

# Linux 基础

## 第 12 讲 shell 脚本：变量和文本的处理



# 变量的解析处理

- shell 中的变量保存的是文本，并且 shell（bash）提供了非常多的对文本处理的方法。
- 对于变量 a 来说，用于处理的格式基本遵循以下形式：  
`${a...}`
- 对于这样的处理结果，会生成新的值，可以赋值，或者是输出，但是不会改变 a 的值，除非重新赋值：

`a=${a:2}` # 截取 a 从第 2 个字符开始到末尾并重新赋值

# 变量的长度

- `${#a}` 可以获取变量 `a` 的长度。对于中文也是支持的，会输出中文字符个数，而不是对应 UTF-8 编码的字符长度。
- 变量的长度可以保存在新的变量中。

```
1 #!/bin/bash
2
3 a='我是中国人'
4 echo ${#a}
5
```

# 变量截取

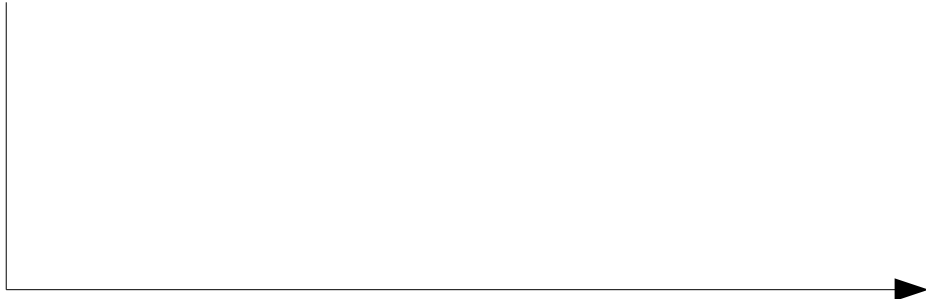
- 对文本的截取是经常要用到的，在编程语言中，都是通过调用函数来实现的。
- 在 shell 中，使用以下语法来截取文本：

`${a:start}`

`${a:start:length}`

- `start` 是字符串开始的位置，从 0 开始计数，`length` 截取的长度，如果不指定 `length` 或 `length` 超过字符串长度则截取到末尾。

```
1 #!/bin/bash
2
3 a='雨纷纷，欲断魂。何处有，杏花村。'
4
5 echo ${#a}
6
7 echo ${a:8}
8
```



```
[ ~/mysh ]
io@daojian:$ ./vars.sh
16
何处有，杏花村。
[ ~/mysh ]
io@daojian:$
```

# 移除匹配的前缀和后缀

- 移除匹配的前缀：

`${a#word}`

`${a##word}`


- 移除匹配的后缀：

`${a%word}`

`${a%%word}`

- word 是要匹配的字符串，这不会改变 a 的值，而是返回新的字符串。

```
1 #!/bin/bash
2
3 a='abcdef'
4
5 echo ${a#ab}
6
7 echo ${a%%ef}
8
9 echo $a
10
```



```
io@daojian:$ ./vars2.sh
cxyzdef
abcxyzd
abcxyzdef
```

# 替换匹配的字符串

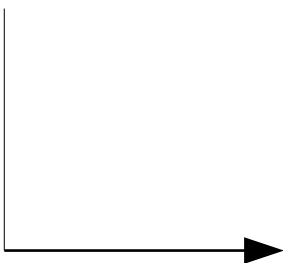
- 对文本的子串替换也是常见的操作， shell 使用以下语法来进行替换操作：

```
${a/pattern/string}
```

- pattern 是要匹配的字符串， string 是要替换的字符串。



```
1 #!/bin/bash
2
3 a='ubuntu debian centos arch mint freebsd openbsd deepin'
4
5 #替换的字符串为空，表示去掉centos
6 b=${a/centos/}
7 echo $b
8
9 echo ${a/mint/netbsd}
10
```



```
io@daojian:$ ./varsreplace.sh
ubuntu debian arch mint freebsd openbsd deepin
ubuntu debian centos arch netbsd freebsd openbsd deepin
[ ~/mysh ]
io@daojian:$
```


# 其他操作

- `${a:-word}` 如果 `a` 的值没有设置或为空，则使用 `word` 表示的值。
- `${a:=word}` 如果 `a` 的值没有设置或为空，则会把 `word` 表示的值赋值给 `a`，并使用 `a` 的值。
- `${a:?word}` 如果 `a` 的值没有设置或为空，则会显示错误信息，信息提示为 `word` 表示的值。

# 获取用户输入并保存到变量

- `read` 是 `shell` 内建命令，用来把输入保存到 `shell` 的一个或多个变量。
- `read` 默认使用 `IFS` 环境变量的设定值作为分隔符：空格、Tab、换行。
- 设置 `IFS` 变量可以让 `read` 使用其他值作为分隔符。

```
1 #!/bin/bash
2
3 printf "输入用户名: "
4 read username
5
6 printf "输入密码: "
7 read passwd
8
9 echo "Username: $username ; Password: $passwd"
10
```



```
io@daojian:$ ./read.sh
输入用户名: wang
输入密码: x1234
Username: wang ; Password: x1234
[ ~/mysh ]
```

# 从结构化文本提取数据

- Linux/Unix 上文本类型的配置文件都是结构化的数据。当然，实际的开发工作中，结构化文本使用也是非常多的。
- 在处理结构化文本时，经常要提取有用的信息字段，在 shell 中，可以提取的方式有很多种，常用的工具有两个：awk、cut。

# awk：模式匹配的程序设计语言

- 把 awk 说成是一种语言肯定会让人望而生畏，当初 awk 的设计就是为了简化通常的文本处理工作。
- 使用 awk 的好处是，不必担心可移植问题，几乎所有的 Linux/Unix 都会有一个 awk 的实现。
- 但是 awk 确实很复杂，以至于完全可以用一本书来讲述。不过你不必担心，我们只会用它来提取文本。

# 使用 awk 提取进程的 PID

- 使用 `awk -F ' ' '{printf $2}'` 这样的模式可以分割并提取指定字段，`-F` 表示要分割的字符，这里指定为空格，`$2` 表示提取第二个字段。
- 使用以下方式可以提取进程的 PID：  

```
ps -e -o pid,user,comm,args | awk -F ' ' '{printf $1 "\n"}'
```
- 其中 `'{printf $1}'` 是要执行的命令，输出第一个字段，配合 `grep` 可以搜索并提取指定进程的 PID。

# cut 提取文本

- 使用 cut 提取文本相对来说很容易， cut 是一个专门用于提取文本片段的工具。
- 使用 cut 提取 /etc/passwd 的用户名和默认登录 shell  

```
cat /etc/passwd | cut -d : -f 1,7
```
- cut 参数 -d 指定分割字符， -f 指定分割后第几个字段。



# 练习

- 1. 假设有进程 `axyz`，不止一个，但是有一个进程其父进程 PID 是 1，现在要筛此进程并提取它的 PID，然后终止此进程。
- 2. 提取 `/etc/passwd` 中的用户字段数据，并排序后保存到一个文件中，但是排序命令你不是很清楚，所以你可能需要通过 `man -k sort` 来获取一些信息。