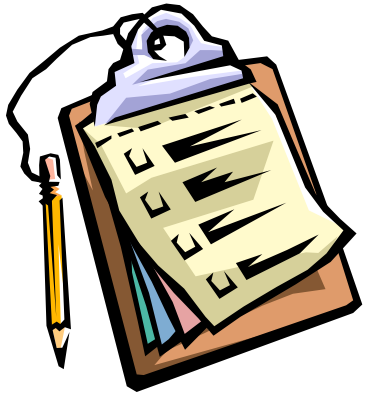
Linux Shell 基础

大连东软信息学院
大学生创业实训中心

目录

Neusoft



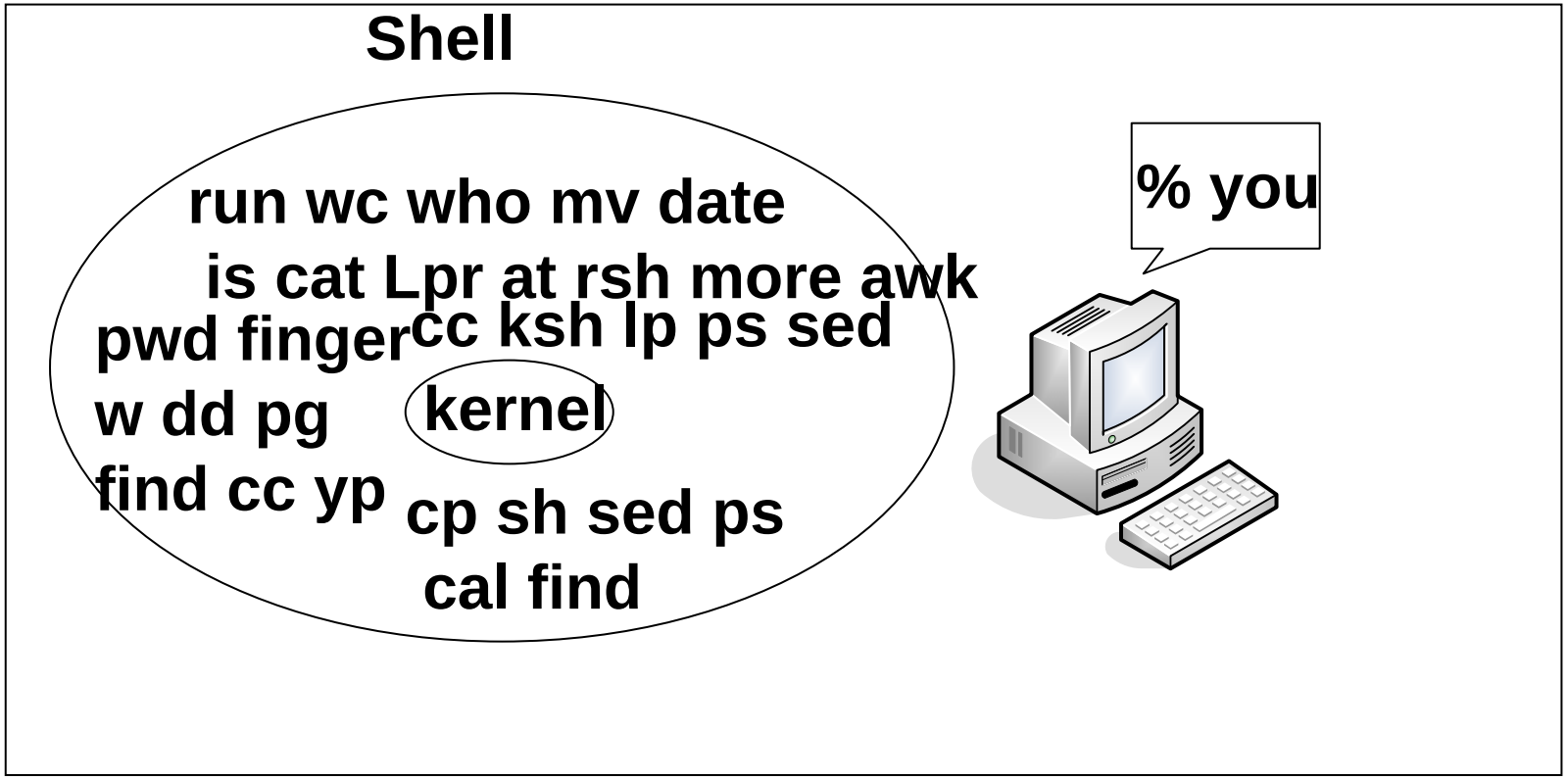
前言

- 1965年AT&T贝尔实验室 Multics的操作系统
- 1969年贝尔实验室 基本文件系统.
  UNIX分支版本（AT&T、BSD）
- 1991年 Linux Torvalds 类似于Unix的操作系统内核
- POSIX标准
- Shell定义的标准:IEEE1003.2 POSIX Shell

Shell的简介

Neusoft

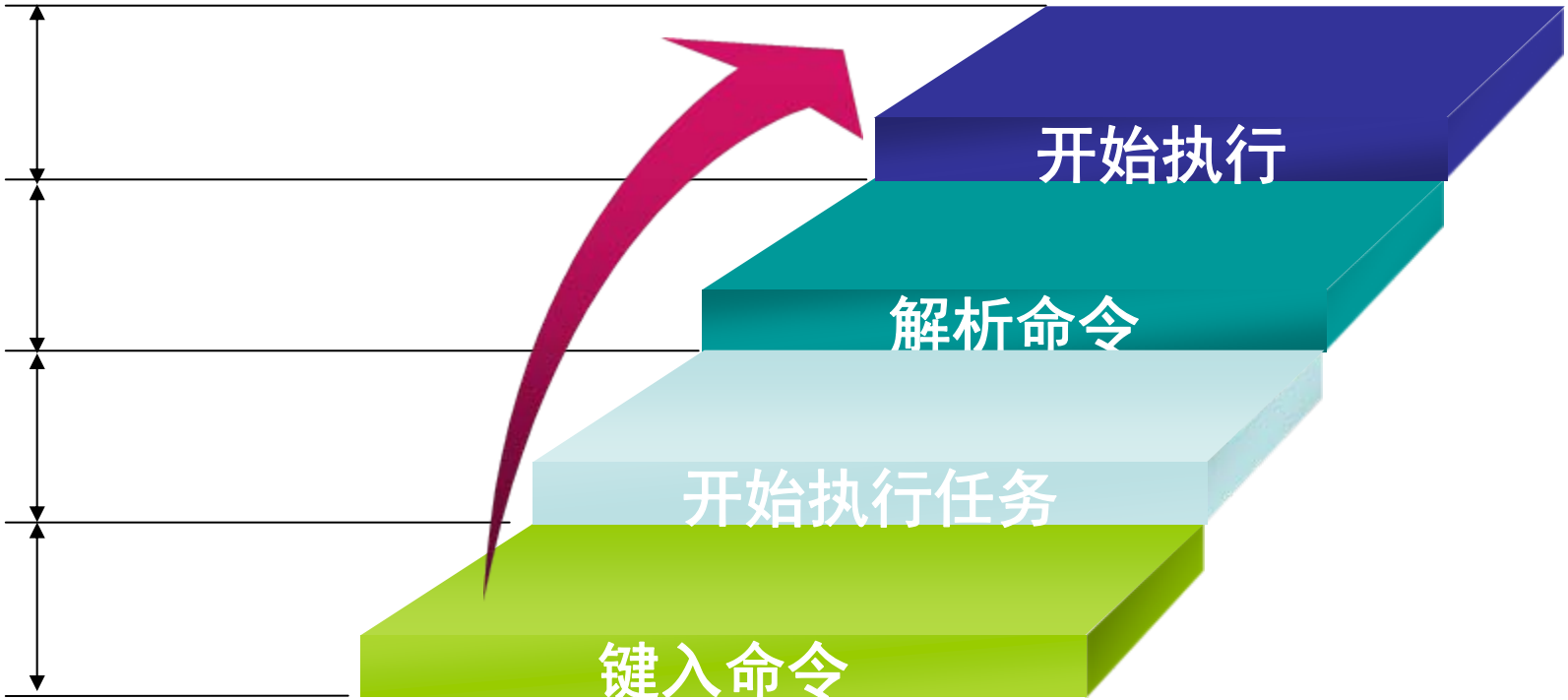
- Shell 是一种特殊的程序。如图



Shell的定义和功能

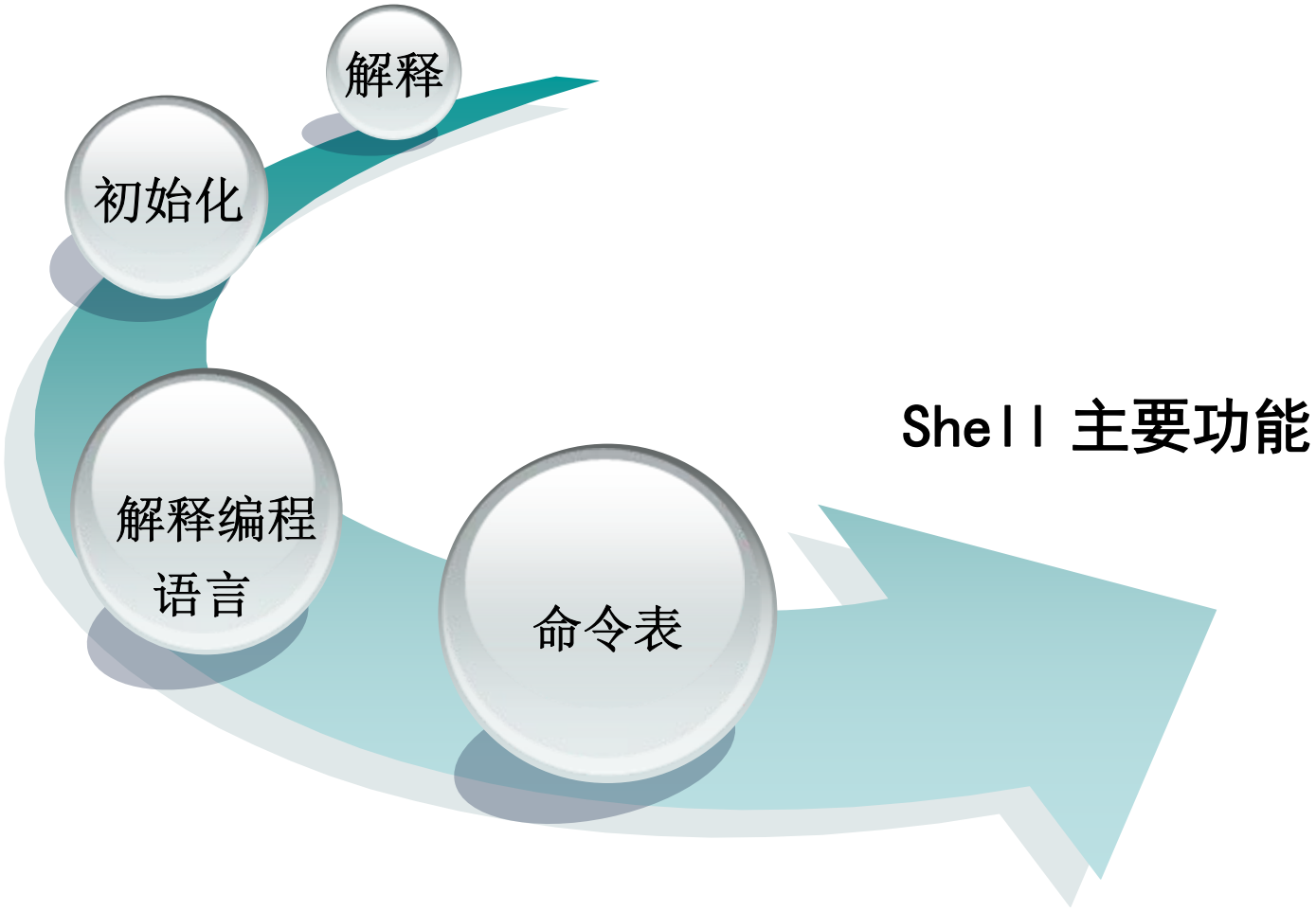
Neusoft

- 创建和控制进程，管理内存、文件系统和通信等。
- Shell是一个工具程序，在用户登录系统后。它解释并运行命令或脚本，从而实现用户与内核的交互。
- 当用户登录成功，系统会启动一个交互的Shell来提示用户输入指令。



Shell的功能

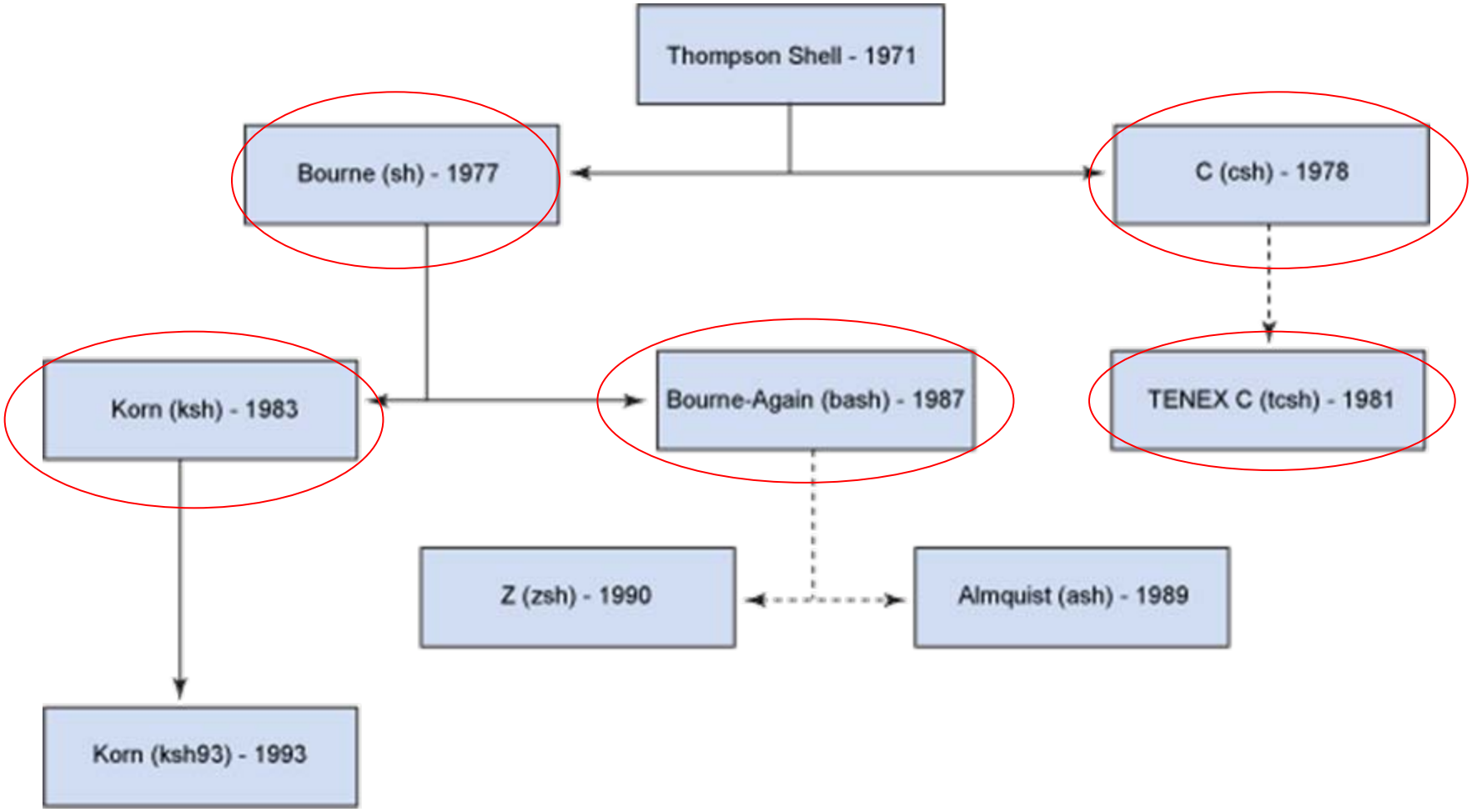
Neusoft



Shell的种类

Neusoft

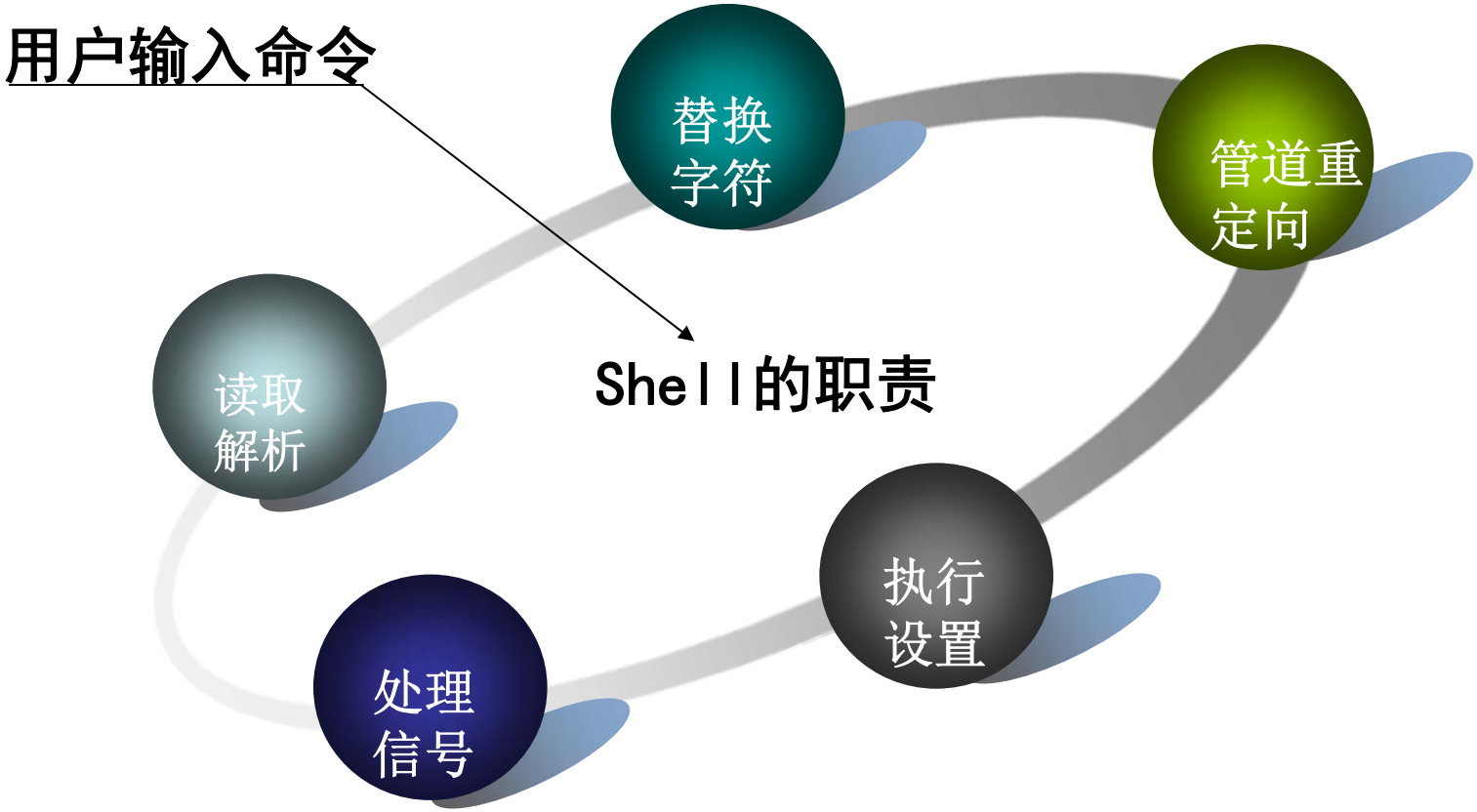
- UNIX系统支持主流的Shell;



- Bash 符号($) C Shell符号(%) KShell符号(%)

- **Shell的职责**

  Shell负责确保用户在输入的命令被正确执行。包括：



- **Linux Shell 介绍**

  用户可以访问到GNU的Shell和工具（非标准UNIX的Shell和工具）。

  BASH是当前UNIX/LINUX用户使用得最为普遍的Shell。

  默认提示符号为（$)。
- **LINUX用户常用的另一个Shell是TC Shell。TC Shell是UNIX C Shell的一个兼容分支，但是新增了许多附加功能，默认的提符是（>)。**
- **Z Shell也是LINUX一种Shell，它结合了Bourned Again Shell、TC Shell和Korn Shell的许多功能。**

- 查看自己所使用的Linux有哪些版本的Shell,可以查看/etc/Shell目录下的文件。

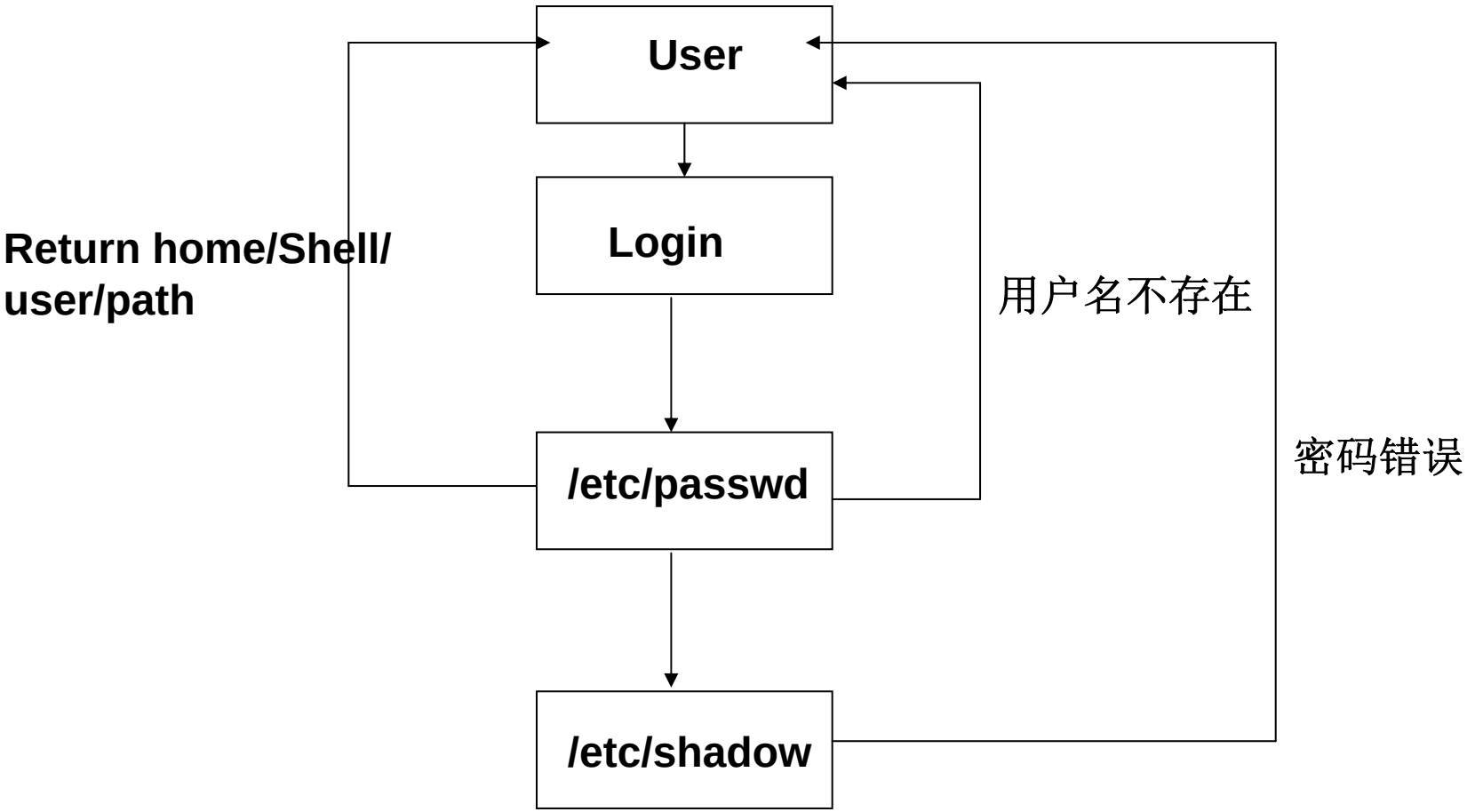
  例如:

```
more /etc/shells
```

  结果:

```
/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/ksh
/bin/zsh
```

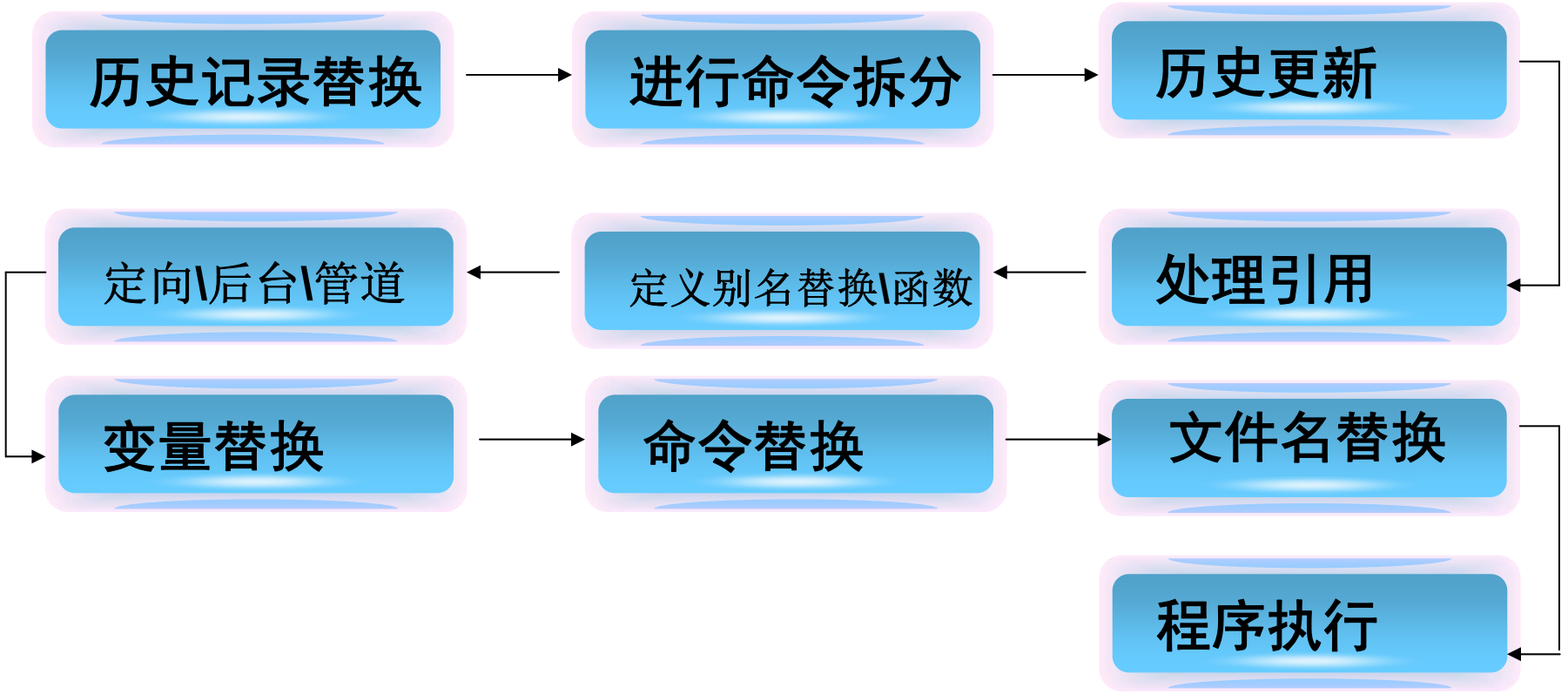
- 系统启动和登陆BASHShell



Linux Shell处理顺序

Neusoft

读取输入并解析命令行



Shell进程

Neusoft

什么是进程

进程是处在执行状态下，可以用唯一的PID标识程序.

什么是系统调用

Shell就以寻找内建命令或者外部可执行程 序来响应，接着安排命令运行. 这是依靠呼叫系统来完成的， 称为系统调用.

- 查看进程运行

  ps 查看进程 pstree 查看进程和子进程

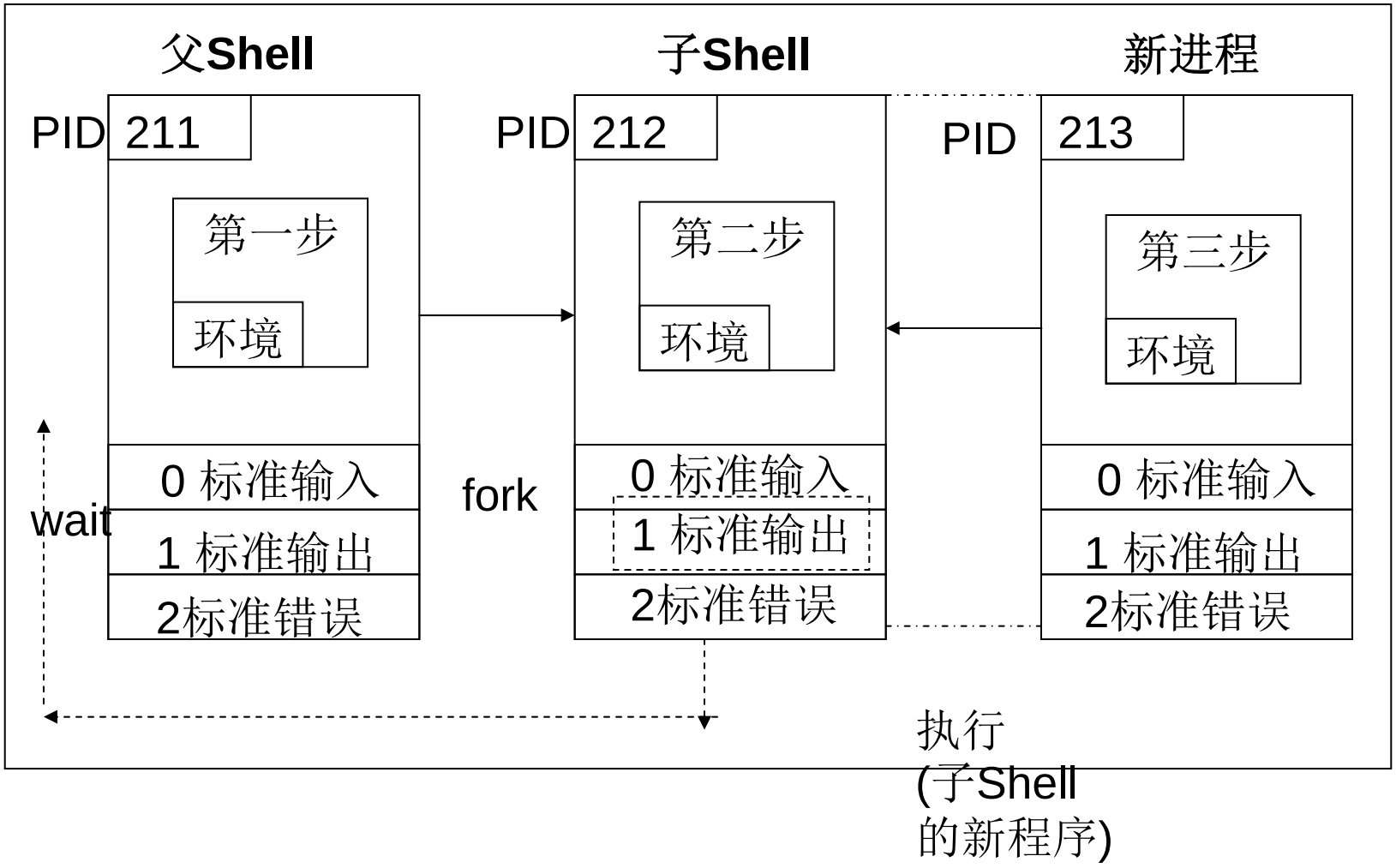
- 建立和终止进程的系统调用

  Fork:用于创建一个新进程.

  Wait:当子进程处理细节的时候如重新定向\管道和后台进程等父亲进程被设计为休眠状态(待机).

  Exec:当在终端输入一条命令,Shell通常会派生出一个新的进程(子进程),子进程的Shell负责执行你输入终端的命令.

  Exit:调用exit系统调用可以在任何时候终止程序的执行. 0表示子进程退出成功,非0状态表示发生某种失败.


父Shell
子Shell
新进程
PID 211
PID 212
PID 213
第一步
环境
第二步
环境
第三步
环境
0 标准输入
1 标准输出
2标准错误
0 标准输入
1 标准输出
2标准错误
0 标准输入
1 标准输出
2标准错误
fork
wait
执行
(子Shell
的新程序)

- 环境与继承

  当登录系统时,Shell启动并从启动它的/bin/login程序中继承了多个变量\I/O流和进程.

- 继承环境
- 所有权

  文件创建掩码 :当文件被建立的时候,它被赋予一套默认访问权限.通过umask进行修改.

  umask的值是000,权限访问777,默认的文件权限是666

  (rw-rw-rw).

  改变所有权和权限

  chmod 命令用于改变目录和文件的权限.

所有权和权限对照表

Neusoft

10进制数字	2进制数字	权限
0	000	none
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

注: r 表示读的权限,w表示写的权限,x表示执行的权限,u表示用户(所有者),g表标组,o表示其他人,a表示所有人

权限实例

Neusoft

- 权限实例:

```
$chmod 755 /home/usera/start.sh
$ls –l /home/usera/start.sh
 -rwxr-xr-x  1 root  root        27 Mar 27 14:07
$chmod g+w /home/usera/start.sh
$ls –l /home/usera/start.sh
  -rwxrwxr-x  1 root  root      27 Mar 27 14:07
$chmod go-rx /home/usera/start.sh
$ls –l /home/usera/start.sh
  -rwx-w----  1 root  root       27 Mar 27 14:07
$chmod a=r /home/usera/start.sh
$ls –l /home/usera/start.sh
   -r--r--r-- 1 root  root       27 Mar 27 14:07
```

文件所有组实例

Neusoft

- **chown -R 命令 改变文件和目录的所有者和组.**

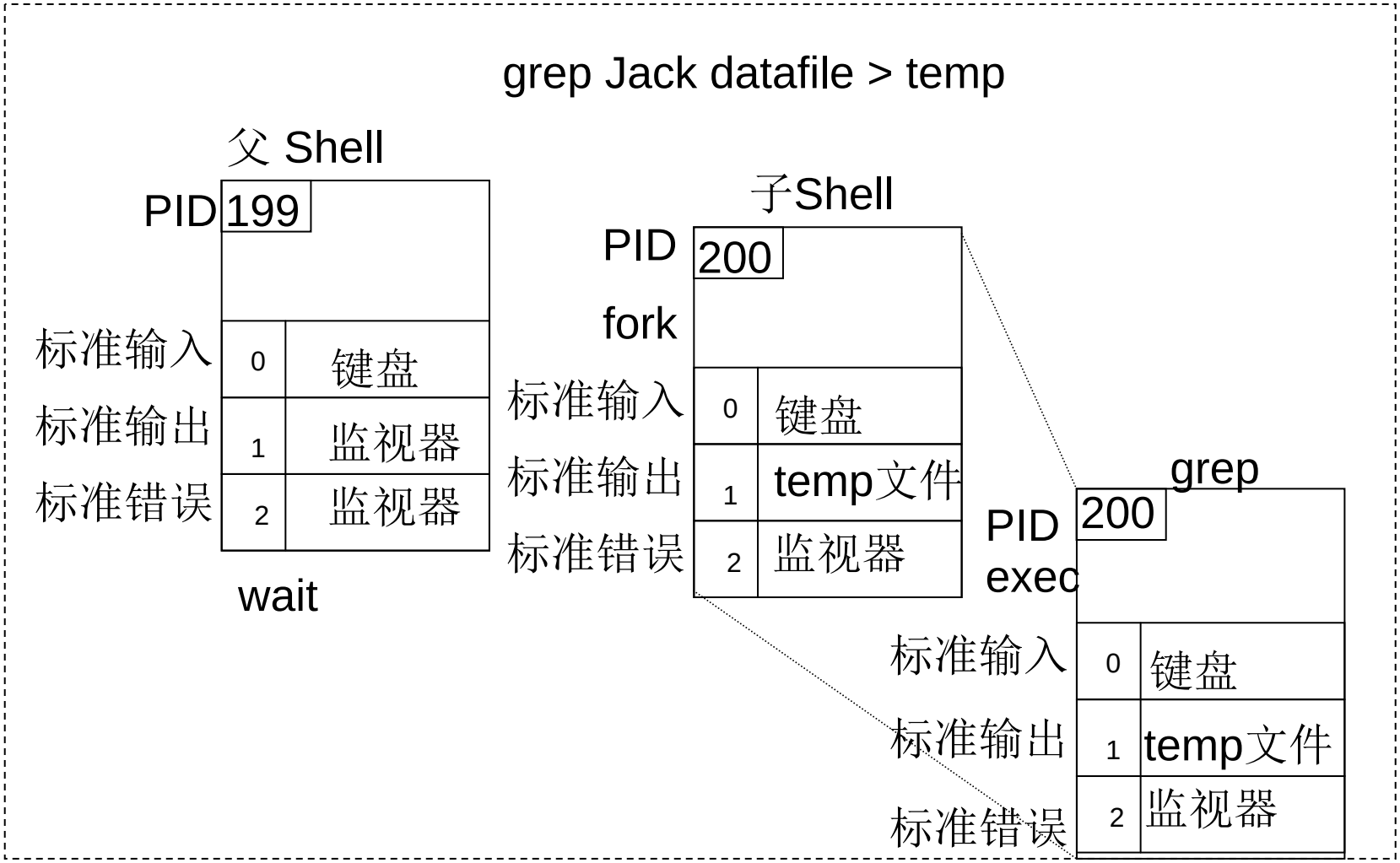
实例:

```
$ls –l filetest
$chown root filetest
  chown:filetest:Operation not permitted
$su root
   Password:
#ls –l filetest
   -rw-rw-r--  1 usera usera 0  Mar 10 12:19 filetest
#chown root filetest
#ls –l filetest
   -rw-rw-r--  1 root usera 0  Mar 10 12:19 filetest
#chown root:root filetest
   -rw-rw-r--  1 root root 0  Mar 10 12:19 filetest
```

- 工作目录
- 变量
  - 局部变量
  - 环境变量
- 文件说明符
- 重定向和管道

- **重新定向**

当文件说明符被分配给一非终端的时候,它就被称为I/O重新定向.



- 实例:

```
# who >usefile
# cat > usefile file
# cat usefile usefile2 >>usefile3
#find / -name root –print 2 > errors
```

- 管道

  管道为进程之间的通信服务,它是把一个命令的输出作为另外一个命令输入的机制.

  “ |”

  例如: who | wc

  如果没有管道,需要三个步骤

```
#who > tempfile
#wc tempfile
#rm tempfile
```

- **Shell与信号**

  当信号发送一个信息给进程时通常会导致进程终止.

  例如挂起\电源掉电或者是程序错误.

(事例)

- 从脚本执行命令

1 Shell脚本格式.每个命令一行.在第一行#!后面输入你希望使用的Shell和路径.

2 保存文件并打开执行权限

3 执行

例如:

```
vi doshell
#!/bin/bash
ls
who
……
```

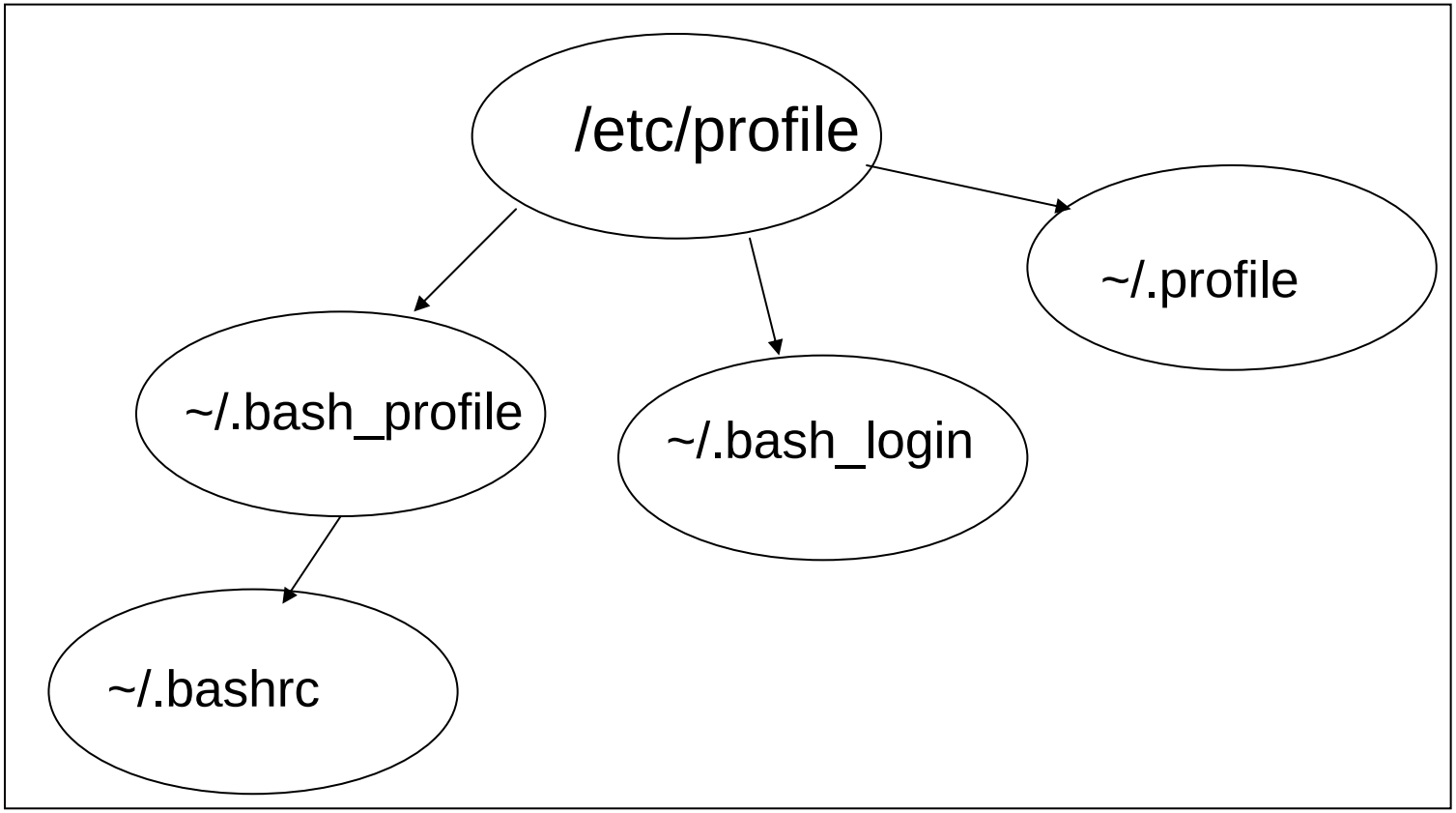
(录一个实际操作)

Neusoft

- 变量与运算符
  - 变量替换
  - 位置变量
  - 进程变量
  - Bash引号规则
  - 运算符
  - 表达式替换
  - 标准Shell变量
  - 影响命令的变量

- 变量：初始化文件
- 系统初始化文件 /etc/profile
- 用户初始化文件 用户主目录 bash_profile 设置用户别名和函数,并建立用户指定的环境变量和启动脚本.

- 文件初始化的顺序



变量赋值练习

vi variable.sh

```
#!/bin/bash
logfile=“monday.dat”
echo “The value of logfile is :”
```

执行后显示

```
The value of logfile is :
monday.dat.
echo "$logfile“
```

执行后显示为内容为空

利用source 命令可强行上一个脚本影响当前Shell的环境

例如上述例子 source variable.sh

- 父Shell环境变量的变动会产生对子Shell的变量影响

  例如:

```
export /etc/profile
```

profile文件内容:

```
PATH=$PATH:/usr/java/j2sdk1.4.2_14/bin:/usr/java/
  j2sdk1.4.2_14/jre/bin:/usr/local/apache/http/bin
```

使用 export PATH 影响当前Shell环境下的子Shell

- 变量替换:用变量的值替换它的名字

  利用花括号表示变量替换

  ${VARIABLE} 基本变量替换.花括号限定变量名的开始和结束

  ${VARIABLE: -DEFAULT}如果VARIABLE值为空, 返回DEFAULT

  ${VARIABLE: =DEFAULT}如果VARIABLE没有值,返回DEFAULT;

  另外,如果VARIABLE没有设置,则把DEFAULT的值赋予它.

  ${VARIABLE: + VALUE} 如果VARIABLE被设置, 返回VALUE;

  否则,返回一个空串

  ${#VARIABLE} 返回VARIABLE值的长度,除非VARIABLE是*或者@.在为*或者@的特殊情况下,则返回 $ @表示的元素的个数.

  ${VARIABLE:?MESSAGE} 如果VARIABLE值为空,则返回MESSAGE的值.Shell也显示出VARIABLE的名字, 对捕获错误很有用

  注: $ @保存传给脚本的参数清单, 字符用在$字符前面,告诉Shell忽略$字符的特殊含义

例:参加shell_variable替换 的事例;

SHELL源码

- 位置变量

  Shell变量使用位置变量来存取脚本参数.

  例: variable_2.sh :

```
#!/bin/bash
    echo "\$0 =*$0*"
    echo "\$1 =*$1*"
```

执行:

```
#variable_2.sh one .dat two.dat
#./variable_2.sh one
    $0 =*./variable_2.sh*
    $1 =*one*
```

当存取的参数超过第10个时,就要用花括号;

例如:

${14} ${19}

- 变量表示的用法

$ * 包括参数的列表

$@ 包括参数的列表

$# 包括参数的个数

例: listparam 脚本,它实不了上述三个变量的用法

```
#!/bin/bash
echo "There are $# parameters."
echo "The parameters are *${*}*"
echo "The parameters are *$@*“
#listparam.sh one two three
There are 3 parameters.
The parameters are *one two three*
The parameters are *one two three*
```

$@ 的作用　listfiles

```
#!/bin/bash
for file in $@
do
ls -l $file
done
```

执行结果

```
# ./listfiles listfiles listparam.sh
-rwxr-xr-x  1 root root 49 May 18 09:03 listfiles
-rwxr-xr-x  1 root root 109 May 15 17:10 listparam.sh
```

- 进程变量

  四个进程变量:

  $? 如果最后执行成功,该变量的值为0.任何他值都表示失败.

  $$ 变量保留当前进程的id号。

  $- 变量保留可以打开的Shell选项清单.

  $! 变量保留最后在后台执行的进程的id号.

例:查看当前命令执行的状态

```
# ls listfiles
      listfiles
# echo $?
     0
# ls filesss
     ls: filesss: No such file or directory
# echo $?
      1
```

- $$ 变量保留当前进程的id号; 例: 创建唯一的文件名

```
#echo `date` > /tmp/$$.out
#cat /tmp/$$.out
   Mon May 18 09:34:18 CST 2009
# ls -al /tmp/$$.out
-rw-r--r--  1 root root 29 May 18 09:34 /tmp/10768.out
```

访问/proc目录结构,了解有关当前进程的信息

```
#cat /proc/$$/status
   Name:   bash
   State:  S (sleeping)
   SleepAVG:       98%
   Tgid:   10768
   Pid:    10768
   PPid:   10766
```

- Bash引号规则

  引号规则:

- 双引号: 括起文件,阻止Shell对大多数特殊字符进行解释.

  例如:在双引号中#字符并不是注释的开头,它只表示一个符号#.然后,$,`和”字符仍保持它们的特殊含义.

- 单引号: 括起文件,阻止对所有字符(除单引号外)进行解释.
- 倒引号: 括起文本,表明命令替换.在倒引号内部的Shell命令被执行,其结果输出代替用倒引号括起来的文本.

- 双引号命令使用规则例:

```
# echo "\"Whell said,\"said she ."
  "Whell said,"said she .
# export logfile=monday.dat
# echo "The value of  \$logfile is $logfile.“
   The value of $logfile is monday.dat.
# echo "Yesterday was `date '+%B%d' --date '1 day ago'`."
  Yesterday was May17.
```

- 单引号使用规则例:

```
# echo '\"Well said, \"said she .'
   \"Well said, \"said she .
# echo '"Well said," said she.'
  "Well said," said she.
# export logfile=monday.dat
# echo 'The value of \$logfile is $logfile.'
  The value of \$logfile is $logfile.
# echo 'Yesterday was `date'+%B%d' --date '1 day ago'`.'
  Yesterday was `date+%B%d --date 1 day ago`.
```

- 运算符 注:以降级的顺序排列

级别	运算符	说明
13	-,+	单目负\单目正
12	!,~	逻辑非\按位取反或补码
11	*,/,%	乘\除\取模
10	+,-	加\减
9	<<,>>	按位左移\按位右移
8	<=,>=,<,>	小于或等于\大于或等\小于\大于
7	= =,!=	等于\不等于
6	&	按位与
5	^	按位异或
4	|	按位异或
3	&&	逻辑与
2	||	逻辑或
1	=,+=,-=,*=,/=,%=,&=,^=,|=<<=,>>=	赋值\运算且赋值

注: Shell不支持幂运算符.对此以及更高级的运算符需要使用Perl或者Tcl.

- 运算符优先级的顺序是如何影响表达式求值过程的:

```
# echo $[1+2*4]
    9
# echo $[(1+2)*4]
   12
```

- 取模运算符

例: 输入一组数求出除3,余数为0的值;

```
get_modulus
    #!/bin/bash
    count=1
    for element in $@
   do
   if test $[$count%3] = 0
   then
   echo $element
   fi
   let count=$count+1
   done
```

```
# ./get_modulus 1 2 3 4 5 6
     3
     6
```

- 按位运算符

运算符	说明
~op1	取反运算符op1中所有二进制位的1变为0, 0变为1
op1<<op2	左移运算符把op1中的二进制位向左移动op2位,忽略最左端移出的名位,最右端的各位补上0值,每做一次按位就有效地实现op1乘以2
op2>>op2	右移运算符op1中的二进制位向右移动op2位,忽略最右端移出的各位,最左端的各位补上0值.每做一次按位右移就有效地实现op1除以2
op1&op2	与运算符比较op1和op2的对应位,对于每个二进制位来说,如果二者该位都是1,则结果为1;否则结果位为0

- op1^op2 异或运算符比较op1和op2的对应位,对于每个二进制位来说,如果二者该位互补,则结果位为1;否则,结果位为0
- op1|op2 或运算符比较op1和op2的对应位,对于每个二进制位来说,如果二者该位有一个是1或者都是1,则结果位为1;否则,结果位为0

- 逻辑运算符

&&结果

运算对象1	运算对象2	运算对象1&&运算对象2
0	0	0
1	0	0
0	1	0
1	1	1

||结果

运算对象1	运算对象2	运算对象1&&运算对象2
0	0	0
1	0	1
0	1	1
1	1	1

- 赋值运算符
  简单地把等号右边的表达式的值赋予等号左边的变量
  例:　不必使用let $count=$count+$change
  　　直接使用let $count+=$change
- 表达式替换
  使用$[ ]
  例:

```
#echo $ [12/0]
   -bash: 12/0: division by 0 (error token is "0")
#echo $ # echo $[7#10+1]
   8
# echo $[10#8+1]
   9
```

- 标准Shell变量

BASH:保存用来引用运行当前Shell的Bash实例的全路径名
BASH_VERSION:保存当前Bash实例的版本号。
cdable_vars: 确定传给cd命令的实参是否是一个变量，其值是要切换到的目录。设置为“true”,就强加此特性

CDPATH:保存cd命令的搜索路径
command_oriented_history: 确定Shell是否要把多行命令的所有行都保存在相同的历史项中。设置为“true”,就强加此特性

ENV: 保存用来初启新Shell的文件名

EUID:保存当前用户 的有效用户ID

FCEDIT: 保存fc命令的默认编辑器的名字FIGNORE当实现文件名完整化时，保存要忽略的以冒号分开的后缀清单

Histchars:控制历史扩展和表示法。

HISTCMD:保存当前命令的历史号码

HISTCONTROL:确定命令是否放入历史表中和是否忽略重复的命令。ignoresspace,ignoredups和ignoreboth

HISTFILE:保存存命令历史的文件名。其默认值是~/.bash_history

HISTFILESIZE:保存历史文件中所容纳的最大行数。其默认值500

HISTSIZE:保存记忆的命令数

HOME:保存当前用户的主目录

HOSTFILE:保存当前Shell使主机名完整化,所读取的文件名

HOSTTYPE唯一地描述正行Shell的机器类型

IFS: 保存Bash扩展和词不达意分解内部例程所用的字段区分符的字符列表。默认区分符是空格、制表符和换行符
IGNOREEOF控制Shell对EOF（Ctrl-D）字符如何反应。当Shell在其输入中遇到Ctrl-D或者0x04字符时，它通常终止。

INPUTRC:保存readline初启文件的文件名

LINENO:保存当前脚本或函数的当前序列行号。在脚本或函数的上下文以外其值没意义

PATH=“.:$PATH”

当前目录中的脚本就先被找到,而在路径的后面目录中的脚本就随后被找到

- 影响命令的变量

  declare [options][name [=value]]

  用于显示或设置变量

  -f 只显示函数名

  -r 创建只读变量.只读变量不能被赋予新值或取消设置.除非重起Shell

  -x 创建转出(exported)变量

  -i 创建整数变量.如果想给整数变量赋予文件本值,默认为0

例

```
# declare -r title="paradise Lost"
# title="Xenogenesis"
   -bash: title: readonly variable
```

Export 命令

Export [options][name [=value]]

Export 命令使用四个选项:

-- 表明选项结束.所有后续参数都是实参

-f 表明在”名-值” 中的名字是函数名

-n 把全局变量转换成局部变量.

-p 显示全局变量列表

例:

```
# export declare –x
CLASSPATH="./:/usr/java/j2sdk1.4.2_14/lib:/usr/java/j2sdk1.4.2_14/jre/lib"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
```

- let 命令

  let expression

  用于求整数表达式的值;通常用来增加计数器变量的值.

  例: letcount

```
#!/bin/bash
    count=1
    for element in $@
    do
    echo "$element is element $count"
    let count+=1
    done
```

- 执行

```
./letcount onw two three four five
  one is element 1
  two is element 2
  three is element 3
  four is element 4
  five is element 5
```

- **Local 命令**

  local [name value]]

  用于创建不能传给子Shell存取.因此,只能在函数内部使用local命令.

  例如: “变量=值”

- **Readonly命令**

  readonly [options] [name [=value]]

  用于显示或者设置只读变量

  readonly命令使用两个选项:

  -- 表明选项结束.所有后续参数都是实参

  -f 创建军只读函数

- **Set 命令**

  set [--abefhkmnptuvxldCHP][-o option][name [=value]]

  用于设置或者重置各钟Shell选项.

- **Shift命令**

  shilft [n]

  用于移动位置变量;

  例:$3的值赋予$2,而$2的值赋予$1.

- **Typeset命令**

  typeset [options] [name [=value]]

  用于显示或者设置变量

  功能近似于declare

- **Unset 命令**

  unset [options] name [name …]

  用于取消变量定义

  unset命令使用两个选项:

  -- 表明选项结束,所有后续参数都是实参

  -f 创建只读函数

- 练习 1
  对变量进行赋值练习;
- 练习2
  编写一个简单的Shell程序;

答疑时间

Neusoft

Q & A