En este capítulo, usted va a ver una de las características que hacen de Perl uno de los mejores lenguajes del mundo de programación imperativa: Hashes En los viejos tiempos, los llamábamos "arrays asociativos". Pero la comunidad de Perl decidió por 1995 que eran muchas letras al escribirlas y muchas silabas al pronunciarlo, entones cambiaron el nombre a "Hashes"

Aunque los Hashes son una útil y poderosa característica, es probable que hallas usado otros lenguajes poderosos durante años sin haber escuchado sobre los Hashes. Pero usted va a usar hashes en casi todos los programas en Perl que escriba de ahora en adelante.

¿ Que es un Hash ?

Un hash es una estructura de datos, no es muy diferente a un array, que puede almacenar cualquier número de valores y obtenerlos cuando quiera. Pero en lugar de indexar los valores por un numero, como lo hacemos con los arrays, vamos a indexar los valores por un nombre. Esto significa, que los indices (a los que vamos a llamar llaves) no son números, en lugar de esto vamos a usar cadenas arbitrarias únicas.

Primero que todo, las llaves son cadenas de caracteres, entonces, en lugar de usar el numero 3 para obtener un elemento de un array, vamos a acceder al elemento del hash llamado `wilma`.

Estas llaves, son cadenas arbitrarias - usted puede usar cualquier expresión de cadena para la llave de un hash. Y estas son cadenas únicas - de igual manera que en un array tenemos un único elemento numérico 3, aquí vamos a tener un único elemento del hash llamado `wilma`.

Otra forma de pensar en un hash, es compararlo con un barril de datos, donde cada pieza de datos tiene una etiqueta. Tu puedes llegar al barril y sacar una etiqueta y mirar que pieza de datos contiene. Pero no va a ser el "primer" elemento del barril. En un array, comenzamos desde el elemento 0, seguido del elemento 1, el elemento 2 y así. Pero en un hash, no hay un orden fijo, no hay un primer elemento. Es solo una colección de pares llave-valor.

figs/hashes.png

figs/barril.png

Las llaves y los valores son un conjunto de valores escalares ordinarios, pero las llaves son siempre convertidas a cadenas. Entonces, si usas la expresión numérica `50/20` como una llave, esto se va a convertir en una cadena de tres caracteres "2.5".

Como es usual en Perl,"no existen los límites innecesarios", esto también aplica a un hash, un hash puede ser de cualquier tamaño, desde un hash vacío con cero pares llave-valor, hasta cualquier cantidad de valores con los que pueda llena su memoria.

Algunas implementaciones de hashes (como originalmente era en el lenguaje awk) cuando los hashes son de gran tamaño tienden a ser lentos. Este no es el caso en Perl, en Perl es un algoritmo eficiente, bueno y escalable. Entonces, si un hash tiene un solo árbol de pares llave-valor, es muy rápido "buscar en el barril" y tomar uno de esos valores. Si un hash tiene tres millones de pares clave-valor, debe ser igual de rápido tomar un valor. No se asuste, los hashs grandes no muerden.

Vale la pena volver a mencionar de nuevo que las llaves de un hash son siempre únicas, aunque los valores pueden estar duplicados. Los valores de un hash pueden ser todos números, cadenas, valores `undef`, o una mezcla En efecto, cualquier valor escalar, incluyendo otros typos de datos escalares.. Pero las llaves son siempre arbitrariamente cadenas únicas.

¿ Porque usar un Hash ?

Cuando usted escucha por primera vez hablar de hashes, especialmente si usted ha vivido gran parte de su vida productiva como programador en otros lenguajes que no poseen hashes, podría preguntarse, quien querría una de estas criaturas extrañas. Pues bien, la idea general es que usted tendrá un grupo de datos relacionado con otro grupo de datos. Por ejemplo, aquí tenemos algunos

de los hashes que podrá encontrar en aplicaciones típicas de Perl:

Nombres y Apellidos

> En este caso, el primer nombre es la llave, y el apellido es el valor. Este tipo de hash requiere, por supuesto, que los nombres sean únicos. Si tiene dos personas con el mismo nombre, este ejemplo no va a funcionar para usted. Con este hash, puedes buscar a cualquiera por el nombre y obtener su apellido. Si usted busca por el nombre `Walter`, va a obtener el valor `Vargas`.

Nombre de Host y dirección IP

> Usted debe saber que cada computadora en Internet tiene un nombre de host y una dirección IP. Esto es porque a las máquinas les gusta trabajar con números, pero a los humanos se les hace mas fácil recordar nombres. Los nombres de hosts son cadenas únicas, entonces pueden usarse para hacer este hash. Con este hash puedes buscar por el nombre de host y obtener la dirección IP correspondiente.

Palabras, contar el número de veces que una palabra aparece.

> La idea aquí es, que si usted quiere saber cuantas veces aparece una palabra en un documento dado. Puede hacer un hashs que contenga como llaves del hash a las palabras y como valor el número de repeticiones de la palabra.

Cédula y Nombre

> Este es un ejemplo común, ya sabemos que los números de cédulas son cadenas únicas, entonces podemos tener un hash que contenga como llave el número de cédula y como valor el nombre de la persona.

Acceder al elemento de un Hash

Para acceder al elemento de un hash usamos la siguiente sintaxis:

```
$hash{$alguna_llave_del_hash}
```

Es bastante similar a la sintaxis que usamos para acceder a un array, pero en este caso vamos a usar llaves en lugar de corchetes alrededor del subíndice (llave). Aquí daremos un vistazo dentro de la mente de Larry Wall: Larry decía, vamos a usar llaves en lugar de corchetes porque estamos haciendo algo mas sofisticado que acceder a un array, entonces debemos usar un signo de puntuación mas sofisticado.

Podemos asignar los valores de la siguiente manera:

```
$apellido{"fred"}       = "flintstone";
$apellido{"barney"}     = "rubble";
$apellido{"hugo"}       = "Chávez";
```

Con el hash anterior, podríamos usar el siguiente código para accederlo:

```
foreach $persona (qw< fred barney hugo >){
    print "He oído algo sobre $persona $apellido{$persona}.\n";
}
```

El nombre del hash es de igual manera otro identificador de Perl (letras, dígitos y pisos bajos, pero no puede iniciar con un número). Y son de un espacio de nombres separado, lo que significa que no hay conexión entre la variable `$apellido{"walter"}`, la subrutina `&apellido` o la variable `$apellido`.

Por su puesto, la llave de un hash puede ser una expresión, no solo cadenas literales o variables escalares simples. Por ejemplo:

```
$foo = "bar";
```

```
print $apellido{ $foo . "ney"};     # Imprime "rubble"
```

Cuando guardamos algo en un elemento que ya existía en el hash, este sobre escribe al valor anterior:

```
$apellido{"fred"} = "astaire";
```

Es igual a lo que pasa con los arrays y los escalares.

Si accede a un elemento que no se encuentra en el hash va a obtener `undef`:

```
$granito = $apellido{"larry"};  # No hay larry: undef
```

Hash como un todo

Para referirse a un hash completo, usamos el signo de porcentaje (%) como sígil. Entonces, el hash que hemos estado usando en las últimas páginas actualmente se llama `%apellido`.

Por conveniencia, un hash puede convertirse en una lista, y luego volver a convertirlo en un hash. Asignando un hash en contexto de lista, donde la lista esta conformada por un conjunto de pares llave-valor.

```
%some_hash = (
    "foo",   35,      "bar",   12.4, 2.5, "hello",
    "wilma", 1.72e30, "betty", "bye\n"
  )
```

El valor de un hash (en contexto de lista) es una lista simple de pares llave-valor:

```
@any_array = %some_hash;
```

A esto le llamamos *desenrollar* el hash, básicamente convertimos el hash en una lista de pares llave-valor. Como era de esperarse, los pares no están necesariamente en el mismo orden de la lista original.

```
print "@any_array\n";
    # vamos a obtener algo como esto:
    # betty bye (una nueva linea) wilma 1.72e+30 foo 35 2.5 hello bar
12.4
```

El orden es algo confuso porque Perl mantiene los pares llave-valor en el orden que sea mas conveniente para Perl, de manera que pueda buscarse cualquier item rápidamente. Se usa un hash cuando no nos importa el orden de los elementos, o tenemos una forma fácil de ordenarlos como queramos.

Asignación de Hash

Esto no es algo común, pero usted puede copiar un hash a otro con la siguiente sintaxis obvia:

```
%new_hash = %old_hash;
```

Esto es mas complejo para Perl de lo que parece, mientras en otros lenguajes como C o Pascal, es tan simple como copiar un bloque de memoria, las estructuras de datos en Perl sin mas complejas. Entonces, esta linea de código le dice a Perl que desenrolle a %old_hash en una lista de pares llaves-valor, y que arme el hash %new_hash desde la lista de pares llave-valor.

Es mas común hacer otro tipo de transformaciones con un hash. Por ejemplo, podemos obtener el

inverso de un hash así:

```
%inverse_hash = reverse %any_hash;
```

La Gran Flecha

Cuando asignamos una lista a un hash, algunas veces no es obvio que elementos son llaves y que elementos son valores. Por ejemplo, en esta asignación, los humanos debemos ir contando por la lista diciendo: "llave, valor, llave, valor, llave, valor ...", para poder determinar si 2.5 es una llave o un valor:

```
%some_hash = (
    "foo",   35,      "bar",   12.4, 2.5, "hello",
    "wilma", 1.72e30, "betty", "bye\n"
);
```

¿ No seria agradable que Perl nos diera una manera emparejar las llaves y los valores en contexto de lista ?, Larry también pensaba esto y por eso invento a la flecha mayor (=>). En Perl, es simplemente un hechizo para representar una coma, por lo que algunas veces se le llama "la coma gorda". En la gramática de Perl, cada vez que usted necesite una coma (,), puedes usar la flecha grande, esto va a ser lo mismo para Perl. Por ejemplo:

```
my %last_name = ( # Un hash también puede ser una variable léxica.
    "fred" => "flintstone",
    "dino" => undef,
    "barney" => "rubble",
    "betty" => "rubble",
);
```

Como puede ver, es fácil determinar en el ejemplo anterior, que elementos son valores y que elementos son llaves.

Operadores de Hash

Naturalmente, hay algunas funciones útiles para poder trabajar con hashs.

Las funciones keys y values

La función `keys` obtiene una lista de todas las llaves de un hash, mientras que la función `values` obtiene los valores correspondientes. Si el hash no tiene elementos, ambas funciones retornan una lista vacía.

```
my %hash = ("a" => 1, "b" => 2, "c" => 3);
my @k = keys %hash;
my @v = values %hash;
```

En el ejemplo anterior @k va a contener "a", "b", y "c", y @v va a contener 1, 2, y 3 en el mismo orden. Recuerde que Perl no mantiene el orden de un hash. Pero cual sea el orden que tenga la lista de llaves va a ser el orden correspondiente de la lista de valores.

En contexto escalar estas funciones retornan la cantidad de elementos en el hash. Ejemplo:

```
my $count = keys %hash;
```

En un ciclo `while`, lo podemos usar como expresión en el contexto booleano, por ejemplo:

```
if (%hash){
```

```
    print "Esto es un valor verdadero\n";
}
```

La función each

Si usted quiere iterar el hash completo, una de las maneras usuales es usar la función `each`, que retorna un par llave-valor como una lista de dos elementos. En cada evaluación de esta función sobre el mismo hash, va a retornar el siguiente par llave-valor, hasta que todos los elementos sean accedidos. Cuando no hay mas pares, `each` retorna una lista vacía.

En la práctica, la única manera de usar `each` es en un ciclo while, por ejemplo:

```
while ( ($key, $value) = each %hash ) {
    print "$key => $value\n";
}
```

Por supuesto, `each` retorna los pares llave-valor desordenadamente. Si necesita recorrer el hash en orden, simplemente ordena las llaves, por ejemplo:

```
foreach $key ( sort keys %hash) {
    $value = $hash{$key};
    print "$key => $value \n";
}
```

La función exists

Para verificar que una llave exista en un hash, puede usar la función `exists`, que retorna un valor verdadero si la llave existe en el hash, de lo contrario retorna falso:

```
if (exists $books{"dino"}) {
    print "Hey, there's a library card for dino!\n";
}
```

La función delete

La función `delete` remueve una llave de el hash. Si no existe la llave, la función termina, y en ese caso causa una advertencia o un error.

```
my $person = "larry";
delete $books{$person};
```

Note que esto no es lo mismo que guardar un elemento como `undef` en el hash. En efecto, si chequea el hash con `exists($books{"betty"})` va a obtener verdadero, luego de usar `delete`, la llave no puede existir en el hash, pero después de guardar `undef` la llave va a existir en el hash.

Interpolación de un elemento en un Hash

Puedes interpolar un elemento simple de un hash colocándolo entre dobles comillas.

```
foreach $person (sort keys %books) {
    if ($books{$person}) {
        print "$person has $books{$person} items\n";
    }
}
```

Pero no es posible interpolar el hash completo, si tratamos de usar `"%books"` simplemente obtendremos seis caracteres de (literalmente) %books.

El hash %ENV

Su programa, como cualquier otro programa, corre en un ambiente específico, de modo que el programa puede mirar en el ambiente para obtener información de sus alrededores. Perl guarda esa información en el hash `%ENV`. Por ejemplo, podemos ver el PATH de la siguiente forma:

```
print "PATH is $ENV{PATH}\n";
```

Dependiendo de su sistema operativo particular, va a obtener algo parecido a esto:

```
    PATH is
/Users/elsanto/.vim/bin/:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
```

Ejercicios

1. Escriba un programa que pregunte al usuario por un nombre e imprima correctamente su apellido. Use los nombres de las personas que usted conozca, o use la tabla siguiente:

```
fred    flinstone
barney  rubble
wilma   flinstone
```

2. Escriba un programa que lea una serie de palabras (una palabra por linea) hasta el final de la entrada (End-Of-Input), e imprima un resumen de cuantas veces fue vista cada palabra.

3. Escriba un programa que liste todas las llaves de los valores en el hash `%ENV`. Imprima los resultados en dos columnas en orden ASCIIbetico. Una vez que tenga el programa funcionando, intente definir nuevas variables de entorno y asegúrese que estas variables están en la salida de su programa.