

Bash 的模式扩展

网道 (WangDoc.com) , 互联网文档计划

📖 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

目录 [隐藏]

- 1. 简介
- 2. 波浪线扩展
- 3. ? 字符扩展
- 4. * 字符扩展
- 5. 方括号扩展
- 6. [start-end] 扩展
- 7. 大括号扩展
- 8. {start..end} 扩展
- 9. 变量扩展
- 10. 子命令扩展
- 11. 算术扩展
- 12. 字符类
- 13. 使用注意点
- 14. 量词语法
- 15. shopt 命令
- 16. 参考链接

1. 简介

Shell 接收到用户输入的命令以后，会根据空格将用户的输入，拆分成一个个词元（token）。然后，Shell 会扩展词元里面的特殊字符，扩展完成后才会调用相应的命令。

这种特殊字符的扩展，称为模式扩展（globbing）。其中有些用到通配符，又称为通配符扩展（wildcard expansion）。Bash 一共提供八种扩展。

- 波浪线扩展
- `?` 字符扩展
- `*` 字符扩展
- 方括号扩展
- 大括号扩展
- 变量扩展
- 子命令扩展
- 算术扩展

本章介绍这八种扩展。

Bash 是先进行扩展，再执行命令。因此，扩展的结果是由 Bash 负责的，与所要执行的命令无关。命令本身并不存在参数扩展，收到什么参数就原样执行。这一点务必需要记住。

模块扩展的英文单词是 `globbing`，这个词来自于早期的 Unix 系统有一个 `/etc/glob` 文件，保存扩展的模板。后来 Bash 内置了这个功能，但是这个名字就保留了下来。

模式扩展与正则表达式的关系是，模式扩展早于正则表达式出现，可以看作是原始的正则表达式。它的功能没有正则那么强大灵活，但是优点是简单和方便。


Bash 允许用户关闭扩展。

```
$ set -o noglob
# 或者
$ set -f
```


下面的命令可以重新打开扩展。

```
$ set +o noglob
# 或者
```

 **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

```
$ set +f
```

2. 波浪线扩展

波浪线 `~` 会自动扩展成当前用户的主目录。

```
$ echo ~  
/home/me
```

`~/dir` 表示扩展成主目录的某个子目录，`dir` 是主目录里面的一个子目录名。

```
# 进入 /home/me/foo 目录  
$ cd ~/foo
```

`~user` 表示扩展成用户 `user` 的主目录。

```
$ echo ~foo  
/home/foo
```

```
$ echo ~root  
/root
```

上面例子中，Bash 会根据波浪号后面的用户名，返回该用户的主目录。

如果 `~user` 的 `user` 是不存在的用户名，则波浪号扩展不起作用。

```
$ echo ~nonExistedUser  
~nonExistedUser
```

`~+` 会扩展成当前所在的目录，等同于 `pwd` 命令。

```
$ cd ~/foo  
$ echo ~+  
/home/me/foo
```

3. ? 字符扩展

? 字符代表文件路径里面的任意单个字符，不包括空字符。比如， `Data???` 匹配所有 `Data` 后面跟着三个字符的文件名。

```
# 存在文件 a.txt 和 b.txt
$ ls ?.txt
a.txt b.txt
```

上面命令中，? 表示单个字符，所以会同时匹配 `a.txt` 和 `b.txt`。

如果匹配多个字符，就需要多个?连用。

```
# 存在文件 a.txt、b.txt 和 ab.txt
$ ls ???.txt
ab.txt
```

上面命令中，?? 匹配了两个字符。

? 字符扩展属于文件名扩展，只有文件确实存在的前提下，才会发生扩展。如果文件不存在，扩展就不会发生。

```
# 当前目录有 a.txt 文件
$ echo ?.txt
a.txt
```

```
# 当前目录为空目录
$ echo ?.txt
?.txt
```

上面例子中，如果 `?.txt` 可以扩展成文件名，`echo` 命令会输出扩展后的结果；如果不能扩展成文件名，`echo` 就会原样输出 `?.txt`。

4. * 字符扩展

* 字符代表文件路径里面的任意数量的任意字符，包括零个字符。

```
# 存在文件 a.txt、b.txt 和 ab.txt
$ ls *.txt
a.txt b.txt ab.txt
```

上面例子中，*.txt 代表后缀名为 .txt 的所有文件。

如果想输出当前目录的所有文件，直接用 * 即可。

```
$ ls *
```

* 可以匹配空字符，下面是一个例子。

```
# 存在文件 a.txt、b.txt 和 ab.txt
$ ls a*.txt
a.txt ab.txt

$ ls *b*
b.txt ab.txt
```

注意，* 不会匹配隐藏文件（以 . 开头的文件），即 ls * 不会输出隐藏文件。

如果要匹配隐藏文件，需要写成 .* 。

```
# 显示所有隐藏文件
$ echo .*
```

如果要匹配隐藏文件，同时要排除 . 和 .. 这两个特殊的隐藏文件，可以与方括号扩展结合使用，写成 .[!..]* 。

```
$ echo .[!..]*
```

注意，* 字符扩展属于文件名扩展，只有文件确实存在的前提下才会扩展。如果文件不存在，就会原样输出。

```
# 当前目录不存在 c 开头的文件
$ echo c*.txt
c*.txt
```

上面例子中，当前目录里面没有 `c` 开头的文件，导致 `c*.txt` 会原样输出。

* 只匹配当前目录，不会匹配子目录。

```
# 子目录有一个 a.txt
# 无效的写法
$ ls *.txt

# 有效的写法
$ ls */*.txt
```

上面的例子，文本文件在子目录，`*.txt` 不会产生匹配，必须写成 `*/*.txt`。有几层子目录，就必须写几层星号。

Bash 4.0 引入了一个参数 `globstar`，当该参数打开时，允许 `**` 匹配零个或多个子目录。因此，`**/*.txt` 可以匹配顶层的文本文件和任意深度子目录的文本文件。详细介绍请看后面 `shopt` 命令的介绍。

5. 方括号扩展

方括号扩展的形式是 `[...]`，只有文件确实存在的前提下才会扩展。如果文件不存在，就会原样输出。括号之中的任意一个字符。比如，`[aeiou]` 可以匹配五个元音字母中的任意一个。

```
# 存在文件 a.txt 和 b.txt
$ ls [ab].txt
a.txt b.txt

# 只存在文件 a.txt
$ ls [ab].txt
a.txt
```

上面例子中，`[ab]` 可以匹配 `a` 或 `b`，前提是确实存在相应的文件。

方括号扩展属于文件名匹配，即扩展后的结果必须符合现有的文件路径。如果不存在匹配，就会保持原样，不进行扩展。

```
# 不存在文件 a.txt 和 b.txt
$ ls [ab].txt
ls: 无法访问 '[ab].txt': 没有那个文件或目录
```

上面例子中，由于扩展后的文件不存在，`[ab].txt` 就原样输出了，导致 `ls` 命名报错。

方括号扩展还有两种变体：`[^...]` 和 `[!...]`。它们表示匹配不在方括号里面的字符，这两种写法是等价的。比如，`[^abc]` 或 `[!abc]` 表示匹配除了 `a`、`b`、`c` 以外的字符。

```
# 存在 aaa、bbb、aba 三个文件
$ ls ?[!a]?
aba bbb
```

上面命令中，`[!a]` 表示文件名第二个字符不是 `a` 的文件名，所以返回了 `aba` 和 `bbb` 两个文件。

注意，如果需要匹配 `[` 字符，可以放在方括号内，比如 `[[aeiou]`。如果需要匹配连字号 `-`，只能放在方括号内部的开头或结尾，比如 `[-aeiou]` 或 `[aeiou -]`。

6. [start-end] 扩展

方括号扩展有一个简写形式 `[start-end]`，表示匹配一个连续的范围。比如，`[a-c]` 等同于 `[abc]`，`[0-9]` 匹配 `[0123456789]`。

```
# 存在文件 a.txt、b.txt 和 c.txt
$ ls [a-c].txt
a.txt
b.txt
c.txt
```

```
# 存在文件 report1.txt、report2.txt 和 report3.txt
$ ls report[0-9].txt
report1.txt
report2.txt
report3.txt
...
```

下面是一些常用简写的例子。

- `[a-z]`：所有小写字母。
- `[a-zA-Z]`：所有小写字母与大写字母。
- `[a-zA-Z0-9]`：所有小写字母、大写字母与数字。
- `[abc]*`：所有以 `a`、`b`、`c` 字符之一开头的文件名。
- `program.[co]`：文件 `program.c` 与文件 `program.o`。
- `BACKUP.[0-9][0-9][0-9]`：所有以 `BACKUP.` 开头，后面是三个数字的文件名。

这种简写形式有一个否定形式 `[!start-end]`，表示匹配不属于这个范围的字符。比如，`[!a-zA-Z]` 表示匹配非英文字母的字符。

```
$ ls report[!1-3].txt
report4.txt report5.txt
```

上面代码中，`[!1-3]` 表示排除1、2和3。

7. 大括号扩展

大括号扩展 `{...}` 表示分别扩展成大括号里面的所有值，各个值之间使用逗号分隔。比如，`{1,2,3}` 扩展成 `1 2 3`。

```
$ echo {1,2,3}
1 2 3

$ echo d{a,e,i,u,o}g
dag deg dig dug dog
```



```
$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
```

注意，大括号扩展不是文件名扩展。它会扩展成所有给定的值，而不管是否有对应的文件存在。

```
$ ls {a,b,c}.txt
ls: 无法访问'a.txt': 没有那个文件或目录
ls: 无法访问'b.txt': 没有那个文件或目录
ls: 无法访问'c.txt': 没有那个文件或目录
```

上面例子中，即使不存在对应的文件，`{a,b,c}` 依然扩展成三个文件名，导致 `ls` 命令报了三个错误。

另一个需要注意的地方是，大括号内部的逗号前后不能有空格。否则，大括号扩展会失效。

```
$ echo {1 , 2}
{1 , 2}
```

上面例子中，逗号前后有空格，Bash 就会认为这不是大括号扩展，而是三个独立的参数。

逗号前面可以没有值，表示扩展的第一项为空。

```
$ cp a.log{,.bak}

# 等同于
# cp a.log a.log.bak
```

大括号可以嵌套。

```
$ echo {j{p,pe}g,png}
jpg jpeg png

$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

大括号也可以与其他模式联用，并且总是先于其他模式进行扩展。

```
$ echo /bin/{cat,b*}
/bin/cat /bin/b2sum /bin/base32 /bin/base64 ... ..
```

基本等同于

```
$ echo /bin/cat;echo /bin/b*
```

上面例子中，会先进行大括号扩展，然后进行 `*` 扩展，等同于执行两条 `echo` 命令。

大括号可以用于多字符的模式，方括号不行（只能匹配单字符）。

```
$ echo {cat,dog}
cat dog
```

由于大括号扩展 `{...}` 不是文件名扩展，所以它总是会扩展的。这与方括号扩展 `[...]` 完全不同，如果匹配的文件不存在，方括号就不会扩展。这一点要注意区分。

```
# 不存在 a.txt 和 b.txt
$ echo [ab].txt
[ab].txt
```

```
$ echo {a,b}.txt
a.txt b.txt
```

上面例子中，如果不存在 `a.txt` 和 `b.txt`，那么 `[ab].txt` 就会变成一个普通的文件名，而 `{a,b}.txt` 可以照样扩展。

8. {start..end} 扩展

大括号扩展有一个简写形式 `{start..end}`，表示扩展成一个连续序列。比如，`{a..z}` 可以扩展成26个小写英文字母。

```
$ echo {a..c}
```

```
a b c
```

```
$ echo d{a..d}g
```

```
dag dbg dcg ddg
```

```
$ echo {1..4}
```

```
1 2 3 4
```

```
$ echo Number_{1..5}
```

```
Number_1 Number_2 Number_3 Number_4 Number_5
```

这种简写形式支持逆序。

```
$ echo {c..a}
```

```
c b a
```

```
$ echo {5..1}
```

```
5 4 3 2 1
```

注意，如果遇到无法理解的简写，大括号模式就会原样输出，不会扩展。

```
$ echo {a1..3c}
```

```
{a1..3c}
```

这种简写形式可以嵌套使用，形成复杂的扩展。

```
$ echo .{mp{3..4},m4{a,b,p,v}}
```

```
.mp3 .mp4 .m4a .m4b .m4p .m4v
```

大括号扩展的常见用途为新建一系列目录。

```
$ mkdir {2007..2009}-{01..12}
```

上面命令会新建36个子目录，每个子目录的名字都是“年份-月份”。

这个写法的另一个常见用途，是直接用于 `for` 循环。

```
for i in {1..4}
do
    echo $i
done
```

上面例子会循环4次。

如果整数前面有前导 0，扩展输出的每一项都有前导 0。

```
$ echo {01..5}
01 02 03 04 05

$ echo {001..5}
001 002 003 004 005
```

这种简写形式还可以使用第二个双点号
(`start..end..step`)，用来指定扩展的步长。

```
$ echo {0..8..2}
0 2 4 6 8
```

上面代码将 0 扩展到 8，每次递增的长度为 2，所以一共输出5个数字。

多个简写形式连用，会有循环处理的效果。

```
$ echo {a..c}{1..3}
a1 a2 a3 b1 b2 b3 c1 c2 c3
```

9. 变量扩展

Bash 将美元符号 \$ 开头的词元视为变量，将其扩展成变量值，详见《Bash 变量》一章。

```
$ echo $SHELL
/bin/bash
```

变量名除了放在美元符号后面，也可以放在 `${}` 里面。

```
$ echo ${SHELL}
/bin/bash
```

`${!string*}` 或 `${!string@}` 返回所有匹配给定字符串 `string` 的变量名。

```
$ echo ${!S*}
SECONDS SHELL SHELLOPTS SHLVL SSH_AGENT_PID SSH_AU
```



上面例子中，`${!S*}` 扩展成所有以 `S` 开头的变量名。

10. 子命令扩展

`$(...)` 可以扩展成另一个命令的运行结果，该命令的所有输出都会作为返回值。

```
$ echo $(date)
Tue Jan 28 00:01:13 CST 2020
```

上面例子中，`$(date)` 返回 `date` 命令的运行结果。

还有另一种较老的语法，子命令放在反引号之中，也可以扩展成命令的运行结果。

```
$ echo `date`
Tue Jan 28 00:01:13 CST 2020
```

`$(...)` 可以嵌套，比如 `$(ls $(pwd))`。

11. 算术扩展

`$((...))` 可以扩展成整数运算的结果，详见《Bash 的算术运算》一章。

```
$ echo $((2 + 2))
```

```
4
```

12. 字符类

`[:class:]` 表示一个字符类，扩展成某一类特定字符之中的一个。常用的字符类如下。

- `[:alnum:]` : 匹配任意英文字母与数字
- `[:alpha:]` : 匹配任意英文字母
- `[:blank:]` : 空格和 Tab 键。
- `[:cntrl:]` : ASCII 码 0-31 的不可打印字符。
- `[:digit:]` : 匹配任意数字 0-9。
- `[:graph:]` : A-Z、a-z、0-9 和标点符号。
- `[:lower:]` : 匹配任意小写字母 a-z。
- `[:print:]` : ASCII 码 32-127 的可打印字符。
- `[:punct:]` : 标点符号（除了 A-Z、a-z、0-9 的可打印字符）。
- `[:space:]` : 空格、Tab、LF (10) 、VT (11) 、FF (12) 、CR (13) 。
- `[:upper:]` : 匹配任意大写字母 A-Z。
- `[:xdigit:]` : 16进制字符 (A-F、a-f、0-9) 。

请看下面的例子。

```
$ echo [[:upper:]]*
```

上面命令输出所有大写字母开头的文件名。

字符类的第一个方括号后面，可以加上感叹号 `!`（或 `^`），表示否定。比如，`[![:digit:]]`（或 `^[[:digit:]]`）匹配所有非数字。

```
$ echo [![:digit:]]*
```

上面命令输出所有不以数字开头的文件名。

字符类也属于文件名扩展，如果没有匹配的文件名，字符类就会原样输出。

```
# 不存在以大写字母开头的文件
$ echo [[:upper:]]*
[[:upper:]]*
```

上面例子中，由于没有可匹配的文件，字符类就原样输出了。

13. 使用注意点

通配符有一些使用注意点，不可不知。

(1) 通配符是先解释，再执行。

Bash 接收到命令以后，发现里面有通配符，会进行通配符扩展，然后再执行命令。

```
$ ls a*.txt
ab.txt
```

上面命令的执行过程是，Bash 先将 `a*.txt` 扩展成 `ab.txt`，然后再执行 `ls ab.txt`。

(2) 文件名扩展在不匹配时，会原样输出。

文件名扩展在没有可匹配的文件时，会原样输出。

```
# 不存在 r 开头的文件名
$ echo r*
r*
```

上面代码中，由于不存在 `r` 开头的文件名，`r*` 会原样输出。

下面是另一个例子。

```
$ ls *.csv
```

```
ls: *.csv: No such file or directory
```

另外，前面已经说过，大括号扩展 `{...}` 不是文件名扩展。

(3) 只适用于单层路径。

所有文件名扩展只匹配单层路径，不能跨目录匹配，即无法匹配子目录里面的文件。或者说，`?` 或 `*` 这样的通配符，不能匹配路径分隔符（`/`）。

如果要匹配子目录里面的文件，可以写成下面这样。

```
$ ls */*.txt
```

Bash 4.0 新增了一个 `globstar` 参数，允许 `**` 匹配零个或多个子目录，详见后面 `shopt` 命令的介绍。

(4) 文件名可以使用通配符。

Bash 允许文件名使用通配符，即文件名包括特殊字符。这时引用文件名，需要把文件名放在单引号或双引号里面。

```
$ touch 'fo*'
$ ls
fo*
```

上面代码创建了一个 `fo*` 文件，这时 `*` 就是文件名的一部分。

14. 量词语法

量词语法用来控制模式匹配的次数。它只有在 Bash 的 `extglob` 参数打开的情况下才能使用，不过一般是默认打开的。下面的命令可以查询。

```
$ shopt extglob
extglob          on
```

如果 `extglob` 参数是关闭的，可以用下面的命令打开。


```
$ shopt -s extglob
```

量词语法有下面几个。

- `?(pattern-list)` : 模式匹配零次或一次。
- `*(pattern-list)` : 模式匹配零次或多次。
- `+(pattern-list)` : 模式匹配一次或多次。
- `@(pattern-list)` : 只匹配一次模式。
- `!(pattern-list)` : 匹配给定模式以外的任何内容。

```
$ ls abc?(.)txt
abctxt abc.txt
```

上面例子中, `?(.)` 匹配零个或一个点。

```
$ ls abc?(def)
abc abcdef
```

上面例子中, `?(def)` 匹配零个或一个 `def` 。

```
$ ls abc@(.txt|.php)
abc.php abc.txt
```

上面例子中, `@(.txt|.php)` 匹配文件有且只有一个 `.txt` 或 `.php` 后缀名。

```
$ ls abc+ (.txt)
abc.txt abc.txt.txt
```

上面例子中, `+ (.txt)` 匹配文件有一个或多个 `.txt` 后缀名。

```
$ ls a!(b).txt
a.txt abb.txt ac.txt
```

上面例子中, `!(b)` 表示匹配单个字母 `b` 以外的任意内容, 所以除了 `ab.txt` 以外, 其他文件名都能匹配。

量词语法也属于文件名扩展，如果不存在可匹配的文件，就会原样输出。

```
# 没有 abc 开头的文件名
$ ls abc?(def)
ls: 无法访问 'abc?(def)': 没有那个文件或目录
```

上面例子中，由于没有可匹配的文件，`abc?(def)` 就原样输出，导致 `ls` 命令报错。

15. shopt 命令

`shopt` 命令可以调整 Bash 的行为。它有好几个参数跟通配符扩展有关。

`shopt` 命令的使用方法如下。

```
# 打开某个参数
$ shopt -s [optionname]

# 关闭某个参数
$ shopt -u [optionname]

# 查询某个参数关闭还是打开
$ shopt [optionname]
```

(1) dotglob 参数

`dotglob` 参数可以让扩展结果包括隐藏文件（即点开头的文件）。

正常情况下，扩展结果不包括隐藏文件。

```
$ ls *
abc.txt
```

打开 `dotglob`，就会包括隐藏文件。

```
$ shopt -s dotglob
$ ls *
abc.txt .config
```

(2) nullglob 参数

`nullglob` 参数可以让通配符不匹配任何文件名时，返回空字符。

默认情况下，通配符不匹配任何文件名时，会保持不变。

```
$ rm b*
rm: 无法删除 'b*': 没有那个文件或目录
```

上面例子中，由于当前目录不包括 `b` 开头的文件名，导致 `b*` 不会发生文件名扩展，保持原样不变，所以 `rm` 命令报错没有 `b*` 这个文件。

打开 `nullglob` 参数，就可以让不匹配的通配符返回空字符串。

```
$ shopt -s nullglob
$ rm b*
rm: 缺少操作数
```

上面例子中，由于没有 `b*` 匹配的文件名，所以 `rm b*` 扩展成了 `rm`，导致报错变成了“缺少操作数”。

(3) failglob 参数

`failglob` 参数使得通配符不匹配任何文件名时，Bash 会直接报错，而不是让各个命令去处理。

```
$ shopt -s failglob
$ rm b*
bash: 无匹配: b*
```

上面例子中，打开 `failglob` 以后，由于 `b*` 不匹配任何文件名，Bash 直接报错了，不再让 `rm` 命令去处理。

(4) extglob 参数

`extglob` 参数使得 Bash 支持 ksh 的一些扩展语法。它默认应该是打开的。

```
$ shopt extglob
extglob          on
```

它的主要应用是支持量词语法。如果不希望支持量词语法，可以用下面的命令关闭。

```
$ shopt -u extglob
```

(5) `nocaseglob` 参数

`nocaseglob` 参数可以让通配符扩展不区分大小写。

```
$ shopt -s nocaseglob
$ ls /windows/program*
/windows/ProgramData
/windows/Program Files
/windows/Program Files (x86)
```

上面例子中，打开 `nocaseglob` 以后，`program*` 就不区分大小写了，可以匹配 `ProgramData` 等。

(6) `globstar` 参数

`globstar` 参数可以使得 `**` 匹配零个或多个子目录。该参数默认是关闭的。

假设有下面的文件结构。

```
a.txt
sub1/b.txt
sub1/sub2/c.txt
```

上面的文件结构中，顶层目录、第一级子目录 `sub1`、第二级子目录 `sub1\sub2` 里面各有一个文本文件。请问怎样才能使用通配符，将它们显示出来？

默认情况下，只能写成下面这样。

```
$ ls *.txt */*.txt */*/*.txt  
a.txt sub1/b.txt sub1/sub2/c.txt
```

这是因为 `*` 只匹配当前目录，如果要匹配子目录，只能一层一层写出来。

打开 `globstar` 参数以后，`**` 匹配零个或多个子目录。因此，`*/*.txt` 就可以得到想要的结果。

```
$ shopt -s globstar  
$ ls **/*.txt  
a.txt sub1/b.txt sub1/sub2/c.txt
```

16. 参考链接

- [Think You Understand Wildcards? Think Again](#)
- [Advanced Wildcard Patterns Most People Don't Know](#)

▢ 基本语法

引号和转义 ▢

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