

linux shell 字符串操作详解（长度，读取，替换，截取，连接，对比，删除，位置）

原创 董武明

最新推荐文章于 2025-01-26 08:44:12 发布


阅读量5.7w

收藏 26

点赞数 9

分类专栏：

shell

 shell 专栏收录该内容

22 篇文章

在做shell批处理程序时候，经常会涉及到字符串相关操作。有很多命令语句，如：awk,sed都可以做字符串各种操作。其实shell内置一系列操作符号类似效果，大家知道，使用内部操作符合会省略启动外部程序等时间，因此速度会非常的快。

一、判断读取字符串值

表达式 含义

<code>\${var}</code>	变量var的值, 与\$var相同
<code>\${var-DEFAULT}</code>	如果var没有被声明, 那么就以\$DEFAULT作为其值 *
<code>\${var:-DEFAULT}</code>	如果var没有被声明, 或者其值为空, 那么就以\$DEFAULT作为其值 *
<code>\${var=DEFAULT}</code>	如果var没有被声明, 那么就以\$DEFAULT作为其值 *
<code>\${var:=DEFAULT}</code>	如果var没有被声明, 或者其值为空, 那么就以\$DEFAULT作为其值 *
<code>\${var+OTHER}</code>	如果var声明了, 那么其值就是\$OTHER, 否则就为null字符串
<code>\${var:+OTHER}</code>	如果var被设置了, 那么其值就是\$OTHER, 否则就为null字符串
<code>\${var?ERR_MSG}</code>	如果var没被声明, 那么就打印\$ERR_MSG *
<code>\${var:?ERR_MSG}</code>	如果var没被设置, 那么就打印\$ERR_MSG *
<code>\${!varprefix*}</code>	匹配之前所有以varprefix开头进行声明的变量
<code>\${!varprefix@}</code>	匹配之前所有以varprefix开头进行声明的变量

加入了“*” 不是意思是：当然, 如果变量var已经被设置的话, 那么其值就是\$var.

二、字符串操作（长度，读取，替换）

表达式 含义

<code>\${#string}</code>	\$string的长度
<code>\${string:position}</code>	在\$string中, 从位置\$position开始提取子串

<code>\${string:position:length}</code>	在\$string中, 从位置\$position开始提取长度为\$length的子串
<code>\${string#substring}</code>	从变量\$string的开头, 删除最短匹配\$substring的子串
<code>\${string##substring}</code>	从变量\$string的开头, 删除最长匹配\$substring的子串
<code>\${string%substring}</code>	从变量\$string的结尾, 删除最短匹配\$substring的子串
<code>\${string%%substring}</code>	从变量\$string的结尾, 删除最长匹配\$substring的子串
<code>\${string/substring/replacement}</code>	使用\$replacement, 来代替第一个匹配的\$substring
<code>\${string//substring/replacement}</code>	使用\$replacement, 代替所有匹配的\$substring
<code>\${string/#substring/replacement}</code>	如果\$string的前缀匹配\$substring, 那么就用\$replacement来代替匹配到的\$substring
<code>\${string/%substring/replacement}</code>	如果\$string的后缀匹配\$substring, 那么就用\$replacement来代替匹配到的\$substring

说明: `"* $substring"`可以是一个正则表达式.

实例:

读取:

Java代码 ☆

1. `$ echo ${abc:'ok'}`
2. ok
3. `$ echo $abc`
4. `$ echo ${abc='ok'}`
5. ok
6. `$ echo $abc`
7. ok
- 8.
9. #如果abc 没有声明"=" 还会给abc赋值。
10. `$ var1=11;var2=12;var3=`
11. `$ echo ${!v@}`
12. `var1 var2 var3`
13. `$ echo ${!v*}`
14. `var1 var2 var3`
- 15.
16. `#${!varprefix*}`与`${!varprefix@}`相似, 可以通过变量名前缀字符, 搜索已经定义的变量,无论是否为空值。

1, 取得字符串长度

C代码 ☆

1. string=abc12342341 //等号二边不要有空格
2. echo \${#string} //结果11
3. expr length \$string //结果11
4. expr "\$string" : "." * //结果11 分号二边要有空格,这里的:跟match的用法差不多

2, 字符串所在位置

C代码 ☆

1. expr index \$string '123' //结果4 字符串对应的下标是从1开始的

C代码 ☆

1. str="abc"
2. expr index \$str "a" # 1
3. expr index \$str "b" # 2
4. expr index \$str "x" # 0
5. expr index \$str "" # 0

这个方法让我想起了js的indexOf，各种语言对字符串的操作方法大方向都差不多，如果有语言基础的话，学习shell会很快的。

3, 从字符串开头到子串的最大长度

C代码 ☆

1. expr match \$string 'abc.*3' //结果9

个人觉得这个函数的用处不大，为什么要从开头开始呢。

4, 字符串截取

C代码 ☆

1. echo \${string:4} //2342341 从第4位开始截取后面所有字符串
2. echo \${string:3:3} //123 从第3位开始截取后面3位
3. echo \${string:3:6} //123423 从第3位开始截取后面6位
4. echo \${string:-4} //2341 : 右边有空格 截取后4位
5. echo \${string:~-4} //2341 同上
6. expr substr \$string 3 3 //123 从第3位开始截取后面3位

C代码 ☆

1. str="abcdef"
2. expr substr "\$str" 1 3 # 从第一个位置开始取3个字符, abc
3. expr substr "\$str" 2 5 # 从第二个位置开始取5个字符, bcdef
4. expr substr "\$str" 4 5 # 从第四个位置开始取5个字符, def

- 5.
6. echo \${str:2} # 从第二个位置开始提取字符串, bcdef
7. echo \${str:2:3} # 从第二个位置开始提取3个字符, bcd
8. echo \${str:(-6):5} # 从倒数第二个位置向左提取字符串, abcde
9. echo \${str:(-4):3} # 从倒数第二个位置向左提取6个字符, cde

上面的方法让我想起了, php的substr函数, 后面截取的规则是一样的。

5, 匹配显示内容

C代码 ☆

1. //例3中也有match和这里的match不同, 上面显示的是匹配字符的长度, 而下面的是匹配的内容
2. expr match \$string "[a-c]*[0-9]*" //abc12342341
3. expr \$string : "[a-c]*[0-9]" //abc1
4. expr \$string : ".*[0-9][0-9][0-9]" //341 显示括号中匹配的内容

这里括号的用法, 是不是跟其他的括号用法有相似之处呢,

6, 截取不匹配的内容

C代码 ☆

1. echo \${string#a*3} //42341 从\$string左边开始, 去掉最短匹配子串
2. echo \${string#c*3} //abc12342341 这样什么也没有匹配到
3. echo \${string#*c1*3} //42341 从\$string左边开始, 去掉最短匹配子串
4. echo \${string##a*3} //41 从\$string左边开始, 去掉最长匹配子串
5. echo \${string%3*1} //abc12342 从\$string右边开始, 去掉最短匹配子串
6. echo \${string%%3*1} //abc12 从\$string右边开始, 去掉最长匹配子串

C代码 ☆

1. str="abbc,def,ghi,abcjkl"
2. echo \${str#a*c} # 输出,def,ghi,abcjkl 一个井号(#) 表示从左边截取掉最短的匹配 (这里把abbc字符串去掉)
3. echo \${str##a*c} # 输出jkl, 两个井号(##) 表示从左边截取掉最长的匹配 (这里把abbc,def,ghi,abc字符串去掉)
4. echo \${str#"a*c"} # 输出abbc,def,ghi,abcjkl 因为str中没有"a*c"子串
5. echo \${str##"a*c"} # 输出abbc,def,ghi,abcjkl 同理
6. echo \${str#a*c*} # 空
7. echo \${str##a*c*} # 空
8. echo \${str#d*f} # 输出abbc,def,ghi,abcjkl,
9. echo \${str#*d*f} # 输出,ghi,abcjkl
- 10.
11. echo \${str%a*I} # abbc,def,ghi 一个百分号(%)表示从右边截取最短的匹配
12. echo \${str%%b*I} # a 两个百分号表示(%%)表示从右边截取最长的匹配
13. echo \${str%a*c} # abbc,def,ghi,abcjkl

这里要注意, 必须从字符串的第一个字符开始, 或者从最后一个开始, 可以这样记忆, 井号(#) 通常用于表示一个数字, 它是放在前面的; 百分号(%) 字的后面; 或者这样记忆, 在键盘布局中, 井号(#)总是位于百分号(%) 的左边(即前面)。

7. 匹配并且替换

C代码 ☆

1. `echo ${string/23/bb} //abc1bb42341` 替换一次
2. `echo ${string//23/bb} //abc1bb4bb41` 双斜杠替换所有匹配
3. `echo ${string/#abc/bb} //bb12342341` #以什么开头来匹配，根php中的^有点像
4. `echo ${string/%41/bb} //abc123423bb` %以什么结尾来匹配，根php中的\$有点像

C代码 ☆

1. `str="apple, tree, apple tree"`
2. `echo ${str/apple/APPLE}` # 替换第一次出现的apple
3. `echo ${str//apple/APPLE}` # 替换所有apple
- 4.
5. `echo ${str/#apple/APPLE}` # 如果字符串str以apple开头，则用APPLE替换它
6. `echo ${str/%apple/APPLE}` # 如果字符串str以apple结尾，则用APPLE替换它

C代码 ☆

1. `$ test='c:/windows/boot.ini'`
2. `$ echo ${test//\\}`
3. `c:\windows\boot.ini`
4. `$ echo ${test//\\/}`
5. `c:\windows\boot.ini`
- 6.
7. `#{变量/查找/替换值}` 一个"/"表示替换第一个，"/"表示替换所有,当查找中出现了: "/"请加转义符"\"表示。

8. 比较

C代码 ☆

1. `[["a.txt" == a*]]` # 逻辑真 (pattern matching)
2. `[["a.txt" =~ .*\.txt]]` # 逻辑真 (regex matching)
3. `[["abc" == "abc"]]` # 逻辑真 (string comparision)
4. `[["11" < "2"]]` # 逻辑真 (string comparision), 按ascii值比较

9. 连接

C代码 ☆

1. `s1="hello"`
2. `s2="world"`
3. `echo ${s1}${s2}` # 当然这样写 `s1$s2` 也行，但最好加上大括号

10. 字符串删除

Java代码 ☆

1. `$ test='c:/windows/boot.ini'`

2. \$ echo \${test#}/
3. c:/windows/boot.ini
4. \$ echo \${test#*/}
5. windows/boot.ini
6. \$ echo \${test###}/
7. boot.ini
8. \$ echo \${test%/*}
9. c:/windows
10. \$ echo \${test%%/*}
11. c:
12. #\${变量名#substring正则表达式}从字符串开头开始配备substring,删除匹配上的表达式。
13. #\${变量名%substring正则表达式}从字符串结尾开始配备substring,删除匹配上的表达式。
14. #注意: \${test##*/},\${test%/*} 分别是得到文件名和目录地址最简单方法。

 **董武明**

关注

👍 9

👎

🌟 26

💬 6

🔗 分享

...

【转】linux shell 字符串操作详解 （长度，读取，替换，截取，连接，对比，删除，位置）阿星的笔记

linux shell 字符串操作详解 （长度，读取，替换，截取，连接，对比，删除，位置） 在做shell批处理程序时候，经常会涉及到字符串相关操作。有很多命令语句，如：cat

shell 字符串操作(长度，查找，替换)详解

代码如下:工作中字符串操作举例 filename='/home/admin/jobs/CnClickstat/DFSLoader/loader.cfg' #下面是使用shell字符串操作 buName1=\${filename#*/jobs/} #去除'/home

linux shell 字符串操作详解 （长度，读取，替换，截取，连接，对比，删除，位置） ... Just C

在做shell批处理程序时候，经常会涉及到字符串相关操作。有很多命令语句，如：awk,sed都可以做字符串各种操作。其实shell内置一系列操作符号，可以达到类似效果，

Bash语言的字符串处理 最新发布 2501_90416967的笔记

Bash的字符串处理虽然简单，却能帮助我们完成许多复杂的任务。从基本的字符串拼接到高级的模式匹配，Bash提供了丰富的功能来处理文本和数据。掌握这些字符串处

shell字符串操作 盛源的笔记

1 字符串截取操作 假设有变量 var=http://www.ssjt.com/shy.htm 1.2删除某个子串及其左边的内容 1) #号截取，从左边开始查找要删除的内容 echo \${var#*/} // 表示从左边

shell中的字符串操作 珂

SHELL字符串操作bash Shell提供了多种字符串处理的命令： awk命令 expr命令

【shell】shell字符串操作（声明、长度、拼接、切片、转换、替换等操作） 测试开发自

👉博__主👉：米码收割机👉技__能👉：C++/Python语言👉公众号👉：测试开发自动化👉专__注👉：专注主流机器人、人工智能等相关领域的开发、测试技术。

Linux shell 字符串操作详解 （长度，读取，替换，截取，连接，对比，删除，位置） 热门推荐 zif1665119803的笔记

Linux shell 字符串操作详解 （长度，读取，替换，截取，连接，对比，删除，位置） 1. Linux shell 截取字符变量的前8位2. 按指定的字符串截取3. 按照指定要求分割：4.

Shell字符串操作大全 GatsbyNev

1.字符串声明 一般字符串声明时，都会赋值 str='test' 但是在脚本运行中，为了避免出现引用为声明的字符串变量时，可以在如下处理： #如果str没声明，则输出DEFAULT

SHELL字符串处理技巧（\${}、##、%） winshining的笔记

在SHELL编程中，经常要处理一些字符串变量。比如，计算长度啊、截取子串啊、字符替换啊等等，常常要用到awk、expr、sed、tr等命令。下面给大家介绍个简单的字

Shell脚本之字符串操作 xinfesmile123的笔记

文章目录前言一、字符串获取二、字符串截取1. #号截取（自左向右）2. ##号截取（自左向右）3. %号截取（自右向左）4. %%号截取（自右向左）三、字符串终端打印1

Shell字符串常见操作 Zephyrzh的博客，关注【凡登】成为大厂技

shell属于弱类型语言，它不会强制你声明变量的数据类型。当给一个变量赋值时，根据赋值的内容自动确定数据类型。本文主要介绍字符串基本操作获取长度、截取、匹

Shell编程——字符串的相关操作（定义、拼接、截取、求长度） 主要记录嵌入式学习与开发过

以下内容源于C语言中文网的学习与整理，非原创，如有侵权请告知删除。

shell 字符串操作

Shell 字符串操作详解 #### 一、引言 在Shell编程中，字符串操作是一项基本而重要的技能。无论是简单的脚本还是复杂的自动化任务，掌握如何有效地操作字符串都

shell字符串处理 皓月如我的笔记

转自：http://mcuos.com/thread-2357-1-1.html 一、构造字符串 直接构造 STR_ZERO=hello STR_FIRST="i am a string" STR_SECOND='success' 重复多次 #repeat the

Shell: 字符串操作 加油，码

转载：http://www.cnblogs.com/dwdxdy/archive/2012/07/25/2608257.html，谢谢！ Shell内置的一系列字符串操作符号。具体的相关操作符号介绍如下： 表达式 含义 \${#s



董武明

码龄13年

 暂无认证

126

5万+

12万+

86万+



原创

周排名

总排名

访问

等级

7332

87

144

82

396

积分

粉丝

获赞

评论

收藏









私信

关注

AI 搜索

Deepseek

R1 满血版



搜博主文章




热门文章


- ssh-keygen的使用方法&配置

authorized_keys两台linux机器相互认证


64930


- linux shell 字符串操作详解（长度，读取，替换，截取，连接，对比，删除，位置）


57091


- 75道经典逻辑思维题及答案


43823


- dbeaver Can't create driver instance

37412


- 静态代码分析工具汇总

29618



分类专栏

- 

nexus

1篇
- 

sonarqube

1篇
- 

逻辑思维题

1篇
- 

shell

22篇
- 

linux

46篇
- 

android

32篇
- 

最新评论

Outlook启动时提示“找不到文件Outlook.p...

m0_49065527: 补充说明：删除账号，删除数据后，使用这个方法，还是没有回到 ...

Outlook启动时提示“找不到文件Outlook.p...

m0_49065527: CD 就是打上CD 空格 后接文件夹路径。

Foxmail 邮件太多导致邮箱卡顿、卡死且...

m0_51256539: 实际上使用一套土办法就行: foxmail卡顿-邮件过多/另类的备份的 ...

RVCT的Linux环境变量配置 ARM® RVD...

weixin_42619190: 楼主你怎么解决license问题?

Vue 接入 CAS统一认证登录

Benson_7: 你好，为什么我配置了但是还是报错CAS Authentication requires a cas_ ...

最新文章

夫妻已购共有产权房，未成年子女是否可以买商品房？

增加弹跳训练计划

Gerrit push代码提示! [remote rejected]
HEAD -> dev (more than 10000 commits, and skip-validation not

2024年 3篇	2023年 5篇
2022年 9篇	2021年 5篇
2020年 6篇	2019年 4篇
2018年 7篇	2017年 19篇
2016年 25篇	2015年 16篇
2014年 10篇	2013年 70篇

目录