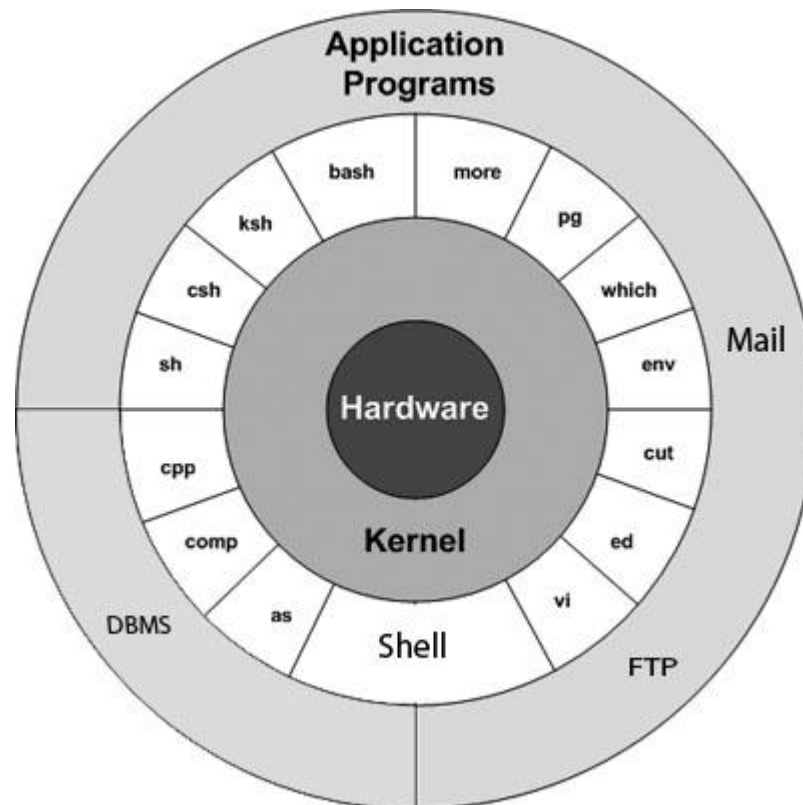


Shell脚本编程基础



内容

①

- Shell概述

- Shell的概念
- Shell的特点
- Shell版本
- Shell程序示例
- Shell程序执行方法

- Linux命令帮手

- 命令行编辑
- screen工具

②

- Shell特殊字符

- 通配符
- 正则表达式
- 引号
- 转义符与路径符
- 输入输出重定向
- 注释和后台命令
- 命令组合符
- 成组命令

③

- Shell变量

- 用户自定义变量
- 位置变量
- Shell预定义变量
- 环境变量

- 算术运算

- 控制结构

- 函数

④

- 作业控制

- Shell内置命令

- Shell脚本调试

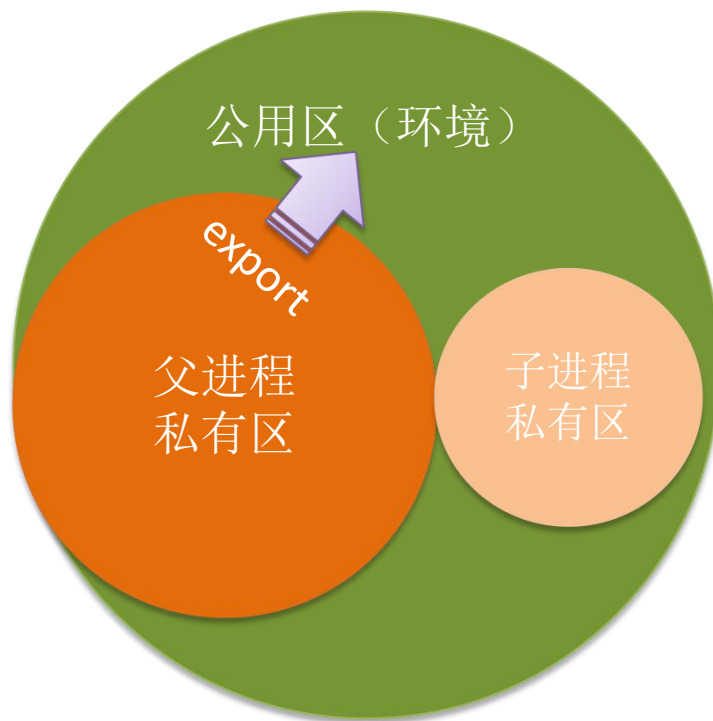
- Shell脚本示例

⑤

进程、父进程与子进程

- **进程**，就是程序的一次执行过程
- 由一个进程创建的另一个进程，前者叫**父进程**，后者是前者的**子进程**
- 会**产生子进程的几种情况**：
 - &，提交后台作业
 - 管道，管道里的每一个命令在一个单独的子进程里执行
 - 圆括号命令列表()
 - 执行外部命令或程序

私有变量和公用变量



作业控制

- **Ctrl + z** 暂停正在运行的命令
- **fg** 将后台程序放到前台
- **bg** 将暂停的程序放到后台继续执行
- **&** 放在命令尾部，后台执行
- **jobs** 查看当前在后台运行的命令
- **kill** 终止后台进程
- **Ctrl + c** 终止前台进程

shell内置命令

- **内置命令**：内置在shell代码中，执行时无需到磁盘上定位，速度快，如cd
- **外部命令**：存放在磁盘上，如/bin、
/usr/bin目录下，执行时需从PATH定义的路径中查找，如date

shell内置命令

- :
- alias
- break
- bg
- bind
- builtin
- cd
- continue
- declare
- dirs
- disown
- echo
- enable
- eval
- exec
- exit
- export
- fc
- fg
- getopts
- hash
- help
- history
- jobs
- kill
- let
- local
- logout
- popd
- pushd
- pwd
- read
- readonly
- return
- set
- shift
- stop
- suspend
- test
- times
- trap
- type
- typeset
- ulimit
- umask
- unalias
- unset
- wait

eval命令

- 强制shell对命令行进行两次扫描，执行变量或命令替换，如：

```
a=protein
```

```
b="$a
```

```
echo $b          #$a
```

```
eval echo $b     #protein
```

```
eval echo "$b    #$a
```

exec、source和.命令

- 三个命令都不会启动新的进程
- `exec`用要被执行命令替换当前的shell进程，并且将老进程的环境清理掉，如：

`exec ls`

执行ls后，shell将退出，因此exec通常放在脚本里面使用，而不是直接在命令中用

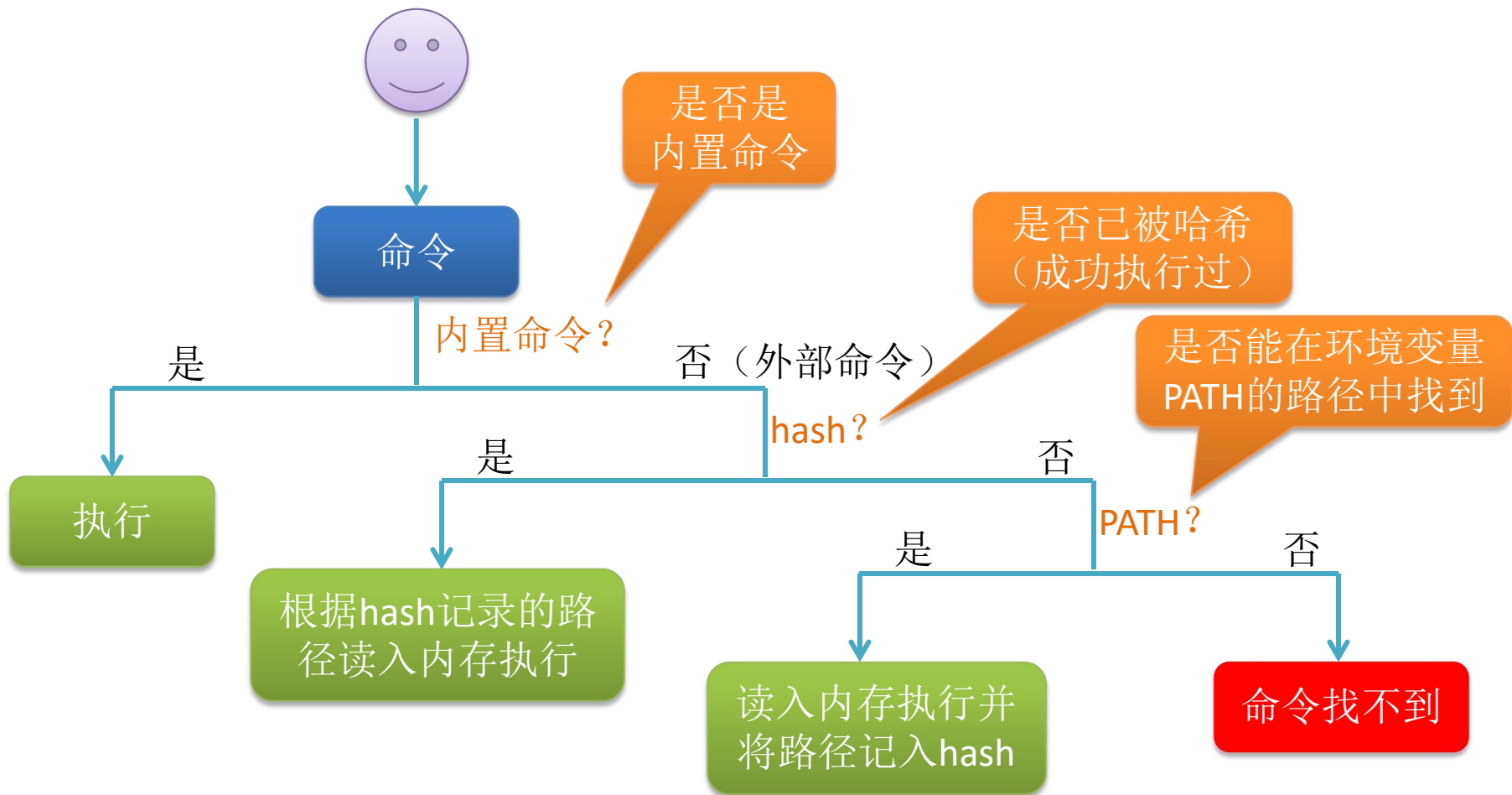
- `source`命令或者`.命令`，不会为脚本新建shell，而只是将脚本包含的命令在当前shell执行，执行完返回当前shell如：

`source test.sh`

hash命令

- Linux为了提高在PATH路径中查询命令的速度，采用了命令哈希表来记录成功执行过的外部命令。当shell在PATH中找到一个命令时，将这个命令及其路径放入哈希表。用户输入一个外部命令时，shell将先在哈希表中进行查询，哈希表中不存在时再去PATH指定的路径中去寻找。
- hash命令列出当前登录shell记录的使用过的外部命令的路径及相关调用次数等信息。加-r选项为清空命令哈希表。

总结：Shell命令执行过程



unset命令

- 删除已定义的变量，如：

```
$a=protein
```

```
$echo $a
```

```
protein
```

```
$unset a      #变量名前不能有$
```

```
$echo $a
```

readonly命令

- 设置变量为只读，后面不能再赋值语句改变该变量的值，防止一个变量被修改。如：

```
$a=protein
```

```
$readonly a      #变量名前不能有$, 参考export与unset
```

```
$a=CDS
```

```
-bash: a: 只读变量
```

```
$unset a
```

```
-bash: unset: a: 无法反设定: 只读 variable
```

type命令

- 显示一个命令的类型或如何被解释的，如：

\$type cd

-cd 是 shell 内嵌

\$type ls

-ls 是 'ls --color=auto' 的别名

\$type man

-man 已被哈希 (/usr/bin/man)

wait命令

- 等待一个进程结束，如：

`$wait 3702` #等待PID为3702的进程结束

`$wait` #等待当前shell所有活动的进程结束

shell脚本调试

- 注释掉部分代码
- 利用echo、tee等输出关键信息，跟踪变量的值
- 使用trap命令捕捉信号
- 利用shell脚本调试选项：-c，-n，-x，-v
- 使用bash专用调试器，如bashdb

shell脚本调试

- 注释掉部分代码
- 利用echo、tee等输出关键信息，跟踪变量的值
- 使用trap命令捕捉信号
- 利用shell脚本调试选项：-c，-n，-x，-v
- 使用bash专用调试器，如bashdb

trap命令

- **trap**命令用于指定在接收到信号后要执行的命令。**trap**命令的一种常见用途是在脚本程序被中断时完成清理工作。如：

```
trap "echo Terminated; rm tmp_files" SIGINT
```

- 常用信号（系统产生的信号）：

SIGHUP(1)	挂起，通常因终端掉线或用户退出而引发
SIGINT(2)	中断，通常因按下Ctrl+C组合键而引发
SIGQUIT(3)	退出，通常因按下Ctrl+\组合键而引发
SIGABRT(6)	中止，通常因某些严重的执行错误而引发
SIGALRM(14)	报警，通常用来处理超时
SIGTERM(15)	终止，通常在系统关机时发送

- **Kill -l**可以列出所有的信号

trap命令

- 伪信号（shell产生的信号）

- EXIT 从一个函数中退出或整个脚本执行完毕
- ERR 当一条命令返回非零值时(代表命令执行不成功)
- DEBUG 脚本中每一条命令执行之前

shell脚本调试选项

- **-c**: 使Shell解析器从字符串而非文件中读取并执行命令

```
$sh -c 'a=1;b=2;let c=$a+$b;echo "c=$c"'
```

- **-n**: 读一遍脚本中的命令但不执行，用于检查脚本中的语法错误
- **-v**: 执行脚本的同时将执行过的脚本命令打印到标准错误输出（**变量替换前的命令**）
- **-x**: 提供跟踪执行信息，将执行的每一条命令和结果依次打印出来（**变量替换后的命令**）

使用shell脚本调试选项的方法

- 脚本运行时在命令行中设定调试选项

```
$sh -n script.sh
```

- 脚本开头提供调试选项

```
#!/bin/bash -v
```

- 脚本中用set命令启用或禁用调试选项

```
#!/bin/sh
```

```
if [ -z "$1" ]
```

```
then
```

```
    set -x
```

```
    echo "ERROR: Insufficient Args. "
```

```
    exit 1
```

```
    set +x
```

```
fi
```

The End