

字符串操作

网道 (WangDoc.com) , 互联网文档计划

本章介绍 Bash 字符串操作的语法。

目录 [隐藏]

1. 字符串的长度
2. 子字符串
3. 搜索和替换
4. 改变大小写

1. 字符串的长度

获取字符串长度的语法如下。

```
${#varname}
```

下面是一个例子。

```
$ myPath=/home/cam/book/long.file.name
$ echo ${#myPath}
29
```

大括号 `{}` 是必需的, 否则 Bash 会将 `$#` 理解成脚本的参数个数, 将变量名理解成文本。

📄 Bash 脚本教程

- 📄 1. 简介
- 📄 2. 基本语法
- 📄 3. 模式扩展
- 📄 4. 引号和转义
- 📄 5. 变量
- 📄 6. 字符串操作
- 📄 7. 算术运算
- 📄 8. 操作历史
- 📄 9. 行操作
- 📄 10. 目录堆栈
- 📄 11. 脚本入门
- 📄 12. read 命令
- 📄 13. 条件判断
- 📄 14. 循环
- 📄 15. 函数
- 📄 16. 数组
- 📄 17. set 命令, shopt 命令

```
$ echo $#myvar
0myvar
```

上面例子中，Bash 将 `##` 和 `myvar` 分开解释了。

2. 子字符串

字符串提取子串的语法如下。

```
${varname:offset:length}
```

上面语法的含义是返回变量 `$varname` 的子字符串，从位置 `offset` 开始（从 0 开始计算），长度为 `length`。

```
$ count=frogfootman
$ echo ${count:4:4}
foot
```

上面例子返回字符串 `frogfootman` 从4号位置开始的长度为4的子字符串 `foot`。

这种语法不能直接操作字符串，只能通过变量来读取字符串，并且不会改变原始字符串。

```
# 报错
$ echo ${"hello":2:3}
```


上面例子中，“`hello`”不是变量名，导致 Bash 报错。


如果省略 `length`，则从位置 `offset` 开始，一直返回到字符串的结尾。


```
$ count=frogfootman
$ echo ${count:4}
footman
```

上面例子是返回变量 `count` 从4号位置一直到结尾的子字符串。

 **18.** 脚本除错

 **19.** `mktemp` 命令，`trap` 命令

 **20.** 启动环境

 **21.** 命令提示符

链接

 [本文源码](#)

 [代码仓库](#)

 [反馈](#)

如果 `offset` 为负值，表示从字符串的末尾开始算起。注意，负数前面必须有一个空格，以防止与 `${variable:-word}` 的变量的设置默认值语法混淆。这时还可以指定 `length`，`length` 可以是正值，也可以是负值（负值不能超过 `offset` 的长度）。

```
$ foo="This string is long."
$ echo ${foo: -5}
long.
$ echo ${foo: -5:2}
lo
$ echo ${foo: -5:-2}
lon
```

上面例子中，`offset` 为 `-5`，表示从倒数第5个字符开始截取，所以返回 `long.`。如果指定长度 `length` 为 `2`，则返回 `lo`；如果 `length` 为 `-2`，表示要排除从字符串末尾开始的2个字符，所以返回 `lon`。

3. 搜索和替换

Bash 提供字符串搜索和替换的多种方法。

(1) 字符串头部的模式匹配。

以下两种语法可以检查字符串开头，是否匹配给定的模式。如果匹配成功，就删除匹配的部分，返回剩下的部分。原始变量不会发生变化。

```
# 如果 pattern 匹配变量 variable 的开头，
# 删除最短匹配（非贪婪匹配）的部分，返回剩余部分
${variable#pattern}

# 如果 pattern 匹配变量 variable 的开头，
# 删除最长匹配（贪婪匹配）的部分，返回剩余部分
${variable##pattern}
```

上面两种语法会删除变量字符串开头的匹配部分（将其替换为空），返回剩下的部分。区别是一个是最短匹配（又称非贪婪匹配），另一个是最长匹配（又称贪婪匹配）。

匹配模式 `pattern` 可以使用 `*`、`?`、`[]` 等通配符。

```
$ myPath=/home/cam/book/long.file.name

$ echo ${myPath#/*/}
cam/book/long.file.name

$ echo ${myPath##/*/}
long.file.name
```

上面例子中，匹配的模式是 `/*/`，其中 `*` 可以匹配任意数量的字符，所以最短匹配是 `/home/`，最长匹配是 `/home/cam/book/`。

下面写法可以删除文件路径的目录部分，只留下文件名。

```
$ path=/home/cam/book/long.file.name

$ echo ${path##*/}
long.file.name
```

上面例子中，模式 `*/` 匹配目录部分，所以只返回文件名。

下面再看一个例子。

```
$ phone="555-456-1414"
$ echo ${phone#*-}
456-1414
$ echo ${phone##*-}
1414
```

如果匹配不成功，则返回原始字符串。

```
$ phone="555-456-1414"
$ echo ${phone#444}
555-456-1414
```

上面例子中，原始字符串里面无法匹配模式 `444`，所以原样返回。

如果要将头部匹配的部分，替换成其他内容，采用下面的写法。

```
# 模式必须出现在字符串的开头
${variable/#pattern/string}

# 示例
$ foo=JPG.JPG
$ echo ${foo/#JPG/jpg}
jpg.JPG
```

上面例子中，被替换的 **JPG** 必须出现在字符串头部，所以返回 **jpg.JPG**。

(2) 字符串尾部的模式匹配。

以下两种语法可以检查字符串结尾，是否匹配给定的模式。如果匹配成功，就删除匹配的部分，返回剩下的部分。原始变量不会发生变化。

```
# 如果 pattern 匹配变量 variable 的结尾，
# 删除最短匹配（非贪婪匹配）的部分，返回剩余部分
${variable%pattern}

# 如果 pattern 匹配变量 variable 的结尾，
# 删除最长匹配（贪婪匹配）的部分，返回剩余部分
${variable%%pattern}
```

上面两种语法会删除变量字符串结尾的匹配部分（将其替换为空），返回剩下的部分。区别是一个是最短匹配（又称非贪婪匹配），另一个是最长匹配（又称贪婪匹配）。

```
$ path=/home/cam/book/long.file.name

$ echo ${path%.*}
/home/cam/book/long.file

$ echo ${path%%.*}
/home/cam/book/long
```

上面例子中，匹配模式是 `.*`，其中 `*` 可以匹配任意数量的字符，所以最短匹配是 `.name`，最长匹配是 `.file.name`。

下面写法可以删除路径的文件名部分，只留下目录部分。

```
$ path=/home/cam/book/long.file.name

$ echo ${path%/*}
/home/cam/book
```

上面例子中，模式 `/*` 匹配文件名部分，所以只返回目录部分。

下面的写法可以替换文件的后缀名。

```
$ file=foo.png
$ echo ${file%.png}.jpg
foo.jpg
```

上面的例子将文件的后缀名，从 `.png` 改成了 `.jpg`。

下面再看一个例子。

```
$ phone="555-456-1414"
$ echo ${phone%-*}
555-456
$ echo ${phone%%-*}
555
```

如果匹配不成功，则返回原始字符串。

如果要将尾部匹配的部分，替换成其他内容，采用下面的写法。

```
# 模式必须出现在字符串的结尾
${variable/%pattern/string}

# 示例
$ foo=JPG.JPG
$ echo ${foo/%JPG/jpg}
JPG.jpg
```

上面例子中，被替换的 `JPG` 必须出现在字符串尾部，所以返回 `JPG.jpg`。

(3) 任意位置的模式匹配。

以下两种语法可以检查字符串内部，是否匹配给定的模式。如果匹配成功，就删除匹配的部分，换成其他的字符串返回。原始变量不会发生变化。

```
# 如果 pattern 匹配变量 variable 的一部分，
# 最长匹配（贪婪匹配）的那部分被 string 替换，但仅替换第
${variable/pattern/string}
```

```
# 如果 pattern 匹配变量 variable 的一部分，
# 最长匹配（贪婪匹配）的那部分被 string 替换，所有匹配都
${variable//pattern/string}
```

上面两种语法都是最长匹配（贪婪匹配）下的替换，区别是前一个语法仅仅替换第一个匹配，后一个语法替换所有匹配。

```
$ path=/home/cam/foo/foo.name

$ echo ${path/foo/bar}
/home/cam/bar/foo.name

$ echo ${path//foo/bar}
/home/cam/bar/bar.name
```

上面例子中，前一个命令只替换了第一个 `foo`，后一个命令将两个 `foo` 都替换了。

下面的例子将分隔符从 `:` 换成换行符。

```
$ echo -e ${PATH//:/'\n'}
/usr/local/bin
/usr/bin
/bin
...
```

上面例子中，`echo` 命令的 `-e` 参数，表示将替换后的字符串的 `\n` 字符，解释为换行符。

模式部分可以使用通配符。

```
$ phone="555-456-1414"
$ echo ${phone/5?4/-}
55-56-1414
```

上面的例子将 `5-4` 替换成 `-`。

如果省略了 `string` 部分，那么就相当于匹配的部分替换成空字符串，即删除匹配的部分。

```
$ path=/home/cam/foo/foo.name
$ echo ${path/./}
/home/cam/foo/foo
```

上面例子中，第二个斜杠后面的 `string` 部分省略了，所以模式 `.*` 匹配的部分 `.name` 被删除后返回。

前面提到过，这个语法还有两种扩展形式。

```
# 模式必须出现在字符串的开头
${variable/#pattern/string}

# 模式必须出现在字符串的结尾
${variable/%pattern/string}
```

4. 改变大小写

下面的语法可以改变变量的大小写。

```
# 转为大写
${varname^^}

# 转为小写
${varname,,}
```


下面是一个例子。

```
$ foo=heLLo
$ echo ${foo^^}
HELLO
$ echo ${foo,,}
hello
```

[◀ 变量](#)

[算术运算 ▶](#)

本教程采用[知识共享 署名-相同方式共享 3.0协议](#)。

分享本文      

联系: contact@wangdoc.com