企业 shell 脚本实用技巧

别人写命令，你也写命令，命令知多少？

内部命令：集成在 bash 的命令，就是内部命令。内部命令依赖于 shell 类型。这些命令由 shell 程序识别并在 shell 程序内部完成运行，通常在 linux 系统加载运行时 shell 就被加载并驻留在系统内存中。内部命令是写在 bash 源码里面的，其执行速度比外部命令快，因为解析内部命令 shell 不需要创建子进程。

外部命令：外部命令是在 bash 之外额外安装的，在文件系统路径 $PATH 有对应的可执行程序文件，就是外部命令。在系统加载时并不随系统一起被加载到内存中，而是在需要时才将其调用内存。

命令别名：在管理和维护 Linux 系统的过程中，将会使用到大量命令，有一些很长的命令或用法经常被用到，重复而频繁地输入某个很长命令或用法是不可取的。这时可以使用命令别名功能将这个过程简单化。

hash：系统初始 hash 表为空，当外部命令执行时，默认会从 PATH 路径下寻找该命令，找到后会将这条命令的路径记录到 hash 表中，当再次使用该命令时，shell 解释器首先会查看 hash 表，存

在将执行之，如果不存在，将会去 PATH 路径下寻找。利用 hash 缓存表可大大提高命令的调用速率。

function： 函数是存在内存里的一组代码的命名的元素。函数创建于脚本运行环境之中，并且可以执行

compound commands：在 shell 中指循环、判断、分支、选择、的表达式命令

命令查找方式/命令的优先级

1. 获取一个命令执行的优先级别，至上往下

alias

compound commands

function

build_in

hash

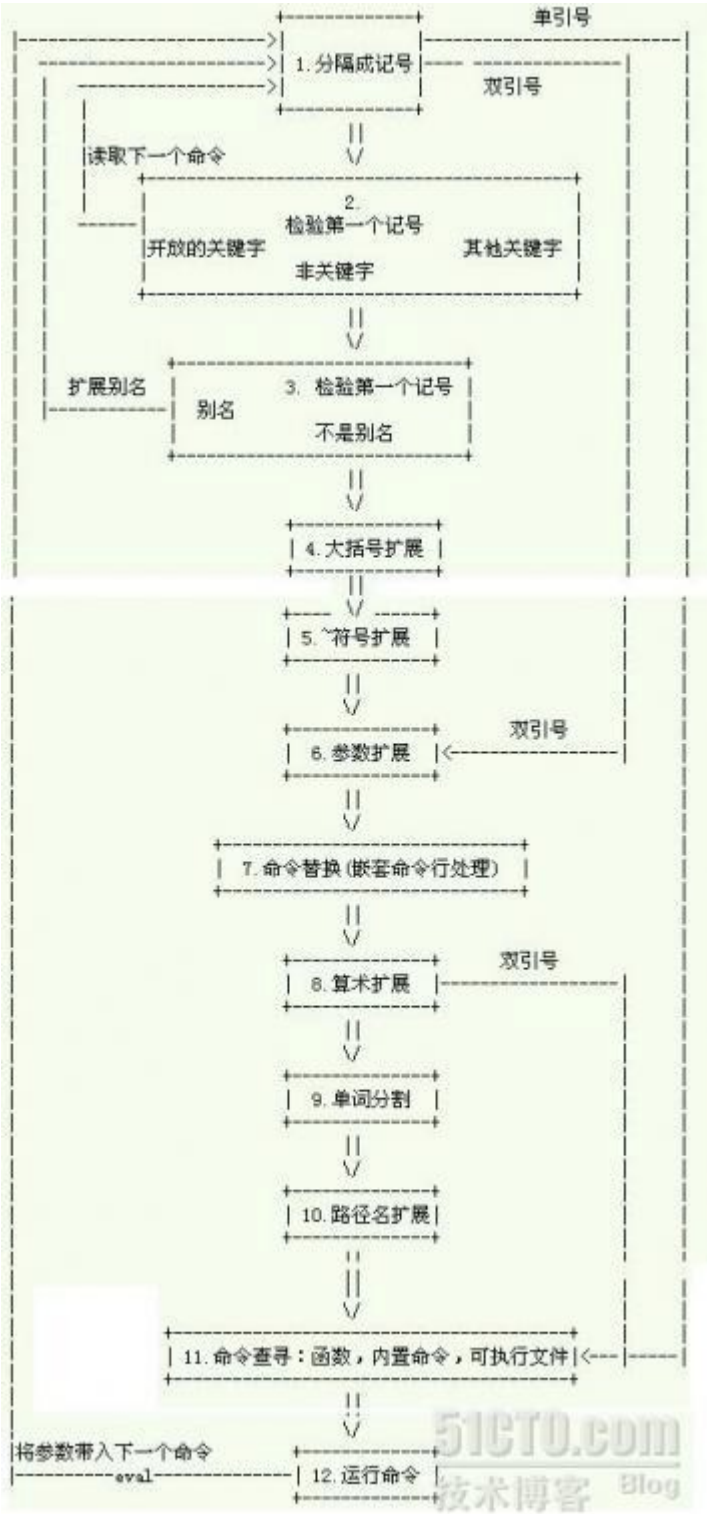
$PATH

error：command not found

练习：

验证命令的优先级

命令解析顺序



Shell 从标准输入或脚本中读取的每行称为一个管道行，它包含一个或多个由 0 个或多个管道字符(|)分隔的命令。对每一个管道行，进行 12 个步骤的处理。

结合上面的插图，这里给出命令行的 12 个步骤。

1. 将命令行分成由固定元字符集分隔的记号 ：

SPACE, TAB, NEWLINE, ; , (,), <, >, |, &

记号类型包括单词，关键字，I/O 重定向符和分号。

2.检测每个命令的第一个记号，查看是否为不带引号或反斜线的关键字。 如果是一个开放的关键字，如 if 和其他控制结构起始字符串，function，{或(，则命令实际上为一复合命令。shell 在内部对复合命令进行处理，读取下一个命令，并重复这一过程。如果关键字不是复合命令起始字符串(如 then 等一个控制结构中间出现的关键字)，则给出语法错误信号。

3.依据别名列表检查每个命令的第一个关键字。 如果找到相应匹配，则替换其别名定义，并退回第一步;否则进入第 4 步。该策略允许递归别名，还允许定义关键字别名。如 alias procedure=function

4.执行大括号扩展 ，例如 a{b,c}变成 ab ac

5.如果~位于单词开头，用$HOME 替换~。使用 usr 的主目录替换~user。

6.对任何以符号$开头的表达式执行参数(变量)替换

7.对形式$(string)的表达式进行命令替换
这里是嵌套的命令行处理。

8.计算形式为$((string))的算术表达式

9.把行的参数，命令和算术替换部分再次分成单词，这次它使用$IFS 中的字符做分割符而不是步骤 1 的元字符集。

10.对出现*, ?, [/]对执行路径名扩展，也称为通配符扩展

11. 按命令优先级表(跳过别名)，进行命令查寻

12.设置完 I/O 重定向和其他操作后执行该命令。

关于引用

1. 单引号跳过了前 10 个步骤，不能在单引号里放单引号

2. 双引号跳过了步骤 1~5，步骤 9~10，也就是说，只处理 6~8 个步骤。

也就是说，双引号忽略了管道字符，别名，~替换，通配符扩展，和通过分隔符分裂成单词。

双引号里的单引号没有作用，但双引号允许参数替换，命令替换和算术表达式求值。可以在双引号里包含双引号，方式是加上转义符"/"，还必须转义$, `, /。

简单备份 web 脚本

```
#!/bin/bash
# web backup script

# Variables and Function definition
PATH=$PATH:/sbin:/usr/sbin:/usr/local/bin
FTPHOST='192.168.1.254'
FTPUSER='ftpuser'
FTPPASSWORD='ftppasswdxxx'
BACKUP_DATE=$(date -d '1 day ago' +%F)
```

```
# Program Main
cd  /var/backup
find . -type f -mtime +2 -exec rm -f {} \;
tar czf web-${BACKUP_DATE}.tar.gz /var/www/html

# Update Data File
ftp -i -n <<EOF
open ${FTPHOST}
user ${FTPUSER} ${FTPPASSWORD}
binary
mkdir ${BACKUP_DATE%-*}
cd  ${BACKUP_DATE%-*}
put web-${BACKUP_DATE}.tar.gz
bye
EOF
```

备份 mysql 脚本

删除日志脚本

内部命令：集成在 bash 的命令,依赖 shell 类型，没有可执行程序
内部命令查看：enable [-n 关闭 -s 开启 cd]
内部命令查看帮助 man bash   命令 --help
重命令 echo，还是能用,因为 echo 既是内部命令，也是外部命令
/usr/bin/cd 内部命令不能写绝对路径 因为会开启子进程
source /usr/bin/cd /etc 或者./usr/bin/cd /etc 不开子进程
vim /usr/bin/cd   builtin 调用？可以没有可执行文件   例如：which set
sleep 12345 && pstree -p   进程树查看是否开启进程
管道与小括号可以开启子进程，bash 就是开启一个子进程
vim /etc/passwd   将 root 的/bin/bash 改成/bin/rbash   && ln -s bash rbash
额外安装的： $PATH 需要时才会被调用到内存中 同时 hash 表也会记录
hash:系统初始 hash 表为空，hash 缓存记录调用外部命令 移动路径会导致缓存调用错误
执行 hash 可查看 hash -l
hash -p 自己脚本 ls --->重命名   hash -d
函数脚本与外部命令相同
别名：alias cd="vim"
复合命令：select i in 1 2 3;do echo $i;done
函数：function vim(){
> echo haha
> }
alias > compound(复合命令:if for..) unset> 函数 unset > build_in（内部命令）> hash（外部缓存） > $PATH（外部命令） > error

vim ff
#!/bin/bash
echo haha ; id root
bash <ff   执行
bash <ff >ff   为空执行
$IFS 换行符
#!/bin/bash
touch abc
echo haha