shell编程范例之字符串操作

忙活了一个礼拜，终于等到周末，可以空下来写点东西。
这次介绍 _字符串操作_ 了，这里先得明白两个东西，什么是字符串，对字符串有哪些操作？

下面是"在线新华字典"的解释：
字符串:
简称“串”。有限字符的序列。数据元素为字符的线性表，是一种数据的逻辑结构。在计算机中可有不同的存储结构。在串上可进行求子串、插入字符、删除字符、置换字符等运算。

而字符呢？
字符:
计算机程序设计及操作时使用的符号。包括字母、数字、空格符、提示符及各种专用字符等。

照这样说，之前介绍的数值操作中的数字，逻辑运算中的真假值，都是以字符的形式呈现出来的，是一种特别的字符，对它们的运算只不过是字符操作的特例罢了。而这里将研究一般字符的运算，它具有非常重要的意义，因为对我们来说，一般的工作都是处理字符而已。这些运算实际上将围绕上述两个定义来做。

第一、找出字符或者字符串的类型，是数字、字母还是其他特定字符，是可打印字符，还是不可打印字符（一些控制字符）。
第二、找出组成字符串的字符个数和字符串的存储结构（比如数组）。
第三、对串的常规操作：求子串、插入字符、删除字符、置换字符、字符串的比较等。
第四、对串的一些比较复杂而有趣的操作，这里将在最后介绍一些有趣的范例。

1. 字符串的属性

1.1 字符串的类型

字符有可能是数字、字母、空格、其他特殊字符，而字符串有可能是它们任何一种或者多种的组合，在组合之后还可能形成一个具有特定意义的字符串，诸如邮件地址，URL地址等。

概要示例: 下面我们来看看如何判断字符的类型。

```
// 数字或者数字组合（能够返回结果，即程序退出状态是0，说明属于这种类型，反之不然）
$ i=5;j=9423483247234;
$ echo $i | grep [0-9]*
5
$ echo $j | grep [0-9]*
9423483247234
$ echo $j | grep [0-9]* >/dev/null
$ echo $?
0
// 字符组合（小写字母、大写字母、两者的组合）
$ c="A"; d="fwefewjuew"; e="fewfEFWefwefe"
$ echo $c | grep [A-Z]
A
$ echo $d | grep "[a-z]*"
fwefewjuew
$ echo $e | grep "[a-zA-Z]*"
fewfEFWefwefe
// 字母和数字的组合
$ ic="432fwfwefeFWEwefwef"
$ echo $ic | grep "[0-9a-zA-Z]*"
432fwfwefeFWEwefwef
// 空格或者Tab键等
```

```
$ echo " " | grep " "

$ echo -e "\t" | grep "[[:space:]]" #[[:space:]]会同时匹配空格和TAB键

$ echo -e " \t" | grep "[[:space:]]"

$ echo -e "\t" | grep "<tab>" #<tab>为在键盘上按下TAB键，而不是字符<tab>
// 匹配邮件地址
$ echo "test2007@lzu.cn" | grep "[0-9a-zA-Z\.]*@[0-9a-zA-Z\.]"
test2007@lzu.cn
// 匹配URL地址(以http链接为例)
$ echo "http://news.lzu.edu.cn/article.jsp?newsid=10135" | grep "http://[0-9a-zA-Z\./=?]*"
http://news.lzu.edu.cn/article.jsp?newsid=10135
```

说明：
[1] /dev/null和/dev/zero是非常有趣的两个设备，它们都犹如一个黑洞，什么东西掉进去都会消失殆尽；后者则是一个能源箱，你总能从那里取到0，直到你退出。两者的部分用法见：关于zero及NULL设备的一些问题
[2] [[:space:]]是grep用于匹配空格或者TAB键类型字符串的一种标记，其他类似的标记请查看grep的帮助,man grep。
[3] 上面都是用grep来进行模式匹配，实际上sed, awk都可以用来做模式匹配，关于匹配中用到的正则匹配模式知识，大家可以参考正则匹配模式，更多相关资料请看参考资料。
[4] 如果仅仅想判断字符串是否为空，即判断字符串的长度是否为零，那么可以简单的通过test命令的-z选项来判断，具体用法见test命令，man test.

概要示例: 判断字符是否可打印？如何控制字符在终端的显示。

```
// 用grep判断某个字符是否为可打印字符
$ echo "\t\n" | grep "[[:print:]]"
\t\n
$ echo $?
0
$ echo -e "\t\n" | grep "[[:print:]]"
$ echo $?
1
// 用echo的-e选项在屏幕控制字符显示位置、颜色、背景等
$ echo -e "\33[31;40m" #设置前景色为黑色，背景色为红色
$ echo -e "\33[11;29H Hello, World\!" #在屏幕的第11行，29列开始打印字符串Hello,World!
// 在屏幕的某个位置动态显示当前系统时间
$ while :; do echo -e "\33[11;29H "$(date "+%Y-%m-%d %H:%M:%S"); done
// 用col命令过滤掉某些控制字符，在处理诸如script,screen等截屏命令的输出结果时，很有用
$ screen -L
$ cat /bin/cat
$ exit
$ cat screenlog.0 | col -b   # 把一些控制字符过滤后，就可以保留可读的操作日志
```

更多关于字符在终端的显示控制方法，请参考资料[20]和字符显示实例[21]：用shell实现的一个动态时钟。

1.2 字符串的长度

概要示例: 除了组成字符串的字符类型外，字符串还有哪些属性呢？组成字符串的字符个数。下面我们来计算字符串的长度，即所有字符的个数，并简单介绍几种求字符串中指定字符个数的方法。

```
// 计算某个字符串的长度，即所有字符的个数[这计算方法是五花八门，择其优着而用之]
$ var="get the length of me"
$ echo ${var}     # 这里等同于$var
```

```
get the length of me
$ echo ${#var}
20
$ expr length "$var"
20
$ echo $var | awk '{printf("%d\n", length($0));}'
20
$ echo -n $var |  wc -c
20
// 计算某些指定一个字符或者多个字符的个数
$ echo $var | tr -cd g | wc -c
2
$ echo -n $var | sed -e 's/[^g]//g' | wc -c
2
$ echo -n $var | sed -e 's/[^gt]//g' | wc -c
5
// 如果要统计单词个数，更多相关信息见《shell编程之数值计算》之 _单词统计_ 实例。
$ echo $var | wc -w
5
$ echo "$var" | tr " " "\n" | grep get | uniq -c
1
$ echo "$var" | tr " " "\n" | grep get | wc -l
1
```

说明：
${}操作符在Bash里头一个“大牛”，能胜任相当多的工作，具体就看看网中人的《shell十三问》之《Shell十三问》之"$(( )) 與 $( ) 還有${ } 差在哪？" 吧。

1.3 字符串的存储

在我们看来，字符串是一连串的字符而已，但是为了操作方便，我们往往可以让字符串呈现出一定的结构。在这里，我们不关心字符串在内存中的实际存储结构，仅仅关系它呈现出来的逻辑结构。比如，这样一个字符串："get the length of me"，我们可以从不同的方面来呈现它。

1.3.1 通过字符在串中的位置来呈现它

这样我们就可以通过指定位置来找到某个子串。这在c语言里头通常可以利用指针来做。而在shell编程中，有很多可用的工具，诸如expr，awk都提供了类似的方法来实现子串的查询动作。两者都几乎支持模式匹配(match)和完全匹配(index)。这在后面的字符串操作中将详细介绍。

1.3.2 根据某个分割符来取得字符串的各个部分

这里最常见的就是行分割符、空格或者TAB分割符了，前者用来当行号，我们似乎已经司空见惯了，因为我们的编辑器就这样“莫名”地处理着行分割符（在 unix下为\n，在其他系统下有一些不同，比如windows下为\r\n)。而空格或者TAB键经常用来分割数据库的各个字段，这似乎也是司空见惯的事情。

正是因为这样，所以产生了大量优秀的行编辑工具，诸如grep,awk,sed等。在“行内”（姑且这么说吧，就是处理单行，即字符串里头不再包含行分割符）的字符串分割方面，cut和awk提供了非常优越的“行内”（处理单行）处理能力。

1.3.3 更方便地处理用分割符分割好的各个部分

同样是用到分割符，但为了更方便的操作分割以后的字符串的各个部分，我们抽象了“数组”这么一个数据结构，从而让我们更加方便地通过下标来获取某个指定的部分。bash提供了这么一种数据结构，而优秀的awk也同样提供了它，我们这里将简单介绍它们的用法。

概要示例：利用数组存放"get the length of me"的用空格分开的各个部分。

```
//1. bash提供的数组数据结构，它是以数字为下标的，和C语言从0开始的下标一样
$ var="get the length of me"
$ var_arr=($var)    #这里把字符串var存放到字符串数组var_arr中了，默认以空格作为分割符
$ echo ${var_arr[0]} ${var_arr[1]} ${var_arr[2]} ${var_arr[3]} ${var_arr[4]}
get the length of me
$ echo ${var_arr[@]}    #这个就是整个字符串所有部分啦，这里可以用*代替@，下同
get the length of me
$ echo ${#var_arr[@]}    #记得上面求某个字符串的长度么，#操作符，如果想求某个数组元素的字符串长度，那么就把@换成下标吧
5
// 你也可以直接给某个数组元素赋值
$ var_arr[5]="new_element"
$ echo ${var_arr[5]}
6
$ echo ${var_arr[5]}
new_element
// bash里头实际上还提供了一种类似于“数组”的功能，即"for i in 用指定分割符分开的字符串" 的用法
// 即，你可以很方便的获取某个字符串的某个部分
$  for i in $var; do echo -n $i" "; done;
get the length of me

//2. awk里头的数组，注意比较它和bash提供的数组的异同
// split把一行按照空格分割，存放到数组var_arr中，并返回数组的长度。注意：这里的第一个元素下标不是0，而是1
$ echo $var | awk '{printf("%d %s\n", split($0, var_arr, " "), var_arr[1]);}'
5 get
// 实际上，上面的操作很类似awk自身的行处理功能：awk默认把一行按照空格分割为多个域，并可以通过$1,$2,$3...来获取,$0表示整行
// 这里的NF是该行的域的总数，类似于上面数组的长度，它同样提供了一种通过“下标”访问某个字符串的功能
$ echo $var | awk '{printf("%d | %s %s %s %s %s | %s\n", NF, $1, $2, $3, $4, $5, $0);}'
5 | get the length of me | get the length of me
// awk的“数组”功能何止于此呢，看看它的for引用吧，注意，这个和bash里头的for不太一样，i不是元素本身，而是下标
$ echo $var | awk '{split($0, var_arr, " "); for(i in var_arr) printf("%s ",var_arr);}'
get the length of me
$ echo $var | awk '{split($0, var_arr, " "); for(i in var_arr) printf("%s ",i);}'
1 2 3 4 5
// awk还有更“厉害”的处理能力，它的下标可以不是数字，而可以是字符串，从而变成了“关联”数组，这种“关联”的作用在某些方便将让我们非常方便
// 比如，我们这里就实现一个非凡的应用，把某个文件中的某个系统调用名替换成地址，如果你真正用起它，你会感慨它的“鬼斧神工”的。
// 这就是我在一个场合最好才发现的随好的实现方案：有兴趣看看awk手册帖子中我在3楼回复的实例吧。
$ cat symbol
sys_exit
sys_read
sys_close
$ ls /boot/System.map*
$ awk '{if(FILENAME ~ "System.map") map[$3]=$1; else {printf("%s\n", map[$1])}}' /boot/System.map-2.6.20-16-generic symbol
c0129a80
c0177310
c0175d80
// 另外，awk还支持删除某个数组元素，如果你不用了就可以用delete函数给删除掉。如果某些场合有需要的话，别忘了awk还支持二维数组。
```

okay，就介绍到这里啦。为什么要介绍这些内容？再接着看下面的内容，你就会发现，那些有些的工具是怎么产生和发展起来的了，如果累了，看看最后一篇参考资料吧，它介绍了一些linux命令名字的由来，说不定可以帮助你理解本节下面的部分呢。

2. 字符串常规操作

字符串操作包括取子串、查询子串、插入子串、删除子串、子串替换、子串比较、子串排序、子串进制转换、子串编码转换等。

2.1 取子串

概要示例：取子串的方法主要有：直接到指定位置求子串，字符匹配求子串。

```
// 按照位置取子串，比如从什么位置开始，取多少个字符
$ var="get the length of me"
$ echo ${var:0:3}
get
$ echo ${var:(-2)}   # 方向相反呢
me
$ echo `expr substr "$var" 5 3` #记得把$var引起来，否则expr会因为空格而解析错误
the
$ echo $var | awk '{printf("%s\n", substr($0, 9, 6))}'
length

// 匹配字符求子串
$ echo ${var%% *} #从右边开始计算，删除最左边的空格右边的所有字符
get
$ echo ${var% *} #从右边开始计算，删除第一个空格右边的所有字符
get the length of
$ echo ${var##* }  #从左边开始计算，删除最右边的空格左边的所有字符
me
$ echo ${var#* }  #从左边开始计算，删除第一个空格左边的所有字符
the length of me

$ echo $var | awk '{printf("%s\n", $1);}' # awk把$var按照空格分开为多个变量，依次为$1,$2,$3,$4,$5
get
$ echo $var | awk '{printf("%s\n", $5);}'
me

$ echo $var | cut -d" " -f 5  #差点把cut这个小东西忘记啦，用起来和awk类似, -d指定分割符，如同awk用-F指定分割符一样，-f指定“域”，如同awk的$数字。

$ echo $var | sed 's/ [a-z]*//g'  #删除所有 空格＋字母串 的字符串，所以get后面的全部被删除了
get
$ echo $var | sed 's/[a-z]* //g'
me

$ echo $var | tr " " "\n" | sed -n 1p #sed有按地址（行）打印(p)的功能，记得先用tr把空格换成行号
get
$ echo $var | tr " " "\n" | sed -n 5p
me

// tr也可以用来取子串哦，它也可以类似#和%来“拿掉”一些字符串来实现取子串
$ echo $var | tr -d " "
getthelengthofme
$ echo $var | tr -cd "[a-z]" #把所有的空格都拿掉了，仅仅保留字母字符串，注意-c和-d的用法
getthelengthofme
```

说明：
[1] %和#的区别是，删除字符的方向不一样,前者在右，后者在左，%%和%,##和#的方向是前者是最大匹配，后者是最小匹配。(好的记忆方法见网中人的键盘记忆法:#$%是键盘依次从左到右的三个键)

[2] tr的-c选项是complement的缩写，即invert，而-d选项是删除的意思，tr -cd "[a-z]"这样一来就变成保留所有的字母啦。

对于字符串的截取，实际上还有一些命令，如果head,tail等可以实现有意思的功能，可以截取某个字符串的前面、后面指定的行数或者字节数。例如：

```
$ echo "abcdefghijk" | head -c 4
abcd
$ echo -n "abcdefghijk" | tail -c 4
hijk
```

2.2. 查询子串

概要示例：子串查询包括：返回符合某个模式的子串本身和返回子串在目标串中的位置。

准备：在进行下面的操作之前，请把http://oss.lzu.edu.cn/blog/blog.php?do_showone/tid_1385.html链接中的内容复制到一个文本text里头，用于下面的操作。

```
// 查询子串在目标串中的位置
$ var="get the length of me"
$ expr index "$var" t        #貌似仅仅可以返回某个字符或者多个字符中第一个字符出现的位置
3
$ echo $var | awk '{printf("%d\n", match($0,"the"));}'     #awk却能找出字串,match还可以匹配正则表达式
5

// 查询子串，返回包含子串的行(awk,sed都可以实现这些功能,但是grep最擅长)
$ grep "consists of" text   # 查询text文件包含consists of的行，并打印这些行
$ grep "consists[[:space:]]of" -n -H text # 打印文件名，子串所在行的行号和该行的内容
$ grep "consists[[:space:]]of" -n -o text # 仅仅打印行号和匹配到的子串本身的内容
$ awk '/consists of/{ printf("%s:%d:%s\n",FILENAME, FNR, $0)}' text  #看到没？和grep的结果一样
$ sed -n -e '/consists of/=;/consists of/p' text #同样可以打印行号
```

说明：
[1] awk,grep,sed都能通过模式匹配查找指定的字符串，但是它们各有擅长的领域，我们将在后续的章节中继续使用和比较它们，从而发现各自的优点。
[2] 在这里我们姑且把文件内容当成了一个大的字符串，在后面的章节中我们将专门介绍文件的操作，所以对文件内容中存放字符串的操作将会有更深入的分析和介绍。

2.3. 子串替换

子串替换就是把某个指定的子串替换成其他的字符串，实际上这里就蕴含了“插入子串”和“删除子串”的操作。例如，你想插入某个字符串到某个子串之前，就可以把原来的子串替换成”子串+新的字符串“，如果想删除某个子串，就把子串替换成空串。不过有些工具提供了一些专门的用法来做插入子串和删除子串的操作，所以呆伙还是会专门介绍的。另外，要想替换掉某个子串，一般都是先找到子串（查询子串），然后再把它替换掉的，实质上很多工具在使用和设计上都体现了这么一点。

概要示例：下面我们把变量var中的空格替换成下划线看看。

```
// 用{}运算符，还记得么？网中人的教程。
$ var="get the length of me"
$ echo ${var/ /_}         #把第一个空格替换成下划线
get_the length of me
$ echo ${var// /_}         #把所有空格都替换成了下划线了
get_the_length_of_me
```

```
// 用awk，awk提供了转换的最小替换函数sub和全局替换函数gsub，类似/和//
$ echo $var | awk '{sub(" ", "_", $0); printf("%s\n", $0);}'
get_the length of me
$ echo $var | awk '{gsub(" ", "_", $0); printf("%s\n", $0);}'
get_the_length_of_me

// 用sed了，子串替换可是sed的特长
$ echo $var | sed -e 's/ /_'    #s <= substitude
get_the length of me
$ echo $var | sed -e 's/ /_/g'    #看到没有，简短两个命令就实现了最小匹配和最大匹配g <= global
get_the_length_of_me

// 有忘记tr命令么？可以用替换单个字符的
$ echo $var | tr " " "_"
get_the_length_of_me
$ echo $var | tr '[a-z]' '[A-Z]'   #这个可有意思了，把所有小写字母都替换为大写字母
GET THE LENGTH OF ME
```

说明：sed还有很有趣的标签用法呢，下面再介绍吧。

有一种比较有意思的字符串替换是，整个文件行的倒置，这个可以通过tac命令实现，它会把文件中所有的行全部倒转过来。在一定意义上来说，排序实际上也是一个字符串替换。

2.4. 插入子串

插入子串：就是在指定的位置插入子串，这个位置可能是某个子串的位置，也可能是从某个文件开头算起的某个长度。通过上面的练习，我们发现这两者之间实际上是类似的。

公式：插入子串=把"old子串"替换成"old子串+new子串"或者"new子串+old子串"

概要示例：：下面在var字符串的空格之前或之后插入一个下划线

```
// 用{}
$ var="get the length of me"
$ echo ${var/ /_ }        #在指定字符串之前插入一个字符串
get_ the length of me
$ echo ${var// /_ }
get_ the_ length_ of_ me
$ echo ${var/ / _}        #在指定字符串之后插入一个字符串
get _the length of me
$ echo ${var// / _}
get _the _length _of _me

// 其他的还用演示么？这里主要介绍sed怎么用来插入字符吧，因为它的标签功能很有趣
$ echo $var | sed -e 's/\( \)/_\1/' #\(和\)将不匹配到的字符串存放为一个标签，按匹配顺序为\1,\2...
get_ the length of me
$ echo $var | sed -e 's/\( \)/_\1/g'
get_ the_ length_ of_ me
$ echo $var | sed -e 's/\( \)/\1_/'
get _the length of me
$ echo $var | sed -e 's/\( \)/\1_/g'
get _the _length _of _me

// 看看sed的标签的顺序是不是\1,\2...，看到没？\2和\1掉换位置后，the和get的位置掉换了
$ echo $var | sed -e 's/\([a-z]*\) \([a-z]*\) /\2 \1 /g'
the get of length me
// sed还有专门的插入指令，a和i,分别表示在匹配的行后和行前插入指定字符
$ echo $var | sed '/get/a test'
```

```
get the length of me
test
$ echo $var | sed '/get/i test'
test
get the length of me
```

2.5. 删除子串

删除子串：应该很简单了吧，把子串替换成“空”（什么都没有）不就变成了删除么。还是来简单复习一下替换吧。

概要示例：：把var字符串中所有的空格给删除掉。

鼓励： 这样一替换不知道变成什么单词啦，谁认得呢？但是中文却是连在一起的，所以中文有多难，你想到了么？原来你也是个语言天才，而英语并不可怕，你有学会它的天赋，只要你有这个打算。

```
// 再用{}
$ echo ${var// /}
getthelengthofme
// 再用awk
$ echo $var | awk '{gsub(" ","",$0); printf("%s\n", $0);}'
// 再用sed
$ echo $var | sed 's/ //g'
getthelengthofme
// 还有更简单的tr命令，tr也可以把" "给删除掉，看
$ echo $var | tr -d " "
getthelengthofme
```

如果要删除掉第一个空格后面所有的字符串该怎么办呢？还记得{}的#和%用法么？如果不记得，回到这一节的还头开始复习吧。（实际上删除子串和取子串未尝不是两种互补的运算呢，删除掉某些不想要的子串，也就同时取得另外那些想要的子串——这个世界就是一个“二元”的世界，非常有趣）

2.6. 子串比较

这个很简单：还记得test命令的用法么？man test。它可以用来判断两个字符串是否相等的。另外，你发现了“字符串是否相等”和“字符串能否跟另外一个字符串匹配"两个问题之间的关系吗？如果两个字符串完全匹配，那么这两个字符串就相等了。所以呢，上面用到的字符串匹配方法，也同样可以用到这里。

2.7. 子串排序

差点忘记这个重要的内容了，子串排序可是经常用到的，常见的有按字母序、数字序等正序或反序排列。sort命令可以用来做这个工作，它和其他行处理命令一样，是按行操作的，另外，它类似cut和awk，可以指定分割符，并指定需要排序的列。

```
$ var="get the length of me"
$ echo $var | tr ' ' '\n' | sort   #正序排
get
length
me
of
the
$ echo $var | tr ' ' '\n' | sort  -r #反序排
the
of
me
length
```

get

```
$ cat data.txt
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
41 45 44 44 26 44 42 20 20 38 37 25 45 45 45
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
44 20 30 39 35 38 38 28 25 30 36 20 24 32 33
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
41 33 51 39 20 20 44 37 38 39 42 40 37 50 50
46 47 48 49 50 51 52 53 54 55 56
42 43 41 42 45 42 19 39 75 17 17
$ cat data.txt | sort -k 2 -n
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
44 20 30 39 35 38 38 28 25 30 36 20 24 32 33
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
41 33 51 39 20 20 44 37 38 39 42 40 37 50 50
42 43 41 42 45 42 19 39 75 17 17
41 45 44 44 26 44 42 20 20 38 37 25 45 45 45
46 47 48 49 50 51 52 53 54 55 56
```

2.7. 子串进制转换

如果字母和数字字符用来计数，那么就存在进制转换的问题。在数值计算一节的回复资料里，我们已经介绍了bc命令，这里再简单的复习一下。

```
$ echo "ibase=10;obase=16;10" | bc
A
```

说明：ibase指定输入进制，obase指出输出进制，这样通过调整ibase和obase，你想怎么转就怎么转啦！

2.7. 子串编码转换

什么是字符编码？这个就不用介绍了吧，看过那些乱七八糟显示的网页么？大多是因为浏览器显示时的”编码“和网页实际采用的”编码“不一致导致的。字符编码通常是指把一序列”可打印“字符转换成二进制表示，而字符解码呢则是执行相反的过程，如果这两个过程不匹配，则出现了所谓的”乱码“。

为了解决”乱码“问题呢？就需要进行编码转换。在linux下，我们可以使用iconv这个工具来进行相关操作。这样的情况经常在不同的操作系统之间移动文件，不同的编辑器之间交换文件的时候遇到，目前在windows下常用的汉字编码是gb2312，而在linux下则大多采用utf8。

```
$ nihao_gb2312=$(echo "你好" | iconv -f utf8 -t gb2312)
$ echo $nihao_gb2312
?              ?              ?              ?
$ nihao_utf8=$(echo $nihao_gb2312 | iconv -f gb2312 -t utf8)
$ PS1="$ "
$ echo $nihao_utf8
你好
```

说明：我的终端默认编码是utf8，所以结果如上。

3. 字符串操作范例

实际上，在用Bash编程时，大部分时间都是在处理字符串，因此把这一节熟练掌握非常重要。

3.1 处理一个非常有意义的字符串：URL地址

范例演示：处理URL地址

URL 地址(URL（Uniform Resoure Locator：统一资源定位器）是WWW页的地址)几乎是我们日常生活的玩伴，我们已经到了无法离开它的地步啦，对它的操作很多，包括判断URL地址的有效性，截取地址的各个部分（服务器类型、服务器地址、端口、路径等）并对各个部分进行进一步的操作。

下面我们来具体处理这个URL地址：
ftp://anonymous:ftp@mirror.lzu.edu.cn/software/scim-1.4.7.tar.gz

```
$ url="ftp://anonymous:ftp@mirror.lzu.edu.cn/software/scim-1.4.7.tar.gz"
// 匹配URL地址，判断URL地址的有效性
$ echo $url | grep "ftp://[a-z]*:[a-z]*@[a-z\./-]*"
// 截取服务器类型
$ echo ${url%%:*}
ftp
$ echo $url | cut -d":" -f 1
ftp
// 截取域名
$ tmp=${url##*@} ; echo ${tmp%%/*}
mirror.lzu.edu.cn
// 截取路径
$ tmp=${url##*@} ; echo ${tmp%/*}
mirror.lzu.edu.cn/software
// 截取文件名
$ basename $url
scim-1.4.7.tar.gz
$ echo ${url##*/}
scim-1.4.7.tar.gz
// 截取文件类型（扩展名）
$ echo $url | sed -e 's/.*[0-9].\(.*\)/\1/g'
tar.gz
```

有了上面的知识，我们就可以非常容易地进行这些工作啦：修改某个文件的文件名，比如调整它的编码，下载某个网页里头的所有pdf文档等。这些就作为练习自己做吧，如果遇到问题，可以在回帖交流。相应地可以参考这个例子：

[1] 用脚本下载某个网页中的英文原著(pdf文档）
http://oss.lzu.edu.cn/blog/blog.php?do_showone/tid_1228.html

3.2 处理格式化的文本：/etc/passwd

平时做工作，大多数时候处理的都是一些“格式化”的文本，比如类似/etc/passwd这样的有固定行和列的文本，也有类似tree命令输出的那种具有树形结构的文本，当然还有其他具有特定结构的文本。

关于树状结构的文本的处理，可以考虑看看这两个例子：
[1] 用AWK转换树形数据成关系表
http://oss.lzu.edu.cn/blog/blog.php?do_showone/tid_1260.html
[2] 用Graphviz进行可视化操作——绘制函数调用关系图
http://oss.lzu.edu.cn/blog/blog.php?do_showone/tid_1425.html
实际上，只要把握好特性结构的一些特点，并根据具体的应用场合，处理起来就不会困难。

下面我们来介绍具体有固定行和列的文本的操作，以/etc/passwd文件为例。关于这个文件的帮忙和用户，请通过man 5 passwd查看。下面我们对这个文件以及相关的文件进行一些有意义的操作。

```
// 选取/etc/passwd文件中的用户名和组ID两列
```

```
$ cat /etc/passwd | cut -d":" -f1,4
// 选取/etc/group文件中的组名和组ID两列
$ cat /etc/group | cut -d":" -f1,3
// 如果想找出所有用户所在的组，怎么办?
$ join -o 1.1,2.1 -t":" -1 4 -2 3 /etc/passwd /etc/group
root:root
bin:bin
daemon:daemon
adm:adm
lp:lp
pop:pop
nobody:nogroup
falcon:users
// 先解释一下：join命令用来连接两个文件，有点类似于数据库的两个表的连接。-t指定分割符，"-1 4 -2 3"指定按照第一个文件的第4列和第二个文件的第3列，即组ID进行连接，"-o 1.1,2.1"表示仅仅输出第一个文件的第一列和第二个文件的第一列，这样就得到了我们要的结果，不过，可惜的是，这个结果并不准确，再进行下面的操作，你就会发现：
$ cat /etc/passwd | sort -t":" -n -k 4 > /tmp/passwd
$ cat /etc/group | sort -t":" -n -k 3 > /tmp/group
$ join -o 1.1,2.1 -t":" -1 4 -2 3 /tmp/passwd /tmp/group
halt:root
operator:root
root:root
shutdown:root
sync:root
bin:bin
daemon:daemon
adm:adm
lp:lp
pop:pop
nobody:nogroup
falcon:users
games:users
// 可以看到这个结果才是正确的，所以以后使用join千万要注意这个问题，否则采取更保守的做法似乎更能保证正确性，更多关于文件连接的讨论见参考资料[14]
```

上面涉及到了处理某格式化行中的指定列，包括截取（如SQL的select用法），连接（如SQL的join用法），排序（如SQL的order by用法)，都可以通过指定分割符来拆分某个格式化的行，另外，“截取”的做法还有很多，不光是cut，awk，甚至通过IFS指定分割符的read命令也可以做到，例如：

```
$ IFS=":"; cat /etc/group | while read C1 C2 C3 C4; do echo $C1 $C3; done
```

因此，熟悉这些用法，我们的工作将变得非常灵活有趣。

到这里，需要做一个简单的练习，如何把按照列对应的用户名和用户ID转换成按照行对应的，即把类似下面的数据：

```
$ cat /etc/passwd | cut -d":" -f1,3 --output-delimiter=" "
root 0
bin 1
daemon 2
```

转换成：

```
$ cat a
root    bin     daemon
0       1       2
```

并转换回去，有什么办法呢？记得诸如tr，paste,split等命令都可以使用。

参考方法：
＊正转换：先截取用户名一列存入文件user，再截取用户ID存入id，再把两个文件用paste -s命令连在一起，这样就完成了正转换。
＊逆转换：先把正转换得到的结果用split -l拆分成两个文件，再把两个拆分后的文件用tr把分割符"\t"替换成"\n"，只有用paste命令把两个文件连在一起，这样就完成了逆转换。

更多有趣的例子，可以参考该序列第一部分的回复，即参考资料[16]的回复，以及兰大开源社区镜像站用的镜像脚本，即参考资料[17]，另外，参考资料[18]关于用Shell实现一个五笔反查小工具也值得阅读和改进。

*更多例子将逐步补充和完善。

参考和推荐资料：

[1] 《高级Bash脚本编程指南》之操作字符串
http://www.linuxpk.com/doc/abs/string-manipulation.html
[2] 《高级Bash脚本编程指南》之指定变量的类型
http://www.linuxpk.com/doc/abs/declareref.html
[3] 《Shell十三问》之$(( )) 與 $( ) 還有${ } 差在哪?
http://bbs.chinaunix.net/viewthread.php?tid=218853&extra=&page=7#pid1617953
[4] Regular Expressions - User guide
http://www.zytrax.com/tech/web/regex.htm
[5] Regular Expression Tutorial
http://analyser.oli.tudelft.nl/regex/index.html.en
[6] Grep Tutorial
http://www.panix.com/~elflord/unix/grep.html
[7] Sed Tutorial
http://www.panix.com/~elflord/unix/sed.html
[8] awk Tutorial
http://www.gnulamp.com/awk.html
[9] sed Tutorial
http://www.gnulamp.com/sed.html
[10] An awk Primer
http://www.vectorsite.net/tsawk.html
[11] 一些奇怪的 unix 指令名字的由来
http://www.linuxsir.org/bbs/showthread.php?t=24264
[12] 磨练构建正则表达式模式的技能
http://www.ibm.com/developerworks/cn/aix/library/au-expressions.html
[13] 实用正则表达式
http://www.linuxlong.com/forum/bbs-27-1.html
[14] AWK使用手册 3 楼的回复帖
http://oss.lzu.edu.cn/modules/newbb/viewtopic.php?topic_id=1006&forum=26
[15] 基础11：文件分类、合并和分割(sort,uniq,join,cut,paste,split)
http://blog.chinaunix.net/u/9465/showart_144700.html
[16] Shell编程范例之数值运算
http://oss.lzu.edu.cn/blog/blog.php?do_showone/tid_1391.html
[17] 兰大Mirror镜像站的镜像脚本
http://oss.lzu.edu.cn/blog/article.php?tid_1236.html
[18] 一个用Shell写的五笔反查小工具
http://oss.lzu.edu.cn/blog/blog.php?/do_showone/tid_1017.html
[19] 使用Linux 文本工具简化数据的提取
http://linux.chinaunix.net/docs/2006-09-22/2803.shtml
[20] 如何控制终端：光标位置，字符颜色，背景，清屏...
http://oss.lzu.edu.cn/modules/newbb/viewtopic.php?topic_id=962&forum=13
[21] 在终端动态显示时间
http://oss.lzu.edu.cn/modules/newbb/viewtopic.php?topic_id=964&forum=26

后记：

[1] 这一节本来是上个礼拜该弄好的，但是这些天太忙了，到现在才写好一个“初稿”，等到有时间再补充具体的范例。这一节的范例应该是最最有趣的，所有得好好研究一下几个有趣的范例。
[2] 写完[1]貌似是1点多，刚check了一下错别字和语法什么的，再添加了一节，即“字符串的存储结构”，到现在已经快half past 2啦，晚安，朋友们。
[3] 26号，添加“子串进制转换”和“子串编码转换”两小节以及一个处理URL地址的范例。