元字符、引用和变量

主要内容

- **元字符转义和引用：echo和print命令**
- 变量:
  - 变量定义和使用/环境变量/本地化设置
  - export/declare/printenv/set/env/locale命令

元字符: 转义　cmd [options] [arguments]

- shell分析命令，确定命令以及后面的参数部分，执行的程序再决定哪些是选项和参数部分
- 基于shell元字符(包括空格类字符)进行单词分割，然后进行相应的shell扩展
- 如果不想要元字符的特殊含义，而是元字符本身，引入字符转义：
  - 在元字符之前添加一个转义字符表示忽略其后面的字符的特殊含义
  - Bash使用的转义字符为反斜线(\)，如果要传递的字符串本身包含\，则在之前添加一个\（即\\），表示后面的字符为\本身
  - **转义仅针对元字符，不支持通过转义描述（\n换行、\t制表）等控制字符**
  - 注意：如果\后面的字符不是shell元字符，则\被移走，仅保留后面的字符。其他语言中，一般不会移走\

字符串	含义
Linux*	通配符扩展：以Linux开头，后面任意字符
Linux*	Linux*
*\ Linux\nC	字符串的值为* LinuxnC, 注意Bash并不认识\n

```
dlmao@mars:~$ set -x
dlmao@mars:~$ echo \\\*\ Linux\nC
+ echo '\* LinuxnC'
\* LinuxnC
```

元字符：引用

假设有一个文件名为 1 My Heart Will Go On*.mp3

```
dlmao@mars:~$ ls 1 My Heart Will Go On*.mp3
+ ls --color=auto 1 My Heart Will Go 'On*.mp3'
# # 首先分解为参数 1/My/Heart/Will/Go/On*.mp3
# On*.mp3进行文件通配符扩展，展开为匹配的文件名，如果没有，则仍为On*.mp3
ls: cannot access '1': No such file or directory
ls: cannot access 'My': No such file or directory
ls: cannot access 'Heart': No such file or directory
ls: cannot access 'Will': No such file or directory
ls: cannot access 'Go': No such file or directory
ls: cannot access 'On*.mp3': No such file or directory
dlmao@mars:~$ ls 1\ My\ Heart\ Will\ Go\ On\*.mp3
+ ls --color=auto '1 My Heart Will Go On*.mp3'
ls: cannot access '1 My Heart Will Go On*.mp3': No such file or directory
dlmao@mars:~$ ls '1 My Heart Will Go On*.mp3'
+ ls --color=auto '1 My Heart Will Go On*.mp3'
ls: cannot access '1 My Heart Will Go On*.mp3': No such file or directory
```

- 引入引用(quote)的概念。可以采用单引号或者双引号，引号内包含的字符串作为整体当成一个单词看待

引用

- 单引号抑制所有的shell扩展，元字符(包括\)没有特殊的含义，不需要转义
- 双引号抑制几乎所有的shell扩展
  - 单词分割，文件名扩展，波浪号扩展和花括号扩展等失效
  - 参数扩展($), 算术扩展($)和命令替换(`或$) 仍然有效
  - 元字符$(参数和算术扩展)、\(反斜线，转义)和`(反引号，命令替换)
- 引号需要配对，允许跨越多行
- 引用的目的：不是定义字符串，实际上shell只有一种数据类型：字符串
  - 避免那些特殊的字符被解释为bash元字符
  - 引用包含的内容作为一个整体看待。在进行引用时引入的引号本身并不会传递给命令，而是去掉引号后整体传递

```
dlmao@mars:~$ echo '$USER is $USER ; $TERM   is $TERM' # 1个参数
+ echo '$USER is $USER ; $TERM   is $TERM'
$USER is $USER ; $TERM   is $TERM
dlmao@mars:~$ echo "\$USER is $USER ; \$TERM is $TERM" # 1个参数
+ echo '$USER is dlmao ; $TERM is xterm'
$USER is dlmao ; $TERM is xterm
dlmao@mars:~$ echo \$USER is $USER \; \$TERM   is $TERM # 多个参数
+ echo '$USER' is dlmao ';' '$TERM' is xterm
$USER is dlmao ; $TERM is xterm
```

* **echo命令**将后面给出的(多个)参数显示，参数间添加一空格，最后附加一个换行
* **$USER**为参数扩展，表示变量USER的值

跨越多行的命令

- 引号需要配对，允许跨越多行
- 如果反斜线后面为换行符，转义的结果是换行符失去其本来的意义（不标识命令的结束），表示续行
- 交互式Shell中，在分析命令时，发现其跨越多行时，会在下一行开始输出一个2级提示符(PS2)，提醒命令尚未结束
- 1级提示符PS1表示输入命令之前输出的提示符
- echo $PS2可以查看2级提示符的值，缺省为>
- 与其他程序语言一样，除了续行和单双引号引用外，那些条件/循环等程序结构以及子shell等也允许跨越多行

```
demo@mars:~$ echo 'This line ends without a matching quote, not need to escape(\) the new line.
> Here is the missing quote'
This line ends without a matching quote, not need to escape(\) the new line.
Here is the missing quote
demo@mars:~$ echo This is a very, very long \
> line that goes on and on for a \
> very, very long time.
This is a very, very long line that goes on and on for a very, very long time.
```

echo命令：输出参数

echo [-neE] [STRING]...

将后面的(多个)参数输出，有多个参数时，参数间以空格隔开，最后附加一个换行

- -n 表示最后不附加换行符
- -e 表示后面的参数允许字符转义
- -E 表示不支持字符转义，缺省为采用-E选项

```
~$ echo -e '1\tpython\n2\tlinux'
1       python
2       linux
~$ echo -e '\e[04m\060 \x31 \u9999\e[0m'
0 1 香
```

\转义	含义	\转义	含义
\a	响铃(警告)	\n	换行
\b	退格	\r	回车
\t	制表符	\v	纵向制表
\e \E	ESC字符，相当于\033	\\	反斜杠
\f	换页	\c	cancel，表示当前行及后续行内容不输出。出现在第一行时表示此后内容不输出
\0nnn	ASCII为八进制的nnn（1到3个八进制数字）的字符		
\xHH	ASCII为十六进制的HH（1到2个十六进制数字）的字符		
\uHHHH	Unicode为4位十六进制的数字HHHH的字符		
\UHHHHHHHH	Unicode为8位十六进制的数字HHHHHHHH的字符		

格式化输出 printf

printf [-v var] FORMAT [ARGUMENT]...　　man 1 printf

- 提供C语言中printf函数类似的功能， man 3 printf
- 选项-v var: 格式化的结果存放在var中而不是标准输出中
- FORMAT为格式化字符串，其中
  - 普通字符原封不动
  - 允许通过转义方式描述控制字符，比如\t \n等，相比函数printf()，引入\e \E表示\033(ESC字符)
  - 格式定义%，可指定对齐/宽度和精度，后面对应位置的参数按照指定的方式转换后输出
- 在%s %d %f %e等格式外，新增两种格式：
  - %b 识别参数中的用转义方式描述的转义序列，将其转换为对应的字符
  - %q 将参数中的shell元字符等特殊字符加上转义字符使其可在shell中使用

```
demo@mars:~$ printf "%-5s %-10s %-6s\n" \# Name Score ; printf "%-5s %-10s %-4.2f\n" 3 Jeff 77.564
#     Name       Score
3     Jeff       77.56
demo@mars:~$ printf 'I %blove%b Linux\n' '\e[31m' '\e[0m'
I love Linux

demo@mars:~$ printf 'I love %q\n'  '*Linux*'
I love \*Linux\*
```

Bash支持的引用 $'…'

- 与传统的单引号引用类似，不支持任何shell扩展，只是它允许采用转义方式描述字符，比如\n \t 分别表示换行、制表
- 具体支持的转义形式可以参见echo命令中的-e选项支持的转义序列

```
IFS=$' \t\n'
# 单词分割扩展所使用的字符，缺省为空格、制表符和换行符
```

```
$ echo $'\033[01;32mGreen and Bold\n\e[0mNormal'
Green and Bold
Normal
$ echo -e '\033[01;32mGreen and Bold\n\e[0mNormal'
Green and Bold
Normal
```

主要内容

- 元字符转义和引用：echo和print命令
- **变量：**
  - 变量定义和使用/环境变量/本地化设置
  - export/declare/printenv/set/env/locale命令

变量（variable）

- 变量是一个用来存储数据的实体，每个变量通过一个变量名标识，而变量的值就是存储在变量中的数据
- 变量名要求必须由大小写英文字母(A-Za-z)、数字（0-9）或者下划线(_)组成，且第一个字符必须是字母或者下划线，不能是数字
  - 变量名TERM、DISPLAY、tz、var_1、_first等是合法的，但是2var是不合法的
  - 变量名是大小写敏感的，PATH和path是两个不同的变量
- shell只有一种数据类型，那就是字符串，没有整数和浮点类型
  - 不需要像其他程序语言一定要通过引号来定义字符串字面量
  - 引号的目的是减少对于shell元字符的转义，而且引号包含的内容作为一个整体看待
- 变量没有类型的概念，所有变量都是字符串类型，当然也不需要声明其类型
  - bash实际上还支持(一维)数组类型，比如array=(1 2 3)

变量定义和使用

```
var=value     # 变量var保存了字符串value
var= value    # 命令为var=  参数为value
var =value    # 命令为var，参数为=value
var = value   # 命令为var，参数为=和value
```

- 创建变量或者给变量赋值
  - var=value，注意=前后不能有空白字符
  - var= 表示设置变量值为空字符串(null)，也可以采用var=''
  - Bash支持一次给多个变量赋值，在其他shell中可能不支持

    var1=value1 var2=value2
  - 如果value中有空格，应该使用转义或者引用方式，比如var='hello world'
- 删除变量： unset name…
- 使用变量(变量扩展)
  - $var或${var} 获得变量var的值， 花括号用于确认var的界限
    - 考虑 $var1 和 ${var}1的区别
  - 变量没有定义时，$var不会报错，而是替换为空字符串

变量定义和使用

问题：多个字符组成的文本(比如var)到底表示是变量的名字还是字符串呢？

一个名字，比如var，如果出现

- 在赋值语句中，=左边表示变量名，=右边表示字符串字面量
- unset命令的参数部分，表示变量名
- 在$之后，表示变量var的值
- 其他地方一般表示字符串

```
dlmao@mars:~$ COURSE=/home/demo/linux_course NAME='Linux Operating System'
dlmao@mars:~$ echo NAME $NAME
NAME Linux Operating System
dlmao@mars :~$ echo ${NAME}s $COURSE
Linux Operating Systems /home/demo/linux_course
dlmao@mars:~$ unset NAME
dlmao@mars:~$ echo NAME:$NAME
NAME:
```

变量扩展和引用

```
$ song="My song.mp3"
$ ls $song
ls: cannot access 'My': No such file or directory
ls: cannot access 'song.mp3': No such file or directory
$ ls "$song"
ls: cannot access 'My song.mp3': No such file or
directory
```

- ls $song 进行变量替换，等价于ls My song.mp3，变量替换后接下来还要进行单词分割和通配符扩展等
- 通过引号将字符串作为一个整体，不会进行单词分割和通配符扩展，该单词作为ls的第一个参数
- 单引号不支持变量扩展，而双引号支持变量扩展

  ls "My song.mp3"

建议： 所有$扩展都采用双引号引用避免单词分割和通配符扩展

特殊参数(parameter)

- $扩展称为参数扩展，变量是有名字的参数
- 不能直接给这些特殊的参数赋值
- $0 $1, $2… $# $* $@，用于bash脚本中，表示命令行参数(第0个、1个… 参数个数，所有参数），**暂时不讲**
- $$ 当前shell的PID(进程ID)
- $? 上一命令执行返回的状态码，0表示成功
- $! 最近后台执行的进程PID
- $_ 表示上个命令的最后一个参数，相当于!!:$

```
$ ls /binn
ls: cannot access '/binn': No such file or directory
$ echo $?
2
$ mkdir -p unix/bin
$ cd $_    # 等价于 cd unix/bin
```

环境变量和Shell变量

- 环境变量属于”全局“（export属性）变量
  - 可以在创建(定义)它们的shell及其派生出来的任意子进程 (执行命令的进程）继承使用
  - 但是子进程中对于其环境变量的修改不会反馈回父进程
- shell变量属于”局部“（没有export属性）变量，只在创建它们的shell中可用
  - 有些shell变量是供shell使用的，存放了对shell本身有意义的信息。如前面介绍过的IGNOREEOF可确定是否忽略CTRL-D产生的EOF信号
  - 有些shell变量是用户创建的，用于临时保存数据
- 环境变量和shell变量在同一个名字空间
  - 一个变量是环境变量，只不过其具有export属性而已
  - 没有标记export属性的变量属于shell变量
- shell函数内可以通过local命令定义真正意义的局部变量，表示只在函数内部有效，比如 local tmp tmp2=tmp.txt

环境变量

export [-fnp] [name[=value] ...]

内置命令export，给变量name标记属性为exported，成为环境变量

- 如果传递了相应的值(value)，则先给变量赋值，再标记exported
- 如果不传递任何参数，相当于export -p，显示所有环境变量，以declare –x name=value形式
- -n选项表示去除exported属性

declare [-aAfFgiInrtux] [-p] [name[=value] ...]

内置命令declare，给变量赋值和设置或取消变量属性

- 主要属性包括：是否只读-r 、整数类型-i、数组类型-a、环境变量-x等
- declare -x 查看环境变量
- declare -p 查看所有变量(包括环境变量，不包括函数） declare -f 查看shell函数

printenv [VARIABLE]... 外部命令printenv，查看所有环境变量或者指定的环境变量

命令	说明
export BASE=~/root NAME="Linux "	给两个变量赋值，并且属性为环境变量
H=demo	变量H赋值
export H或declare –x H	变量H属性为环境变量
export 或declare –x或printenv或env	查看环境变量

常用环境变量

环境变量	含义
DISPLAY	X服务器的地址
PWD	当前工作目录
HOME	用户主目录
PATH	用于查找执行程序的目录列表，冒号分隔
LD_LIBRARY_PATH	库文件所在目录
TERM	正在使用的终端类型
USER	当前用户
LOGNAME	当前登录用户
SHELL	当前SHELL路径
EDITOR	默认编辑器
PAGER	默认文本文件分页查看程序
LANG	当前本地化环境，比如zh_CN.UTF-8
LANGUAGE	应用程序的界面使用的语言，消息(出错菜单等)的本地化翻译,可同时设置多种语言信息

寻找要执行的命令

- 命令别名(alias): 解析别名，找到真正要执行的命令
- shell内置命令: 在shell进程内运行该命令，比如cd/history/alias/type/exit/export等
  - 在当前Shell环境下执行，影响当前shell环境
- shell函数: 一段小的shell脚本，同样在当前shell的上下文环境中运行
- 外部命令: 可以独立运行的可执行程序。shell找到对应的程序然后:
  - 启动一个新的子进程来执行。该子进程继承父进程（这里为Shell）的环境
  - 子进程执行过程中环境的改变与父进程无关
- 命令解析顺序:
  - 如果命令名前包含路径部分，明确指定是要执行哪个目录中的哪个命令。如/usr/bin/lscpu 或./args
  - 否则：首先看是否为命令别名，接下来看是否为shell函数，然后看是否为shell内置命令，如果都找不到则在PATH指出的目录中查找
- 命令执行完后会返回一个状态码，可通过 echo $?查看，返回0表示成功

```
dlmao@mars:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
dlmao@mars:~$ PATH=$PATH:~/bin:.
#将.目录加入到PATH环境变量，表示最后会在当前目录下寻找
```

特殊的shell变量

```
dlmao@mars:~$ CDPATH=.:/usr/share/doc:~
dlmao@mars:~$ echo $RANDOM
1271
```

CDPATH	**cd命令时搜索的目录列表，冒号分隔**
IFS	**shell进行单词分割时使用的分隔符**
PS1	**第1级提示符，等待用户输入命令**
PS2	第2级提示符，等待用户输入命令的后续部分
BASH_VERSION	bash版本
SHELLOPTS	当前shell已开启的shell选项
HISTCONTROL/HISTFILE/HISTFILESIZE/HISTIGNORE/HISTSIZE 历史记录相关	
HOSTNAME	主机名
IGNOREEOF	**忽略EOF字符的次数**
INPUTRC	readline初始化文件，缺省~/.inputrc
RANDOM	**返回0到32767间的随机整数，给RANDOM赋值，相当于设置seed**
SECONDS	当前bash已经运行多少秒
TMPDIR	用于保存创建的临时文件的目录

shell选项

- shell选项控制bash的行为
  - set –o以容易阅读方式显示shell选项的当前状态
  - set +o也显示当前状态，只是每行对应的是用于变为当前状态而所执行的set命令
  - set –o option（设置选项，设为on状态）
  - set +o option（关闭选项，设为off状态）
- shopt命令可查看和设置更多的shell选项
  - shopt查看选项
  - shopt –p 查看选项，只是以怎样变为当前取值的命令方式，类似于set +o
  - shopt –s option开启选项
  - shopt –u option关闭选项

```
$ set -o
allexport       off
braceexpand     on
….
$ set +o
set +o allexport
set -o braceexpand
…
```

```
$ shopt
autocd          off
…
$ shopt -p
shopt -u autocd
$ shopt -u extglob #关闭扩展通配符
$ shopt -s extglob #打开扩展通配符
```

set命令

set [-abefhkmnptuvxBCHP] [-o option-name] [--] [arg ...]

1. 显示所有变量(包括函数)以及变量的值： set
2. 查看和设置(开启和关闭)shell选项，set -o/set -o option/set +o option
3. 设置位置参数： set -- args1 args2 …

选项	set命令	含义
braceexpand	set –B	波浪号扩展
emacs或vi		命令行编辑采用哪种风格
history		允许纪录命令历史
histexpand	set –H	允许命令历史扩展
ignoreeof		是否忽略EOF信号
monitor	set -m	开启作业控制
noclobber	set -C	重定向时不允许覆盖
noglob	set –f	关闭文件通配符扩展
notify	set –b	作业结束时立刻通知
xtrace	set -x	在执行时输出命令以及参数
pipefail	组成管道的命令中最后一个执行不成功的命令的状态码作为管道的状态码，都成功时为0	

shopt选项	含义
dotglob	文件通配符扩展中*?可匹配第一个点
extglob	打开扩展文件通配符扩展
nullglob	通配符扩展没有匹配时扩展成空字符串，而不是扩展模式本身

set +x 关闭xtrace选项

查看变量

- 通过赋值语句可以给变量赋值
- 通过unset命令删除变量
- 通过declare -x或export标记为环境变量

- 知道名字，可通过 echo $var查看
- **内置命令export/declare -x可查看所有的环境变量**
- **外部命令env或printenv可查看所有的环境变量**
- 内置命令set/declare查看包括函数在内的所有变量
- **declare -p 查看所有变量(包换环境变量，不包括函数)**
- declare -f 查看函数

sort命令对于文本行进行排序

```
demo@mars:~$ printenv | sort
HOME=/home/demo
LANG=en_US.UTF-8
LANGUAGE=en_US:en
LESSCLOSE=/usr/bin/lesspipe %s %s
LESSOPEN=| /usr/bin/lesspipe %s
LOGNAME=demo
MAIL=/var/mail/demo
PATH=/usr/local/sbin:/usr/local/bin:/usr/sb
in:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin
PWD=/home/demo
QT_QPA_PLATFORMTHEME=appmenu-qt5
SHELL=/bin/bash
SHLVL=1
SSH_CLIENT=192.168.1.150 59684 22
SSH_CONNECTION=192.168.1.150          59684
192.168.1.155 22
SSH_TTY=/dev/pts/8
TERM=xterm
USER=demo
_=/usr/bin/env
XDG_RUNTIME_DIR=/run/user/1000
XDG_SESSION_ID=19
```

指定执行某个命令时所使用的环境变量

env [OPTION]... [NAME=VALUE]... [COMMAND [ARGS]...]

- -u NAME, --unset=NAME　将NAME从环境中移走
- -, -i, --ignore-environment　初始环境变量为空

外部命令env 设置相应的变量，这些变量并不作用于当前shell，用于执行COMMAND [ARGS]

如果不包含命令部分，显示最终的环境变量

[NAME=VALUE]... [COMMAND [ARGS]...]

- bash在执行命令时允许指定环境变量
- 设置相应的变量，用于执行COMMAND [ARGS]

```
dlmao@mars:~$ date
2020年 03月 24日 星期二 09:06:59 CST
dlmao@mars:~$ env LC_TIME=C date
Tue Mar 24 09:08:48 CST 2020
dlmao@mars:~$ env -i LC_TIME=C
LC_TIME=C
dlmao@mars:~$ LC_TIME=C date
Tue Mar 24 09:06:59 CST 2020
dlmao@mars:~$ LC_TIME=en_GB.UTF-8 date
Tue 24 Mar 09:28:55 CST 2020
```

本地化设置locale

- Locale是软件在运行时的语言环境，可以通过语言(Language)_地域 (Territory) .字符集(Codeset)来描述某一个地域内的人们的语言习惯、文化传统和生活习惯
  - zh_CN.UTF-8 表示区域为中国，语言为汉字，字符集为UTF-8
  - en_US.UTF-8 表示区域为美国，语言为英语，字符集为UTF-8
  - C表示没有进行本地化，而是传统的美国ANSI C区域
- locale进一步分为12大类，可以设置相应的环境变量来单独确定其所使用的习惯

```
# man 7 locale 查看LC环境变量的含义
# 目录/usr/share/i18n包含了internationalization的相关文件
dlmao@mars:~$ ls -F /usr/share/i18n
charmaps/  locales/  SUPPORTED
```

- 系统管理员可修改本地化设置

dpkg-reconfigure locales

LC_ADDRESS	街道和邮局地址格式
LC_COLLATE	比较和排序习惯
LC_CTYPE	字符类以及大小写转换等
LC_MONETARY	货币单位的格式化
LC_MEASUREMENT	度量表达方式
LC_MESSAGES	消息显示及响应所采用的语言格式
LC_NUMERIC	数字格式化
LC_PAPER	缺省纸张大小
LC_RESPONSE	响应(Yes/No)如何呈现
LC_TELEPHONE	电话号码如何表示
LC_TIME	日期和时间的格式
LC_IDENTIFICATION	locale自身包含信息的概述

locale命令:查看当前全局locale配置

locale [OPTION...] NAME

locale [OPTION...] [-a|-m]

locale	查看当前locale配置
locale –a -v	列出当前可用的locale，-v表示查看详细信息
locale –m	列出当前可用的字符集
locale LC_CTYPE	查看当前LC_CTYPE的相关信息

LANGUAGE	消息(出错、菜单等)的本地化翻译,可同时设置多种语言信息，以冒号隔开，比如en_US:zh_CN
LC_ALL	替代除LANGUAGE外的其他locale环境设置，优先级最高
LANG	LC_*没有设置时使用的默认值

- 本地化环境变量的优先级： LC_ALL>LC_*>LANG
- 用户进一步可在其bash配置文件(.bashrc)中设置环境变量来定制自己的locale设置，比如

  export LANG=zh_CN.UTF-8

  export LC_COLLATE=C.UTF-8

  export LANGUAGE=en_US LC_MESSAGES=C.UTF-8
- 建议设置LANG来给出LC_*的默认值，然后通过LC_XXX具体微调某个大类习惯
- LC_ALL一般仅在执行某个命令时使用

  LC_ALL=C command

```
dlmao@mars:~$ locale
LANG=zh_CN.UTF-8
LANGUAGE=en_US
LC_CTYPE="zh_CN.UTF-8"
LC_NUMERIC="zh_CN.UTF-8"
LC_TIME="zh_CN.UTF-8"
LC_COLLATE="zh_CN.UTF-8"
LC_MONETARY="zh_CN.UTF-8"
LC_MESSAGES="zh_CN.UTF-8"
LC_PAPER="zh_CN.UTF-8"
LC_NAME="zh_CN.UTF-8"
LC_ADDRESS="zh_CN.UTF-8"
LC_TELEPHONE="zh_CN.UTF-8"
LC_MEASUREMENT="zh_CN.UTF-8"
LC_IDENTIFICATION="zh_CN.UTF-8"
LC_ALL=
```