目录

shell种类

bash csh ksh zsh

基本语法

定义和使用变量

```
#!/bin/sh
a="hello world"
echo $a
echo 'a is xiaxaiwen${a}'
```

if else

```
if ....; then
   ....
elif ....; then
   ....
else
   ....
fi
```

[] 条件测试

> [] 中前后一定要加空格。

shell常用命令

命令	说明
`echo`	将文字内容打印在屏幕上
`ls`	文件列表
`wc`	计算文件行数(-l),单词数(-w),字符数(-c)
`cp`	文件拷贝
`mv`	重命名文件或移动文件
`rm`	删除文件
`grep`	在文件内搜索字符串比如：`grep 'searchstring' file.txt`

命令	说明
`cut -b`	指定欲显示的文件内容范围，并将它们输出到标准输出设备比如：输出每行第5个到第9个字符 `cut -b5-9 file.txt`，千万不要和cat命令混淆，这是两个完全不同的命令
`cat`	输出文件内容到标准输出设备（屏幕）上
`file`	得到文件类型
`read var`	提示用户输入，并将输入赋值给变量
`sort`	对 file.txt 文件中的行进行排序
`uniq`	删除文本文件中出现的行列比如： `sort file.txt | uniq`
`expr`	进行数学运算，比如运行 `expr 2 "+" 3` 得到结果为 5
`find`	搜索文件比如：根据文件名搜索`find . -name filename -print`
`tee`	将数据输出到标准输出设备(屏幕) 和文件比如：`ls | tee outfile`
`basename`	返回不包含路径的文件名比如： `basename /bin/tux` 将返回 `tux`
`dirname`	返回文件所在路径比如：`dirname /bin/tux` 将返回 `/bin`
`head`	输出文本文件开头几行
`tail`	输出文本文件末尾几行
`sed`	`sed` 是一个基本的查找替换程序。可以从标准输入（比如命令管道）读入文本，并将结果输出到标准输出（屏幕）。该命令采用正则表达式（见参考）进行搜索。不要和shell中的通配符相混淆。比如：将 linuxfocus 替换为 LinuxFocus ：`cat text.file | sed 's/linuxfocus/LinuxFocus/' > newtext.file`
`awk`	`awk` 用来从文本文件中提取字段。缺省地，字段分割符是空格，可以使用 `-F` 指定其他分割符。`cat file.txt | awk -F, '{print $1 "," $3 }'`这里我们使用 `,` 作为字段分割符，同时打印第一个和第三个字段。如果该文件内容如下： Adam Bor, 34, IndiaKerry Miller, 22, 该命令输出结果为：Adam Bor, IndiaKerry Miller, USA
`env`	系统预设的环境变量

shell变量类型

> shell变量名只能包含数字、字母和下划线。

局部变量

在脚本或命令中定义，仅在当前shell实例中有效。

shell变量特殊变量

特殊变量	说明
`$0`	当前脚本的文件名

特殊变量	说明
$n	传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。
$#	传递给脚本或函数的参数个数。
$*	传递给脚本或函数的所有参数。
$@	传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $* 稍有不同，下面将会讲到。
$?	上个命令的退出状态，或函数的返回值。
$$	当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。

条件语句

操作符：

操作符	说明
!!	表示执行上一条指令
!n	表示执行命令历史中第n条指令
*	匹配零个或多个字符
?	匹配一个字符

数值比较运算符：

运算符	说明
-eq	相等
-ne	不等
-gt	大于
-lt	小于
-le	小于等于
-ge	大于等于

字符串比较运算符：

运算符	说明
=	相等
!=	不等
-z	空串
-n	非空串

文件比较运算符：

运算符	说明
-d	目录
-f	文件
-L	链接
-r	可读
-w	可写
-x	可执行
-s	文件非空

逻辑运算符：

运算符	说明
-a	逻辑与
-o	逻辑或
!	逻辑否

反引号` 设置系统的命令输出到变量。 echo * 以串行形式，打印当前整个目录。

替换运算符

- ${var_name:-def_Val} 如果变量var_name存在且为非null，返回该变量的值，否则返回默认值def-Val 注意var_name与:之间没有空格，:与-之间可以有空格。主要用途，如果变量未定义，则用默认值。
- ${var_name:=val} 如果变量var_name存在且为非null，返回该变量的值，否则，把val的值赋给变量var_name，并返回var_name的值val。注意var_name与:之间没有空格，:与=之间也不能有空格。
- ${var_name:?message} 如果变量var_name存在且为非null，返回该变量的值，否则返回该变量的名字var_name:提示信息meesage，并退出当前命令或脚本。 注意 var_name 与 : 之间没有空格，: 与 ? 之间也不能有空格。
- ${var_name:+val} 如果变量var_name存在且为非null，返回val，否则返回null。注意 var_name 与 : 之间没有空格，: 与 + 之间也不能有空格。
- ${#val_name} 返回变量长度。
- $(()) 算术运算操作。
- $((var1 opr var2)) 例如: $((5+1)) 只能是 + - * / 和 () 运算符，并且只能做整数运算。
- $() 命令代换，类似于反引号（`），例如：echo $(date)。

循环语句用法：

- for i in $path
- for i in `seq 1 9`
- for i in $(seq 1 9)
- for i in {a..z}
- for ((i=1; i<=10; i++))

- `while [ $cnt -ge 0 ]`
- `until [ $cnt -lt 0 ]`

case语句

```
name=`basename $0 .sh`
case $1 in
    s|start)
    echo "start..."
    ;;
    stop)
    echo "stop ..."
        ;;
    reload)
        echo "reload..."
        ;;
    *)
        echo "Usage: $name [start|stop|reload]"
        exit 1
        ;;
esac
exit 0
```

> `*)` 相当于其他语言中的 default 。除了 `*)` 模式，各个分支中 `;;` 是必须的，`;;` 相当于其他语言中的 break。`|` 分割多个模式，相当于 or

- `help` 查看所有 bash 保留的关键词。
- `readonly` 定义变量只读。
- `unset` 删除变量。
- `echo -e "Value of a is $a \n"` 使转义字符\n生效，可以使用的转义符：`\\ \a \b \f \n \r \t \v`。
- `${var}` 变量本来的值。
- `${var:-word}` 如果变量 var 为空或已被删除(unset)，那么返回 word，但不改变 var 的值。
- `${var:=word}` 如果变量 var 为空或已被删除(unset)，那么返回 word，并将 var 的值设置为 word。
- `${var:?message}` 如果变量 var 为空或已被删除(unset)，那么将消息 message 送到标准错误输出，可以用来检测变量 var 是否可以被正常赋值。若此替换出现在Shell脚本中，那么脚本将停止运行。
- `${var:+word}` 如果变量 var 被定义，那么返回 word，但不改变 var 的值。
- `val=`expr 2 + 2`` 算数运算。
- `echo ${#string}` 获取字符串长度。
- `echo ${string:1:4}` 输出第1位到第4位的字符串（从0开始）。

数组

定义数组：

```
array_name=(value0 value1 value2 value3)
# 或者
array_name=(
    value0
    value1
    value2
    value3
    )
# 或者
array_name[0]=value0
array_name[1]=value1
array_name[2]=value2
```

数组访问：

```
# 获取数组的值
echo ${array_name[2]}
# 获取数组的所有元素
echo ${array_name[*]}
# 获取数组的所有元素
echo ${array_name[@]}
# 获取数组个数
echo ${#array_name[@]}
# 获取数组个数
echo ${#array_name[*]}
# 获取数组单个元素的长度
echo ${#array_name[2]}
```

重定向

```
# 将stderr重定向到 file
$ command 2 > file
# 将stdin重定向到file
$ command < file
# 将stdout重定向到file
$ command > file
# 将stdout stderr都重定向到file
$ command > file 2>&1
# 将stdin重定向到file1 stdout重定向到file2
$ command < file1 > file2
```

- `-cp` 表示cp命令报错了不会停止，继续执行下面的操作。
- `@cp` 终端中不会打印出命令的执行。
- `declare`

    - `-i` 整型变量。
    - `-a` 数组。
    - `-f` 列出所有定义过的函数。
    - `-x` 将声明的变量作为脚本的环境变量导出。
- `shift` 左移参数，shift 3表示$4变成$1 不带参数的话默认为 `shift 1`。

字符串操作

```
file=/dir1/dir2/dir3/my.file.txt

${file#*/}          # 删掉第一个 / 及其左边的字符串：dir1/dir2/dir3/my.file.txt
${file##*/}         # 删掉最后一个 / 及其左边的字符串：my.file.txt
${file#*.}          # 删掉第一个 . 及其左边的字符串：file.txt
${file##*.}         # 删掉最后一个 . 及其左边的字符串：txt
${file%/*}          # 删掉最后一个 / 及其右边的字符串：/dir1/dir2/dir3
${file%%/*}         # 删掉第一个 / 及其右边的字符串：(空值)
${file%.*}          # 删掉最后一个 . 及其右边的字符串：/dir1/dir2/dir3/my.file
${file%%.*}         # 删掉第一个 . 及其右边的字符串：/dir1/dir2/dir3/my
${file:0:5}         # 提取最左边的5个字节：/dir1
${file:5:5}         # 提取第5个字节右边的连续5个字节：/dir2
${file/dir/path}    # 将第一个 dir 替换为 path：/path1/dir2/dir3/my.file.txt
${file//dir/path}   # 将全部 dir 替换为path：/path1/path2/path3/my.file.txt
${#file}:           # 计算字符串长度：27
```

> 记忆的方法为：`#` 是 去掉左边（键盘上 `#` 在 `$` 的左边）。 `%` 是去掉右边（键盘上 `%` 在 `$` 的右边）。 单一符号是最小匹配；两个符号是最大匹配。

命令中出现的双横杆是什么：`--`

双横杆 `--` 是为了告诉命令，后面的内容不是命令参数，标识命令参数结束（marks the end of options），比如我要生成一个名为 `-f` 的文件：

```
# 使用该命令会保存
$ touch -f
usage:
touch [-A [-][[hh]mm]SS] [-acfhm] [-r file] [-t [[CC]YY]MMDDhhmm[.SS]] file ...

# 加双横杆
$ touch -- -f

# 查看生成的文件
$ ll
-rw-r--r--   1 xxw  staff     0B  9 29 17:43 -f
```

> 参考：https://unix.stackexchange.com/questions/11376/what-does-double-dash-mean .