shell实现netmask掩码和cidr掩码位转换

Network IP Netmask CIDR

在写一个脚本时需要实现掩码位和掩码之间的转换，想简单的通过shell 实现，在openwrt程序上刚好有此脚本，内容如下：

```
#!/bin/bash
# code from www.361way.com
mask2cdr ()
{
   # Assumes there's no "255." after a non-255 byte in the mask
   local x=${1##*255.}
   set -- 0^^^128^192^224^240^248^252^254^ $(( (${#1} - ${#x})*2 )) ${x%%.*}
   x=${1%%$3*}
   echo $(( $2 + (${#x}/4) ))
}
cdr2mask ()
{
   # Number of args to shift, 255..255, first non-255 byte, zeroes
   set -- $(( 5 - ($1 / 8) )) 255 255 255 255 $(( (255 << (8 - ($1 % 8))) & 255 )) 0 0
 0
   [ $1 -gt 1 ] && shift $1 || shift
   echo ${1-0}.${2-0}.${3-0}.${4-0}
}
# examples:
mask2cdr 255.255.192.0
cdr2mask 18
```

上面的代码看起来比较玄妙，其具体解释可以看下英文解释。

mask2cdr()

> To get the CIDR prefix from a dot-decimal netmask like this one:

```
255.255.192.0
```

> you first have to convert the four octets to binary and then count the most significant bits (i.e. the number of leading ones):

```
11111111.11111111.11000000.00000000 # 18 ones = /18 in CIDR
```

> This function does that rather creatively. First, we strip off all of the leading 255 octets (i.e. the octets that are all ones in binary) and store the results in variable x:

```
local x=${1##*255.}
```

> This step uses parameter expansion, which the entire script relies on pretty heavily. If we continue with our example netmask of 255.255.192.0, we now have the following values:

```
$1: 255.255.192.0
$x: 192.0
```

> Next we set three variables: $1, $2, and $3. These are called positional parameters; they are much like ordinary named variables but are typically set when you pass arguments to a script or function. We can set the values directly using set --, for example:

```
set -- foo bar # $1 = foo, $2 = bar
```

> I prefer using named variables over positional parameters since it makes scripts easier to read and debug, but the end result is the same. We set $1 to:

```
0^^^128^192^224^240^248^252^254^
```

> This is really just a table to convert certain decimal values to binary and count the number of 1 bits. We'll come back to this later.

> We set $2 to

```
$(( (${#1} - ${#x})*2 ))
```

> This looks complex, but it is really just counting the number of 1 bits we stripped off in the first command. It breaks down to this:

```
(number of chars in $1 - number of chars in $x) * 2
```

which in our case works out to

```
(13 - 5) * 2 = 16
```

> We stripped off two octets so we get 16. Makes sense.

> We set $3 to:

```
${x%%.*}
```

which is the value of $x with everything after the first . stripped off. In our case, this is 192.

> We need to convert this number to binary and count the number of 1 bits in it, so let's go back to our "conversion table." We can divide the table into equal chunks of four characters each:

```
0^^^ 128^ 192^ 224^ 240^ 248^ 252^ 254^
```

In binary, the above numbers are:

```
00000000 10000000 11000000 11100000 11110000 11111000 11111100 11111110
# 0 ones 1 one    2 ones   3 ones   ...
```

> If we count from the left, each four-character block in the table corresponds to an additional 1 bit in binary. We're trying to convert 192, so let's first lop off the rightmost part of the table, from 192 on, and store it in x:

```
x=${1%%$3*}
```

The value of $x is now

```
0^^^128^
```

which contains two four-character blocks, or two 1 bits in binary.

> Now we just need to add up the 1 bits from our leading 255 octets (16 total, stored in variable $2) and the 1 bits from the previous step (2 total):

```
echo $(( $2 + (${#x}/4) ))
```

where

```
${#x}/4
```

is the number of characters in $x divided by four, i.e. the number of four-character blocks in $x.

Output:

```
18
```

cdr2mask()

> Let's keep running with our previous example, which had a CIDR prefix of 18.

> We use set -- to set positional parameters $1 through $9:

```
$1: $(( 5 - ($1 / 8) ))  # 5 - (18 / 8) = 3 [integer math]
$2: 255
$3: 255
$4: 255
$5: 255
$6: $(( (255 << (8 - ($1 % 8))) & 255 ))  # (255 << (8 - (18 % 8))) & 255 = 192
$7: 0
$8: 0
$9: 0
```

> Let's examine the formulas used to set $1 and $6 a little closer. $1 is set to:

```
$(( 5 - ($1 / 8) ))
```

> The maximum and minimum possible values for a CIDR prefix are 32 for netmask

```
11111111.11111111.11111111.11111111
```

and 0 for netmask

```
00000000.00000000.00000000.00000000
```

> The above formula uses integer division, so the possible results range from 1 to 5:

```
5 - (32 / 8) = 1
5 - ( 0 / 8) = 5
```

$6 is set to:

```
$(( (255 << (8 - ($1 % 8))) & 255 ))
```

> Let's break this down for our example CIDR prefix of 18. First we take the modulus and do some subtraction:

```
8 - (18 % 8) = 6
```

> Next we bitwise shift 255 by this value:

```
255 << 6
```

> This is the same as pushing six 0 bits onto the end of 255 in binary:

```
11111111000000
```

> Finally, we bitwise AND this value with 255:

```
11111111000000 &
00000011111111  # 255
```

which gives

```
00000011000000
```

or simply

```
11000000
```

> Look familiar? This is the third octet in our netmask in binary:

```
11111111.11111111.11000000.00000000
                  ^------^
```

In decimal, the value is 192.

> Next we shift the positional parameters based on the value of $1:

```
[ $1 -gt 1 ] && shift $1 || shift
```

> In our case, the value of $1 is 3, so we shift the positional parameters 3 to the left. The previous value of $4 becomes the new value of $1, the previous value of $5 becomes the value of $2, and so on:

```
$1: 255
$2: 255
$3: 192
$4: 0
$5: 0
$6: 0
```

> These values should look familiar: they are the decimal octets from our netmask (with a couple of extra zeros tacked on at the end). To get the netmask, we simply print out the first four with dots in between them:

```
echo ${1-0}.${2-0}.${3-0}.${4-0}
```

> The -0 after each parameter says to use 0 as the default value if the parameter is not set.

Output:

```
255.255.192.0
```