgetting to complete a compound command, such as if or while . Let's look at what happens if we remove the semicolon after test in the if command:
Otro error común es olvidar completar un comando compuesto, como si o while. Veamos qué sucede si eliminamos el punto y coma después de la prueba en el comando if:

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ] then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
```

The result is this:
El resultado es esto:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 9: syntax error near unexpected token `else'
/home/me/bin/trouble: line 9: `else'
```

Again, the error message points to an error that occurs later than the actual problem. What happens is really pretty interesting. As we recall, if accepts a list of commands and evaluates the exit code of the last

command in the list. In our program, we intend this list to consist of a single command, [, a synonym for test . The [command takes what follows it as a list of arguments; in our case, that's four arguments: $number, 1 , = , and] .
Nuevamente, el mensaje de error apunta a un error que ocurre más tarde que el problema real. Lo que sucede es realmente bastante interesante. Como recordamos, si acepta una lista de comandos y evalúa el código de salida del último comando de la lista. En nuestro programa, pretendemos que esta lista consista en un solo comando, [, un sinónimo de prueba. El comando [toma lo que le sigue como una lista de argumentos; en nuestro caso, son cuatro argumentos: $número, 1; = and].
With the semicolon removed, the word then is added to the list of arguments, which is syntactically legal. The following echo command is legal, too. It's interpreted as another command in the list of commands that if will evaluate for an exit code. The else is encountered next, but it's out of place since the shell recognizes it as a reserved word (a word that has special meaning to the shell) and not the name of a command, which is the reason for the error message.
Con el punto y coma eliminado, la palabra se agrega a la lista de argumentos, que es sintácticamente legal. El siguiente comando echo también es legal. Se interpreta como otro comando en la lista de comandos que evaluará un código de salida. El else se encuentra a continuación, pero está fuera de lugar ya que el shell lo reconoce como una palabra reservada (una palabra que tiene un significado especial para el shell) y no como el nombre de un comando, que es el motivo del mensaje de error.

Unanticipated Expansions
Expansiones imprevistas

It's possible to have errors that occur only intermittently in a script. Sometimes the script will run fine, and other times it will fail because of the results of an expansion. If we return our missing semicolon and change the value of number to an empty variable, we can demonstrate.
Es posible tener errores que ocurren solo de forma intermitente en un script. A veces, el script se ejecutará bien y otras veces fallará debido a los resultados de una expansión. Si devolvemos el punto y coma que falta y cambiamos el valor del número a una variable vacía, podemos demostrar.

```
#!/bin/bash

# trouble: script to demonstrate common errors
number=
if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
```

Running the script with this change results in the following output:
La ejecución del script con este cambio da como resultado el siguiente resultado:

```
[me@linuxbox ~]$ trouble
/home/me/bin/trouble: line 7: [: =: unary operator expected
Number is not equal to 1.
```

We get this rather cryptic error message, followed by the output of the second echo command. The problem is the expansion of the number variable within the test command. When the following command:
Recibimos este mensaje de error bastante críptico, seguido de la salida del segundo comando echo. El problema es la expansión de la variable numérica dentro del comando de prueba. Cuando el siguiente comando:

```
[ $number = 1 ]
```

undergoes expansion with number being empty, the result is this:
se expande con el número vacío, el resultado es este:

```
[ = 1 ]
```

which is invalid and the error is generated. The = operator is a binary operator (it requires a value on each side), but the first value is missing, so the test command expects a unary operator (such as -z) instead. Further, since the test failed (because of the error), the if command receives a non-zero exit code and acts accordingly, and the second echo command is executed.
que no es válido y se genera el error. El operador = es un operador binario (requiere un valor en cada lado), pero falta el primer valor, por lo que el comando de prueba espera un operador unario (como -z) en su lugar. Además, dado que la prueba falló (debido al error), el comando if recibe un código de salida distinto de cero y actúa en consecuencia, y se ejecuta el segundo comando de eco.
This problem can be corrected by adding quotes around the first argument in the test command.
Este problema se puede corregir agregando comillas alrededor del primer argumento en el comando de prueba.

```
[ "$number" = 1 ]
```

Then when expansion occurs, the result will be this:
Luego, cuando se produzca la expansión, el resultado será este:

```
[ "" = 1 ]
```

This yields the correct number of arguments. In addition to empty strings, quotes should be used in cases where a value could expand into multiword strings, as with filenames containing embedded spaces.
Esto produce el número correcto de argumentos. Además de las cadenas vacías, se deben utilizar comillas en los casos en que un valor se pueda expandir en cadenas de varias palabras, como ocurre con los nombres de archivo que contienen espacios incrustados.

> Note
> Nota

> Make it a rule to always enclose variables and command substitutions in double quotes unless word splitting is needed

> Establezca como regla incluir siempre las variables y las sustituciones de comandos entre comillas dobles, a menos que sea necesario dividir las palabras

Logical Errors
Errores Lógicos

Unlike syntactic errors, logical errors do not prevent a script from running. The script will run, but it will not produce the desired result because of a problem with its logic. There are countless numbers of possible logical errors, but here are a few of the most common kinds found in scripts:
A diferencia de los errores sintácticos, los errores lógicos no impiden la ejecución de un script. El script se ejecutará, pero no producirá el resultado deseado debido a un problema con su lógica. Hay un sinnúmero de posibles errores lógicos, pero a continuación se muestran algunos de los tipos más comunes que se encuentran en los scripts:

- **Incorrect conditional expressions**. It's easy to incorrectly code an if / then / else expression and have the wrong logic carried out.Sometimes the logic will be reversed, or it will be incomplete.
  Expresiones condicionales incorrectas. Es fácil codificar incorrectamente una expresión if / then / else y ejecutar la lógica incorrecta. A veces, la lógica se invertirá o estará incompleta.

- **"Off by one"** errors. When coding loops that employ counters, it is possible to overlook that the loop may require that the counting start with zero, rather than one, for the count to conclude at the correct point. These kinds of errors result in either a loop "going off the end" by counting too far or a loop missing the last iteration by terminating one iteration too soon.
  Errores de "Desactivado en uno". Al codificar bucles que emplean contadores, es posible pasar por alto que el bucle puede requerir que el conteo comience con cero, en lugar de uno, para que el conteo concluya en el punto correcto. Este tipo de errores dan como resultado que un bucle "se salga del final" al contar demasiado o que un bucle pierda la última iteración al terminar una iteración demasiado pronto.

- **Unanticipated situations.** Most logic errors result from a program encountering data or situations that were unforeseen by the programmer. As we have seen, this can also include unanticipated expansions, such as a filename that contains embedded spaces that expands into multiple command arguments rather than a single filename.
  Situaciones imprevistas. La mayoría de los errores lógicos se deben a que un programa encuentra datos o situaciones imprevistas por parte del programador. Como hemos visto, esto también puede incluir expansiones inesperadas, como un nombre de archivo que contiene espacios incrustados que se expanden en múltiples argumentos de comando en lugar de un solo nombre de archivo.

Defensive Programming
Programación defensiva

It is important to verify assumptions when programming. This means a careful evaluation of the exit status of programs and commands that are used by a script. Here is an example, based on a true story. An unfortunate system administrator wrote a script to perform a maintenance task on an important server. The script contained the following two lines of code:

Es importante verificar las suposiciones al programar. Esto significa una evaluación cuidadosa del estado de salida de los programas y comandos que utiliza un script. Aquí hay un ejemplo, basado en una historia real. Un desafortunado administrador del sistema escribió un script para realizar una tarea de mantenimiento en un servidor importante. El script contenía las siguientes dos líneas de código:

```
cd $dir_name
rm *
```

There is nothing intrinsically wrong with these two lines, as long as the directory named in the variable, dir_name , exists. But what happens if it does not? In that case, the cd command fails, and the script continues to the next line and deletes the files in the current working directory. Not the desired outcome at all! The hapless administrator destroyed an important part of the server because of this design decision.
No hay nada intrínsecamente incorrecto con estas dos líneas, siempre que exista el directorio nombrado en la variable, dir_name. Pero, ¿qué pasa si no es así? En ese caso, el comando cd falla y la secuencia de comandos continúa a la siguiente línea y elimina los archivos en el directorio de trabajo actual. ¡No es el resultado deseado en absoluto! El desafortunado administrador destruyó una parte importante del servidor debido a esta decisión de diseño.

Let's look at some ways this design could be improved. First, it might be wise to ensure that the dir_name variable expands into only one word by quoting it and make the execution of rm contingent on the success of cd .
Veamos algunas formas en que se podría mejorar este diseño. Primero, sería prudente asegurarse de que la variable dir_name se expanda en una sola palabra citandola y haciendo que la ejecución de rm dependa del éxito de cd

```
cd "$dir_name" && rm *
```

This way, if the cd command fails, the rm command is not carried out. This is better but still leaves open the possibility that the variable, dir_name, is unset or empty, which would result in the files in the user's home directory being deleted. This could also be avoided by checking to see that dir_name actually contains the name of an existing directory.
De esta forma, si falla el comando cd, el comando rm no se ejecuta. Esto es mejor, pero aún deja abierta la posibilidad de que la variable, dir_name, no esté configurada o esté vacía, lo que resultaría en la eliminación de los archivos en el directorio de inicio del usuario. Esto también podría evitarse comprobando que dir_name realmente contenga el nombre de un directorio existente

```
[[ -d "$dir_name" ]] && cd "$dir_name" && rm *
```

Often, it is best to include logic to terminate the script and report an error when a situation such as the one shown previously occurs.
A menudo, es mejor incluir lógica para terminar el script e informar un error cuando ocurre una situación como la que se mostró anteriormente.

```
# Delete files in directory $dir_name
```

```
if [[ ! -d "$dir_name" ]]; then
    echo "No such directory: '$dir_name'" >&2
    exit 1
fi
if ! cd "$dir_name"; then
    echo "Cannot cd to '$dir_name'" >&2
    exit 1
fi
if ! rm *; then
    echo "File deletion failed. Check results" >&2
    exit 1
fi
```

Here, we check both the name, to see that it is an existing directory, and the success of the cd command. If either fails, a descriptive error message is sent to standard error, and the script terminates with an exit status of one to indicate a failure.
Aquí, verificamos tanto el nombre, para ver que es un directorio existente, como el éxito del comando cd. Si alguno de ellos falla, se envía un mensaje de error descriptivo al error estándar y el script termina con un estado de salida de uno para indicar un error.

Watch Out for Filenames
Cuidado con los nombres de archivo

There is another problem with this file deletion script that is more obscure but could be very dangerous. Unix (and Unix-like operating systems) has, in the opinion of many, a serious design flaw when it comes to filenames. Unix is extremely permissive about them. In fact, there are only two characters that cannot be included in a filename. The first is the / character since it is used to separate elements of a pathname, and the second is the null character (a zero byte), which is used internally to mark the ends of strings. Everything else is legal including spaces, tabs, line feeds, leading hyphens, carriage returns, and so on.
Hay otro problema con este script de eliminación de archivos que es más oscuro pero podría ser muy peligroso. Unix (y los sistemas operativos similares a Unix) tiene, en opinión de muchos, un grave defecto de diseño cuando se trata de nombres de archivo. Unix es extremadamente permisivo con ellos. De hecho, solo hay dos caracteres que no se pueden incluir en un nombre de archivo. El primero es el carácter / ya que se usa para separar elementos de un nombre de ruta, y el segundo es el carácter nulo (un byte cero), que se usa internamente para marcar los finales de cadenas. Todo lo demás es legal, incluidos los espacios, tabulaciones, avances de línea, guiones iniciales, retornos de carro, etc.

Of particular concern are leading hyphens. For example, it's perfectly legal to have a file named -rf ~. Consider for a moment what happens when that filename is passed to rm.
Son especialmente preocupantes los guiones iniciales. Por ejemplo, es perfectamente legal tener un archivo llamado -rf ~. Considere por un momento lo que sucede cuando ese nombre de archivo se pasa a rm.
To defend against this problem, we want to change our rm command in the file deletion script from this:
Para defendernos de este problema, queremos cambiar nuestro comando rm en el script de eliminación de archivos de esto:

```
rm *
```

to the following:
a lo siguiente:

```
rm ./*
```

This will prevent a filename starting with a hyphen from being interpreted as a command option. As a general rule, always precede wildcards (such as * and ?) with ./ to prevent misinterpretation by commands. This includes things like *.pdf and ???.mp3 , for example.
Esto evitará que un nombre de archivo que comience con un guión se interprete como una opción de comando. Como regla general, siempre preceda los comodines (como * y?) Con ./ para evitar malas interpretaciones por parte de los comandos. Esto incluye cosas como * .pdf y ???. Mp3, por ejemplo.

Portable File names
Nombres de archivos portátiles

To ensure that a filename is portable between multiple platforms (i.e., different types of computers and operating systems), care must be taken to limit which characters are included in a filename. There is a standard called the POSIX Portable Filename Character Set that can be used to maximize the chances that a filename will work across different systems. The standard is pretty simple. The only characters allowed are the uppercase letters A–Z, the lowercase letters a–z, the numerals 0–9, period (.), hyphen (-), and underscore (*). The standard further suggests that filenames not begin with a hyphen.*
*Para garantizar que un nombre de archivo sea portátil entre múltiples plataformas (es decir, diferentes tipos de computadoras y sistemas operativos), se debe tener cuidado de limitar qué caracteres se incluyen en un nombre de archivo. Existe un estándar llamado Conjunto de caracteres de nombre de archivo portátil POSIX que se puede utilizar para maximizar las posibilidades de que un nombre de archivo funcione en diferentes sistemas. El estándar es bastante simple. Los únicos caracteres permitidos son las letras mayúsculas de la A a la Z, las letras minúsculas de la a a la z, los números del 0 al 9, el punto (.), El guión (-) y el subrayado (*). El estándar sugiere además que los nombres de archivo no comiencen con un guión.

Verifying Input
Verificación de entrada

A general rule of good programming is that if a program accepts input, it must be able to deal with anything it receives. This usually means that input must be carefully screened to ensure that only valid input is accepted for further processing. We saw an example of this in the previous chapter when we studied the read command. One script contained the following test to verify a menu selection:
Una regla general de una buena programación es que si un programa acepta entradas, debe poder manejar cualquier cosa que reciba. Por lo general, esto significa que la entrada debe examinarse cuidadosamente para garantizar que solo se acepte la entrada válida para su procesamiento posterior. Vimos un ejemplo de esto en el capítulo anterior cuando estudiamos el comando de lectura. Un script contenía la siguiente prueba para verificar una selección de menú:

```
[[ $REPLY =~ ^[0-3]$ ]]
```

This test is very specific. It will return a zero exit status only if the string entered by the user is a numeral in the range of zero to three. Nothing else will be accepted. Sometimes these kinds of tests can be challenging to write, but the effort is necessary to produce a high-quality script.
Esta prueba es muy específica. Devolverá un estado de salida cero solo si la cadena ingresada por el usuario es un número en el rango de cero a tres. No se aceptará nada más. A veces, este tipo de pruebas puede ser difícil de escribir, pero el esfuerzo es necesario para producir un guión de alta calidad.

Design Is a Function of Time
El diseño es una función del tiempo

> When I was a college student studying industrial design, a wise professor stated that the amount of design on a project was determined by the amount of time given to the designer. If you were given five minutes to design a device “that kills flies,” you designed a flyswatter. If you were given five months, you might come up with a laser-guided “antifly system” instead.
> Cuando era un estudiante universitario que estudiaba diseño industrial, un profesor sabio dijo que la cantidad de diseño en un proyecto estaba determinada por la cantidad de tiempo que se le daba al diseñador. Si te dieran cinco minutos para diseñar un dispositivo "que mata moscas", diseñaste un matamoscas. Si le dieran cinco meses, puede que se le ocurra en su lugar un "sistema antifly" guiado por láser.
> The same principle applies to programming. Sometimes a “quick-and-dirty” script will do if it’s going to be used once and only by the programmer. That kind of script is common and should be developed quickly to make the effort economical. Such scripts don’t need a lot of comments and defensive checks. On the other hand, if a script is intended for production use, that is, a script that will be used over and over for an important task or by multiple users, it needs much more careful development.
> El mismo principio se aplica a la programación. A veces, un script "rápido y sucio" servirá si el programador lo va a usar una sola vez. Ese tipo de guión es común y debe desarrollarse rápidamente para que el esfuerzo sea económico. Estos scripts no necesitan muchos comentarios ni controles defensivos. Por otro lado, si un script está diseñado para uso en producción, es decir, un script que se usará una y otra vez para una tarea importante o por varios usuarios, necesita un desarrollo mucho más cuidadoso.

Testing
Pruebas

Testing is an important step in every kind of software development, including scripts. There is a saying in the open source world, “release early, release often,” that reflects this fact. By releasing early and often, software gets more exposure to use and testing. Experience has shown that bugs are much easier to find, and much less expensive to fix, if they are found early in the development cycle.
Las pruebas son un paso importante en todo tipo de desarrollo de software, incluidos los scripts. Hay un dicho en el mundo del código abierto, "publique temprano, publique con frecuencia", que refleja este hecho. Al lanzarse temprano y con frecuencia, el software se expone más al uso y las pruebas. La experiencia ha demostrado que los errores son mucho más fáciles de encontrar y mucho menos costosos de corregir si se detectan al principio del ciclo de desarrollo.

In Chapter 26, we saw how stubs can be used to verify program flow. From the earliest stages of script development, they are a valuable technique to check the progress of our work.
En el Capítulo 26, vimos cómo se pueden usar los stubs para verificar el flujo del programa. Desde las primeras etapas del desarrollo del guión, son una técnica valiosa para comprobar el progreso de nuestro trabajo.
Let's look at the file-deletion problem shown previously and see how this could be coded for easy testing. Testing the original fragment of code would be dangerous since its purpose is to delete files, but we could modify the code to make the test safe.
Veamos el problema de eliminación de archivos que se mostró anteriormente y veamos cómo se puede codificar para facilitar las pruebas. Probar el fragmento de código original sería peligroso ya que su propósito es eliminar archivos, pero podríamos modificar el código para que la prueba sea segura.

```
if [[ -d $dir_name ]]; then
    if cd $dir_name; then
        echo rm * # TESTING
    else
        echo "cannot cd to '$dir_name'" >&2
        exit 1
    fi
else
    echo "no such directory: '$dir_name'" >&2
    exit 1
fi
exit # TESTING
```

Because the error conditions already output useful messages, we don't have to add any. The most important change is placing an echo command just before the rm command to allow the command and its expanded argument list to be displayed, rather than the command actually being executed. This change allows safe execution of the code. At the end of the code fragment, we place an exit command to conclude the test and prevent any other part of the script from being carried out. The need for this will vary according to the design of the script.
Debido a que las condiciones de error ya generan mensajes útiles, no tenemos que agregar ninguno. El cambio más importante es colocar un comando echo justo antes del comando rm para permitir que se muestre el comando y su lista ampliada de argumentos, en lugar de que el comando se esté ejecutando realmente. Este cambio permite la ejecución segura del código. Al final del fragmento de código, colocamos un comando de salida para concluir la prueba y evitar que se lleve a cabo cualquier otra parte del script. La necesidad de esto variará según el diseño del guión.
We also include some comments that act as "markers" for our testrelated changes. These can be used to help find and remove the changes when testing is complete.
También incluimos algunos comentarios que actúan como "marcadores" para nuestros cambios relacionados con la prueba. Estos se pueden utilizar para ayudar a encontrar y eliminar los cambios cuando se completen las pruebas.

Test Cases
Casos de prueba

To perform useful testing, it's important to develop and apply good test cases. This is done by carefully choosing input data or operating conditions that reflect edge and corner cases. In our code fragment (which is simple), we want to know how the code performs under three specific conditions.
Para realizar pruebas útiles, es importante desarrollar y aplicar buenos casos de prueba. Esto se hace eligiendo cuidadosamente los datos de entrada o las condiciones de operación que reflejan los casos de borde y esquina. En nuestro fragmento de código (que es simple), queremos saber cómo funciona el código en tres condiciones específicas.

- dir_name contains the name of an existing directory.
  dir_name contiene el nombre de un directorio existente.
- dir_name contains the name of a nonexistent directory.
  dir_name contiene el nombre de un directorio inexistente
- dir_name is empty.
  dir_name está vacio

By performing the test with each of these conditions, good test coverage is achieved.
Al realizar la prueba con cada una de estas condiciones, se logra una buena cobertura de prueba.

Just as with design, testing is a function of time, as well. Not every script feature needs to be extensively tested. It's really a matter of determining what is most important. Since it could be so potentially destructive if it malfunctioned, our code fragment deserves careful consideration during both its design and testing.
Al igual que con el diseño, las pruebas también son una función del tiempo. No es necesario probar exhaustivamente todas las funciones de los scripts. Realmente se trata de determinar qué es lo más importante. Dado que podría ser potencialmente tan destructivo si no funcionara correctamente, nuestro fragmento de código merece una consideración cuidadosa durante su diseño y prueba.

Debugging
Depuración.

If testing reveals a problem with a script, the next step is debugging. "A problem" usually means that the script is, in some way, not performing to the programmer's expectations. If this is the case, we need to carefully determine exactly what the script is actually doing and why. Finding bugs can sometimes involve a lot of detective work.
Si las pruebas revelan un problema con un script, el siguiente paso es depurar. "Un problema" generalmente significa que el guión, de alguna manera, no está funcionando según las expectativas del programador. Si este es el caso, debemos determinar cuidadosamente qué está haciendo realmente el script y por qué. Encontrar errores a veces puede implicar mucho trabajo de detective.

A well-designed script will try to help. It should be programmed defensively to detect abnormal conditions and provide useful feedback to the user. Sometimes, however, problems are quite strange and unexpected,

and more involved techniques are required.
Un guión bien diseñado intentará ayudar. Debe programarse de forma defensiva para detectar condiciones anormales y proporcionar información útil al usuario. A veces, sin embargo, los problemas son bastante extraños e inesperados y se requieren técnicas más complicadas.

Finding the Problem Area
Encontrar el área del problema

In some scripts, particularly long ones, it is sometimes useful to isolate the area of the script that is related to the problem. This won't always be the actual error, but isolation will often provide insights into the actual cause.
En algunos scripts, particularmente los largos, a veces es útil aislar el área del script que está relacionada con el problema. Este no siempre será el error real, pero el aislamiento a menudo proporcionará información sobre la causa real.

One technique that can be used to isolate code is "commenting out" sections of a script. For example, our file deletion fragment could be modified to determine whether the removed section was related to an error.
Una técnica que se puede utilizar para aislar el código es "comentar" secciones de un script. Por ejemplo, nuestro fragmento de eliminación de archivos podría modificarse para determinar si la sección eliminada estaba relacionada con un error.

```
if [[ -d $dir_name ]]; then
    if cd $dir_name; then
        rm *
    else
        echo "cannot cd to '$dir_name'" >&2
        exit 1
    fi
# else
#   echo "no such directory: '$dir_name'" >&2
#   exit 1
fi
```

By placing comment symbols at the beginning of each line in a logical section of a script, we prevent that section from being executed. Testing can then be performed again to see whether the removal of the code has any impact on the behavior of the bug.
Al colocar símbolos de comentario al principio de cada línea en una sección lógica de un script, evitamos que esa sección se ejecute. Las pruebas se pueden volver a realizar para ver si la eliminación del código tiene algún impacto en el comportamiento del error.

Tracing
Rastreo

Bugs are often cases of unexpected logical flow within a script. That is, portions of the script either are never being executed or are being executed in the wrong order or at the wrong time. To view the actual

flow of the program, we use a technique called tracing.
Los errores suelen ser casos de flujo lógico inesperado dentro de un script. Es decir, partes del script nunca se ejecutan o se ejecutan en el orden incorrecto o en el momento incorrecto. Para ver el flujo real del programa, usamos una técnica llamada rastreo.

One tracing method involves placing informative messages in a script that display the location of execution. We can add messages to our code fragment.
Un método de rastreo implica colocar mensajes informativos en un script que muestre la ubicación de ejecución. Podemos agregar mensajes a nuestro fragmento de código.

```
echo "preparing to delete files" >&2
if [[ -d $dir_name ]]; then
    if cd $dir_name; then
        echo "deleting files" >&2
        rm *
    else
        echo "cannot cd to '$dir_name'" >&2
        exit 1
    fi
else
    echo "no such directory: '$dir_name'" >&2
    exit 1
fi
echo "file deletion complete" >&2
```

We send the messages to standard error to separate them from normal output. We also do not indent the lines containing the messages, so it is easier to find when it's time to remove them.
Enviamos los mensajes a error estándar para separarlos de la salida normal. Tampoco aplicamos sangría a las líneas que contienen los mensajes, por lo que es más fácil encontrar cuándo es el momento de eliminarlos.

Now when the script is executed, it's possible to see that the file deletion has been performed.
Ahora, cuando se ejecuta el script, es posible ver que se ha realizado la eliminación del archivo.

```
[me@linuxbox ~]$ deletion-script
preparing to delete files
deleting files
file deletion complete
[me@linuxbox ~]$
```

bash also provides a method of tracing, implemented by the -x option and the set command with the -x option. Using our earlier trouble script, we can activate tracing for the entire script by adding the -x option to the first line.
bash también proporciona un método de rastreo, implementado por la opción -x y el comando set con la opción -x. Usando nuestro script de problemas anterior, podemos activar el rastreo para todo el script agregando la opción -x a la primera línea.

```
#!/bin/bash -x

# trouble: script to demonstrate common errors

number=1

if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
```

When executed, the results look like this:
Cuando se ejecuta, los resultados se ven así:

```
[me@linuxbox ~]$ trouble
+ number=1
+ '[' 1 = 1 ']'
+ echo 'Number is equal to 1.'
Number is equal to 1.
```

With tracing enabled, we see the commands performed with expansions applied. The leading plus signs indicate the display of the trace to distinguish them from lines of regular output. The plus sign is the default character for trace output. It is contained in the PS4 (prompt string 4) shell variable. The contents of this variable can be adjusted to make the prompt more useful.
Con el rastreo habilitado, vemos los comandos realizados con las expansiones aplicadas. Los signos más iniciales indican la visualización de la traza para distinguirlos de las líneas de salida regular. El signo más es el carácter predeterminado para la salida de seguimiento. Está contenido en la variable de shell de PS4 (cadena de solicitud 4). El contenido de esta variable se puede ajustar para que la solicitud sea más útil.

Here, we modify the contents of the variable to include the current line number in the script where the trace is performed. Note that single quotes are required to prevent expansion until the prompt is actually used.
Aquí, modificamos el contenido de la variable para incluir el número de línea actual en el script donde se realiza el rastreo. Tenga en cuenta que se requieren comillas simples para evitar la expansión hasta que se utilice realmente la solicitud.

```
[me@linuxbox ~]$ export PS4='$LINENO + '
[me@linuxbox ~]$ trouble
5 + number=1
7 + '[' 1 = 1 ']'
8 + echo 'Number is equal to 1.'
Number is equal to 1.
```

To perform a trace on a selected portion of a script, rather than the entire script, we can use the set command with the -x option.
Para realizar un seguimiento en una parte seleccionada de un script, en lugar de todo el script, podemos usar el comando set con la opción -x.

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

set -x # Turn on tracing
if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
fi
set +x # Turn off tracing
```

We use the set command with the -x option to activate tracing and the +x option to deactivate tracing. This technique can be used to examine multiple portions of a troublesome script.
Usamos el comando set con la opción -x para activar el rastreo y la opción + x para desactivar el rastreo. Esta técnica se puede utilizar para examinar múltiples partes de un script problemático.

Examining Values During Execution
Examinar valores durante la ejecución

It is often useful, along with tracing, to display the content of variables to see the internal workings of a script while it is being executed. Applying additional echo statements will usually do the trick.

```
#!/bin/bash

# trouble: script to demonstrate common errors

number=1

echo "number=$number" # DEBUG
set -x # Turn on tracing
if [ $number = 1 ]; then
    echo "Number is equal to 1."
else
    echo "Number is not equal to 1."
```

```
fi
set +x # Turn off tracing
```

In this trivial example, we simply display the value of the variable number and mark the added line with a comment to facilitate its later identification and removal. This technique is particularly useful when watching the behavior of loops and arithmetic within scripts.
En este ejemplo trivial, simplemente mostramos el valor del número de variable y marcamos la línea agregada con un comentario para facilitar su posterior identificación y eliminación. Esta técnica es particularmente útil cuando se observa el comportamiento de los bucles y la aritmética dentro de los scripts.

Summing Up
Resumen

In this chapter, we looked at just a few of the problems that can crop up during script development. Of course, there are many more. The techniques described here will enable finding most common bugs. Debugging is a fine art that is developed through experience, both in knowing how to avoid bugs (testing constantly throughout development) and in finding bugs (effective use of tracing).
En este capítulo, analizamos solo algunos de los problemas que pueden surgir durante el desarrollo del script. Por supuesto, hay muchos más. Las técnicas descritas aquí permitirán encontrar los errores más comunes. La depuración es un arte que se desarrolla a través de la experiencia, tanto para saber cómo evitar errores (probar constantemente durante el desarrollo) como para encontrar errores (uso efectivo del rastreo).