第三章 shell 编程

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第一节、 shell 脚本

- 本节重点内容：
- 一、 shell 环境配置
  - 配置 shell 启动文件和环境变量
- 二、 shell 脚本结构
  - 幻行、注释、变量、分支、循环等
- 三、使用 shell 脚本
  - 编辑脚本、修改权限、执行脚本

一、 shell 环境配置

- 1、启动文件
- UNIX/Linux 系统提供启动文件用于 shell 的初始设置。
- 主要的启动文件有：
  - /etc/profile 登录时自动运行，对所有 shell
  - ~/.profile 登录时自动运行，对用户
  - ~/.bash_profile 登录时运行，对某一 shell
  - ~/.bash_login 登录时运行，对某一 shell
  - ~/.bash_logout 退出时自动执行

一、 shell 环境配置

- 2 、环境变量
- 常用环境变量有：
- (1)IFS
- IFS 用作 shell 指定的缺省域分隔符。原理上讲域分隔符可以是任意字符，但缺省通常为空格、新行或 tab 键。 IFS 在分隔文件或变量中各域时很有用。
- 其取值可能为：

IFS=$' \t\n'　IFS 的可能取值

一、shell 环境配置

- (2)PS1
- 基本提示符，包含 shell 提示符，缺省对超级用户为 # ，其他为 $ 。其取值可能为：

```
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
```

- 可以使用任何符号作提示符，如下例中将提示符改为“ welcome: ”

```
root@dai-laptop:/home/dai# export PS1="welcome: "
welcome:
```

一、 shell 环境配置

- (3)PS2
- 附属提示符，缺省为符号 > 。 PS2 用于执行多行命令或超过一行的一个命令。
- (4)PWD
- 保存当前工作目录。可以在 shell 脚本中使用此环境变量取得当前工作目录。

一、 shell 环境配置

- (5)PATH
- 保存 shell 查找命令或程序的目录列表。各目录用分号分割。

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin
```

二、 shell 脚本

幻行

注释

变量

分支

```
#!/bin/sh
#mytar
#this is my tar command

USAGE="Usage: `basename $0` [-c|-t][files|directories]"

if [ $# -lt 2 ] ;then
    echo "$USAGE";
    exit 1;
fi

.....
```

三、使用 shell 脚本

- 1、编辑脚本文件
  - 任意文本编辑器，常用 vi 、 emacs 等。
- 2、使脚本文件具有可执行权限
  - chmod a+x filename.sh
- 3、执行脚本
  - 指定脚本路径，当脚本在当前目录下，可用：./filename.sh 执行脚本
  - 或，修改 PATH 环境变量值：

    $PATH=$PATH:dir;**export** PATH

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第二节、变量

- shell 中的变量分为：
  - 本地变量
  - 环境变量
  - 位置变量
  - 特定变量
- 其中本地变量只用于当前 shell，环境变量对所有子进程起作用，位置变量和特定变量是 shell 定义的只读变量。

一、本地变量

- 1 、标量变量
- 标量变量一次只能被赋予一个值，按照如下格式定义:
- name=value
- 这里 name 是变量名， value 是变量被赋予的值。
- (1) 变量名
- 变量的名称中只能包含**字母、数字**和**下划线**。此外，变量名只能以**字母或下划线**作为开始。

一、本地变量

- 下例哪些是合法的变量名?
  - hello world　X 不能包含空格
  - _1st_name　√
  - 1st　X 数字不能开头
  - OK!　X 不能包含！
- 下面对变量的赋值是否正确?
  - FRUIT=apple orange

X 空格被当作分隔符，正确的写法是:
FRUIT=”apple orange”

一、本地变量

- (2) 访问变量
- 要获取变量的值，只需要在变量名之前加上 $ 符号，当 shell 发现了 $ ，就会执行如下操作：
  - 读取下一个单词来确定变量名
  - 检索变量的值。如果变量没有赋值，则 shell 自动使用空字符串“ ”作为其值。
  - 用变量的值取代 $ 和变量名。
- 这个过程被称为“变量置换”

一、本地变量

- 2、数组变量
- 数组变量是使用一个名称配置一组变量的方法。
- (1) 建立数组变量
- 最简单的建立数组变量的方法为：
- name[index]=value
- 这里 name 是数组名， index 是在数组中的索引， value 是要赋给的值。如：

```
fruit[1]=apple;fruit[2]=banana
```

一、本地变量

注意：

- 数组索引从 0 开始。
- 数组索引不必按顺序排列。
- 如果数组变量名与一个已经被定义的标量变量重名，则这个标量变量就自动成为数组中的 0 号元素。

一、本地变量

- 第二种数组初始化的方法
  - 在 ksh 中，语法为：

    set –A name value1 value2 ……valueN
  - 在 bash 中，语法为：

    name=(value1 …… valueN)

```
fruit=(apple banana)
```

一、本地变量

- (2) 访问数组变量
- 一个数组变量可以通过如下方式访问：
- ${name[index]}
- 这里 name 是数组名， index 是索引。
- 使用下面的方法可以访问数组中所有的项目：
- $ {name[*]}
- $ {name[@]}

一、本地变量

- 3、删除变量
- 删除变量就是告诉 shell 从它跟踪的变量列表中移除给变量。标量变量和数组都可以通过 unset 命令被删除：
- unset name

```
unset fruit
```

一、本地变量

- 4 、显示所有本地变量
- 可以使用 set 命令显示所有在本地定义的 shell 变量。 set 输出可能很长。查看输出时可以看出 shell 已经设置了一些用户变量以使工作环境更加容易使用。

一、本地变量

- 5、只读变量
- 只读变量的值在被定义了之后就不能被更改，这个变量和其值将一直保持到 shell 退出为止，而且不能被删除。
- 只读变量可以通过 readonly 标记，例如：

```
$fruit=apple
$readonly fruit
```

- 试图修改只读变量将导致出错

二、环境变量

- 环境变量对所有子进程（用户进程）起作用，通常为大写，用于配置 shell 环境。
  - 环境变量的查看： set
  - 环境变量的访问： $
  - 环境变量的赋值： =
  - 环境变量的导出： export

  环境变量对所有子进程起作用前，必须导出！

三、位置变量

- 位置变量是 shell 中的只读变量，用 0 到 9 十个数字表示，依次代表脚本调用时的各参数，用于向脚本中传递信息。同本地变量一样，位置变量用 $0 到 $9 访问。下面的例子演示了位置变量的用法。

三、位置变量

```
$ pg param
#!/bin/sh
# param
echo "This is the script name          : $0"
echo "This is the first parameter      : $1"
echo "This is the second parameter     : $2"
echo "This is the third parameter      : $3"
echo "This is the fourth parameter     : $4"
echo "This is the fifth parameter      : $5"
echo "This is the sixth parameter      : $6"
echo "This is the seventh parameter    : $7"
echo "This is the eighth parameter     : $8"
echo "This is the ninth parameter      : $9"
```

三、位置变量

```
$ param Did You See The Full Moon
This is the script name          : ./param
This is the first parameter      : Did
This is the second parameter     : You
This is the third parameter      : See
This is the fourth parameter     : The
This is the fifth parameter      : Full
This is the sixth parameter      : Moon
This is the seventh parameter    :
This is the eighth parameter     :
This is the ninth parameter      :
```

三、位置变量

- 下面脚本演示了另一个例子

```
$  pg findfile
#!/bin/sh
# findfile
find / -name $1 -print

$ findfile passwd
/etc/passwd
/etc/uucp/passwd
/usr/bin/passwd
```

四、特定变量

- 共有 7 个特定变量，用来制定脚本运行时的相关控制信息。

$#	传递到脚本的参数个数
$*	以一个单字符串显示所有向脚本传递的参数。 与位置变量不同，此选项参数可超过 9 个
$$	脚本运行的当前进程 ID 号
$!	后台运行的最后一个进程的进程 ID 号
$@	与$ #相同，但是使用时加引号，并在引号中返回每个参数
$-	显示 shell 使用的当前选项，与 set 命令功能相同
$?	显示最后命令的退出状态。表示没有错误，其他任何值表明有错误。

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第三节、置换

- shell 中除了字母和数字外的字符大多具有特殊含义，把这些**具有特殊含义的字符称为元字符**。
- **置换**是指 shell 将包含元字符的字符串根据其意义转换成新字符串的过程。
- 置换包括：
- 一、文件名置换
- 二、变量置换
- 三、命令置换
- 四、算式置换

一、文件名置换

- 文件名置换就是 shell 将包含有通配元字符（通配符）的字符串转换成一系列的文件名，也称通配。
- 常用的通配符有：*、?和[]，含义如下表。

*	0个或多个任意字符
?	一个任意字符
[]	匹配一个字符集合

对[]中的字符集合，有3中表示方式：
1、全部列出
2、使用连字符 -
3、用！取消一个集合

一、文件名置换

- 举例：
  - 匹配前缀： pre*
  - 匹配后缀： *suf
  - 匹配前缀和后缀： pre*suf
  - 匹配 ch＊＊.doc 形式的文件： ch??.doc
  - 匹配所有小写字母： [a-z]
  - 匹配所有数字： [0-9]
  - 匹配所有不以 a 开头的文件： [!a]*

二、变量置换

- 1、缺省值置换
- 当一个变量的值为空时，被置换为一个缺省值。语法为：
- $ {variable:-value}
- 当 variable 无值时，用 value 替代表达式。

```
root@dai:~/course$ MYFRUIT=banana
root@dai:~/course$ unset MYFRUIT
root@dai:~/course$ FRUIT=${MYFRUIT:-apple}
root@dai:~/course$ echo $MYFRUIT,$FRUIT
 ,apple
root@dai:~/course$
```

二、变量置换

- 2、缺省值赋值
- 当一个变量的值为空时，把缺省值赋给这个变量。语法为：
- $ {variable:=value}
- 当 variable 无值时，用 value 给 variable 赋值。

```
root@dai:~/course$ MYFRUIT=banana
root@dai:~/course$ unset MYFRUIT
root@dai:~/course$ FRUIT=${MYFRUIT:=apple}
root@dai:~/course$ echo $MYFRUIT,$FRUIT
apple,apple
root@dai:~/course$
```

二、变量置换

- 3、空值错误
- 当一个变量未赋值时，输出一个错误信息到标准输出。语法为:
- $ {variable:?msg}
- 当 variable 无值时，打印 msg 到标准输出。

```
root@dai:~/course$ unset MYFRUIT
root@dai:~/course$ FRUIT=${MYFRUIT:-apple}
root@dai:~/course$ echo ${MYFRUIT:?"no variable"}
-bash: MYFRUIT: no variable
root@dai:~/course$
```

二、变量置换

- 4、有值置换
- 当一个变量已经被赋值时，置换一个值。语法为： ${variable:+value}
- 当 variable 被赋值时， value 被置换给它，否则，什么置换也不会发生，返回一个空字符串。

```
root@dai:~/course$unset FRUIT
root@dai:~/course$FRUIT=banana
root@dai:~/course$echo ${FRUIT:+"FRUIT is active"}
FRUIT is active
root@dai:~/course$
```

三、命令置换

- 命令置换是一种机制，能使你获得命令的输出结果。在此机制下，shell 执行命令集合，然后将命令的输出结果置换，可以使用` 和` 进行命令置换，形式如：
- $ `command`
- 其中的 command 可以是一个简单命令、一个管道、或者一个命令列表。

注意：这里使用的是倒引号不是单引号！

```
root@dai:~/course$USER=`whoami`
root@dai:~/course$echo $USER
root
root@dai:~/course$
```

四、算式置换

- 算式置换能使 shell 完成简单的整数运算。
- 算式置换的形式为
- $ ((exp))
- exp 是一个算术表达式，使用算术运算符，+、-、*、/、()。
- 如果计算结果不是一个整数，会截去小数部分。

```
root@dai:~/course$echo  $(( ((5+3*2)-4)/2))
3
root@dai:~/course$
```

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第四节、引用

- 置换是将元字符进行解析的过程。
- **引用**是取消对某些元字符的置换的处理。
- 引用的方法有：
  - 一、反斜线
  - 二、单引号
  - 三、双引号

第四节、引用

- 1、使用反斜线实现引用
- 反斜线使其后的元字符被引用，即失去其特殊含义。如下面的例子：

```
root@dai-laptop:/# echo hello world
hello world
root@dai-laptop:/# echo hello;world
hello
-bash: world: command not found
root@dai-laptop:/# echo hello\;world
hello;world
root@dai-laptop:/#
```

第四节、引用

- 2、使用单引号实现引用
- 单引号‘和’之间的字符都将失去其特殊含义。
- 但是包含的字符串中包含单引号时，不能使用单引号进行引用。

```
root@dai-laptop:/# echo <-$1250.**>;(update?) [y|n]
-bash: syntax error near unexpected token `;'
root@dai-laptop:/# echo '<-$1250.**>;(update?) [y|n]'
<-$1250.**>;(update?) [y|n]
root@dai-laptop:/#
```

第四节、引用

- 3、使用双引号实现引用
- 双引号禁止除了$、`和\以外的所有元字符。
- 如完成如下要求的输出：
- 用变量保存的某用户名 owes <-$1250.**> [as of (用日/月的形式表示的当前日期)]
- 应该使用如下语句：

```
root@dai-laptop:/# echo “$USER owes <-\$1250.**>
[as of (`date +%m/%d`)”
ranga owes <-$1250.**> [as of (09/25)]
root@dai-laptop:/#
```

第四节、引用

- 4、引用的使用场合
- (1) 引用处理忽略单词界限
- $ echo “hello;world” 与
- $ echo hell”o;w”orld 有相同的输出
- Hello;world
- 对整组单词进行引用处理会更易于理解和阅读，但不对整组单词进行引用处理在一些复杂的引用场合会很有用。

第四节、引用

- (2) 在命令中的引用处理组合
- 可以在同一命令行中自由地选择某种引用处理形式。如：
- $echo The ‘$USER’ variable contains this value \> “|$USER|”
- The $USER variable contains this value > |ranga|

第四节、引用

- (3) 在单一参数中内嵌多个空格
- 对于 shell，在命令行参数中的一个或多个空格或 tab 只能算一个。
- 可以通过引用处理在结果中得到多个空格。

```
$ echo Name      Address
Name Address
$ echo “Name      Address”
Name      Address
```

第四节、引用

- (4) 引用处理换行以在下一行中继续
- 可以对换行字符进行引用处理以使一个很长的命令延伸到下一行：
- $ cp file1 file2 fiel3 file4 file5 file6 file7 \
- >file8 file9 /tmp

第四节、引用

- (5) 引用处理以访问包含特殊字符的文件名
- 如，要删除含有 * 字符的文件 ch1* ，可以用：

```
$ rm ch1\*
$ rm ‘ch1*’
$ rm “ch1*”
```

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第五节、流控制

- 本节主要内容：
- 一、 if
- 二、 case
- 三、 while
- 四、 until
- 五、 for
- 六、 select
- 七、 break 和 coutinue

一、 if

- if 语句根据给出的条件是否为真来执行相应的动作。代码返回 0 表示真，非 0 表示假。其语法为：

```
if list1
then
    command1
elif list2
then
    command2
else
    command3
fi
```

一、 if

- 对于短的 if 语句，也可以单独写成一行：
- If list1;then command1;elif list2;then commadn2;else command3;fi;
- f 语句的执行过程为：
  - 执行 list1
  - 如果 list1 退出码为 0，执行 command1，然后 if 语句终止；否则执行 list2 。
  - 如果 list2 退出码为 0，则执行 command2，然后 if 语句终止，否则执行 command3，然后 if 语句终止。

一、 if

- 通常指定给 if 语句的列表 (list) 是一条或多条 test 语句。其语法为：

  test expr

- 也可以用 [和] 代替 test ，写为： [expr] 。

注意： [和] 里， expr 前后有空格。

- test 能理解三中表达式：文件测试；字符串比较；数字比较。

一、 if

- 文件测试
- 测试一个文件是否满足特殊条件，语法为：

  test option file 或 [optionn file]

- 其中 file 是要测试的文件， option 是选项，常用选项见下表。

一、 if

选项	功能
-b file	file 存在且是块文件时返回真
-c file	file 存在且是字符文件时返回真
-d pathname	pathname 存在且是目录时返回真
-e pathname	pathname 指定的文件或目录存在时返回真
-f file	file 存在且是正规文件时返回真
-h file	file 存在且是连接文件时返回真
-p file	file 存在且是管道文件时返回真
-r pathname	pathname 指定的文件或目录存在且可读时返回真
-w pathname	pathname 指定的文件或目录存在且可写时返回真
-x pathname	pathname 指定的文件或目录存在且可执行时返回真

一、 if

- 字符串比较

选项	功能
-z str	当 str 长度为 0 时返回真
-n str	当 str 长度非 0 时返回真
str1=str2	当 str1 等于 str2 时返回真
str1!=str2	当 str1 不等于 str2 时返回真

一、 if

- 数字比较

选项	功能
int1 –eq int2	int1 等于 int2 时返回真
int1 –ne int2	int1 不等于 int2 时返回真
int1 –lt int2	int1 小于 int2 时返回真
int1 –le int2	int1 小于等于 int2 时返回真
int1 –gt int2	int1 大于 int2 时返回真
int1 –ge int2	int1 大于等于 int2 时返回真

一、 if

- 复合表达式
- 可以使用 test 和 [的内建操作符或者 && 和 || 来创建复合表达式。

操作符	功能
! expr	当 expr 为假时返回真
expr1 –a expr2	当 expr1 和 expr2 都为真时返回真
expr1 –o expr2	当 expr1 或 expr2 为真时返回真

二、case

- 语法为：

```
case word in
    pattern1)
          list1
          ;;
    pattern2)
          list2
          ;;
……
    patternN)
          listN
          ;;
esac
```

二、case

- 更简明的形式为：

```
case word in
    pattern1) list1 ;;
……
    patternN) listN ;;
esac
```

实例：处理选项时使用 case 判断位置变量的值
mytar.sh

三、 while

- while 循环在满足某种条件时重复执行一系列命令，直到条件不满足。其语法为：

```
while list
do
    cmd
done
```

- 也可以写成简单形式：

```
While list;do cmd;done
```

三、 while

- while 循环的执行顺序为：
  - 执行 list
  - 如果 list 的退出码非 0 ， 则退出循环；否则执行 cmd
  - 回到第一步，执行 list
  -
  - 实例：计算 5!

四、until

- until 循环重复执行一系列命令，直到条件为真。其语法为：

```
until  list
do
   cmd
done
```

- until 循环的执行顺序为：
  - 执行 list
  - 如果 list 的退出码为 0，则退出循环；否则执行 cmd
  - 回到第一步，执行 list

五、 for

- for 循环的语法为：

```
for name in word1 word2 ..... wordN
do
    list
done
```

- 这里 name 是变量名， word1 到 wordN 是由空格分开的字符序列。每次执行 for 循环时， name 被依次赋值为 word1 到 wordN 中的一个，然后执行 list 语句，再重复，因此， for 循环执行的次数取决于 word 列表中单词的个数。

六、select

- select 循环提供了一种简单的方式来创建一种用户可选择的有限菜单。其语法为：

```
select name in word1 word2 ..... wordN
do
    list
done
```

- 其中 name 是变量名， word1 到 wordN 是由空格分开的字符序列。

六、select

- select 循环执行时，先为 word1 到 wordN 的列表中的每一项建立一个菜单项，等待用户输入，用户输入了对应菜单项中的一个数字时，执行 list 的命令（通常是 case 结构），重复显示菜单；否则再次显示菜单，除非使用 break 等退出循环。

实例：打印菜单 testselect.sh

七、 break 和 continue

- 可以分别使用 break 和 continue 命令退出本层循环和本次循环。

第三章 shell 编程

- 本章主要内容包括：
- 第一节、 shell 脚本
- 第二节、变量
- 第三节、置换
- 第四节、引用
- 第五节、流控制
- 第六节、选项和参数
- 第七节、函数
- 第八节、文本过滤

第六节、选项和参数

- shell 中执行脚本的命令格式是：

  command opts args （命令 选项 参数）

- 本节主要介绍在 shell 脚本中处理选项和参数的方法，既可以使用 case 等语句对位置变量进行手动处理，也可以利用 getopts 完成选项和参数的处理。

第六节、选项和参数

- 一、使用 $0
- $0 表示被执行的命令的名字，对 shell 脚本来说，就是调用它的路径。
  - 用法一：获得调用脚本的命令名字
  - 用法二：用于脚本的使用声明

实例：为 myls.sh 脚本创建两个符号连接 lslong 和 lsall，用符号连接分别执行脚本，脚本根据使用的符号连接执行 ls-l 或 la -a 命令，如用 myls.sh 执行脚本，则显示使用声明

第六节、选项和参数

- 二、其他位置变量
- $1、$2、.....依次表示脚本的第一、第二、.....个参数，使用 case 等语句即可以进行选项和参数的处理。

第六节、选项和参数

- 三、使用 getopts
- getopts 命令的格式为:
- getopts option-string var
- 其中 option-string 是一个字符串，包含要处理的选项字符； var 是一个变量，通常名为 OPTION，用来保存当前选项字符。

第六节、选项和参数

- getopts 的执行过程为：
  - 1 、检查所有的命令行参数，发现以 - 开头的参数时，比较其后的字符是否在 option-string 中，若在，则把该选项字符赋值给 var 变量；否则， var 值为 ? 字符
  - 2 、检查选项字符后是否有 : 符号，有表示该选项需要附带参数，该参数由变量 OPTARG 保存；否则，转下
  - 3 、重复 1 、 2 ，直至所有选项检查完，返回一个非 0 ，并将变量 OPTIND 置为最后一个参数的下标

第一节、系统启动和关闭

- (4)init
- 它的作用是将系统带入另一个运行级别。这个命令有一个参数，即要进入的运行级别。例如：
- #init 0　　进入运行级别 0，即关机

验证：几种关闭方式的区别

一、用户和用户组

- 2、用户组和 GID
- 为便于集中管理，系统对用户进行分组，一个用户至少属于一个组(初始组)，也可以属于多个组(附加组)。
- 同用户帐户一样，系统也用一个数字表示用户组，这就是组 ID(GID)。

思考： GID 的范围是怎样规定的？

GID 的规定同 UID 类似，请参考 daizhe.cublog.cn 的参考资料

一、用户和用户组

注意：系统中有一类用户称为伪用户，这些用户在 /etc/passwd 文件中也有一条记录，但是不能登录，因为它们的登录 shell 为空。它们的存在主要是方便系统管理，满足相应的系统进程对文件属主的要求。如， bin 用户拥有可执行的用户命令文件， sys 用户拥有系统文件。

二、用户帐户管理

说明：

- 用户帐号刚创建时没有口令，所以被系统锁定，无法使用，必须为其指定口令后才可以使用，即使指定的是空口令。
- 超级用户可以为自己和其他用户指定口令，普通用户只能指定自己的口令。
- 普通用户修改自己的口令时会先询问旧口令，而超级用户则不会。
- passwd 命令如果缺省用户名，则修改当前用户的口令。

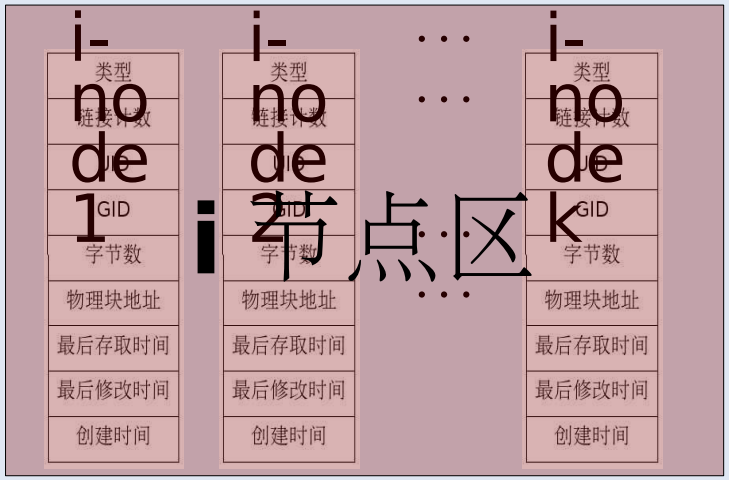
二、文件的访问权限

UNIX/Linux 系统的文件访问权限就是指什么样的用户可以对文件进行什么样的操作。

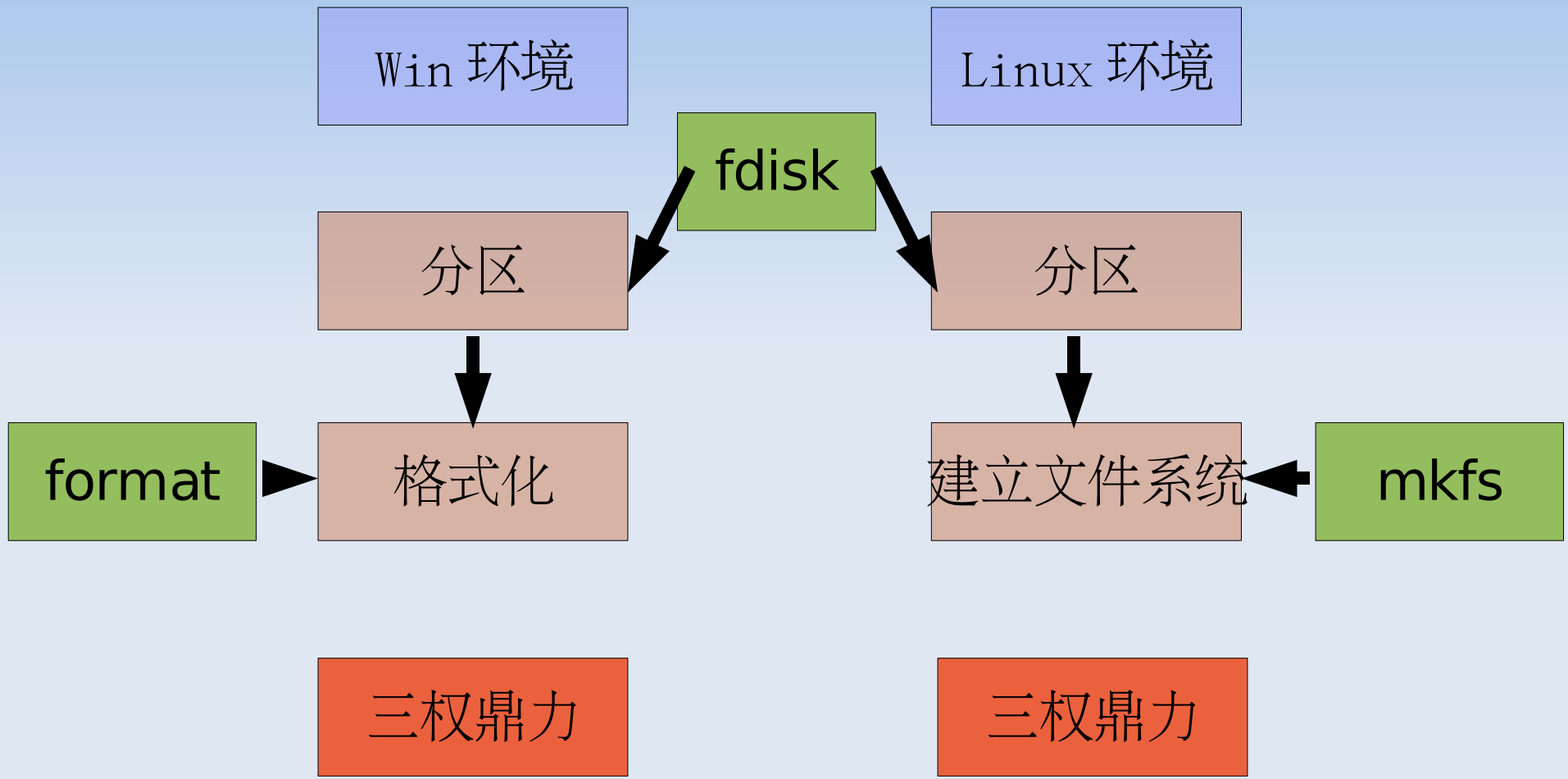
- 对文件的访问权限的管理是 UNIX/Linux 系统管理的一个重要内容。

三、文件系统及其使用(一)

目录项: file1 10
目录项: file2 13
目录项: file3 14
目录项: file4 19
……



三、文件系统及其使用（二）



三、文件系统及其使用(二)

- 1、分区
- (1) 对分区的规定：
- (a)Solaris： /dev/dsk/c#d#s#
  - c# controller number 逻辑控制器号
  - d# disk number 磁盘或逻辑单元号 (LUN)
  - s# slice or partition number 分区号
- 如： /dev/dsk/c0d0s0

指第一个 IDE 接口上的第一块硬盘的第一个分区

三、文件系统及其使用 (二)

- (b)Linux： /dev/hd*n 和 /dev/sd*n
- 第一个 IDE 设备为 /dev/hda，第二个 IDE 设备为 /dev/hdb，依此类推。
- 第一个 SCSI 设备为 /dev/sda，第二个 SCSI 设备为 /dev/sdb，依此类推。
- 而 n 表示分区号码。
- 如， hda1

表示第一个 IDE 硬盘的第一个分区

三、文件系统及其使用 (二)

- (2) 分区工具
- fdisk 是各种 Linux 发行版本中最常用的分区工具，是被定义为 Expert 级别的分区工具。
- fdisk 的使用请参考 daizhe.cublog.cn 中的参考资料

三、文件系统及其使用 (二)

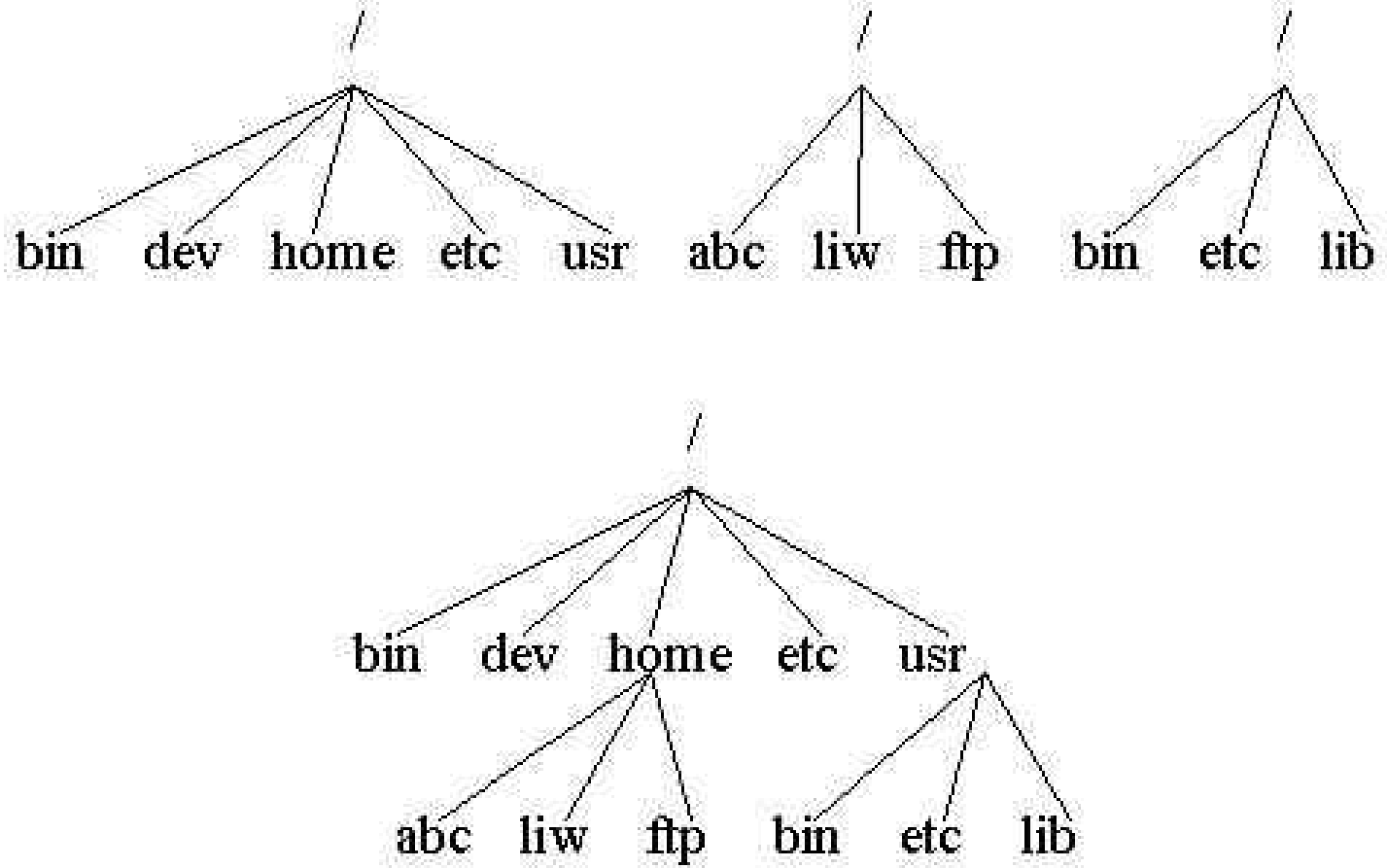
- 2 、建立文件系统 (mkfs/newfs/mke2fs)
- mkfs 命令
- 格式： mkfs 选项 参数
- 重要的选项是： -t fstype
- 例如： #mkfs –t ext2 /dev/hda1

在第一块 IDE 硬盘的第一个分区上建立一个格式为 ext2 的文件系统

三、文件系统及其使用(二)

- 6、文件系统的挂载
- 一个文件系统在可以使用之前，必须先进行挂载(mount)。操作系统做一些记录以确认正常。因为在 UNIX 中所有的文件在一个目录树中，装载操作的结构使新的文件系统的内容看起来好像在某个已经被装载的文件系统的一个已经存在的自目录中。例如，下图显示三个单独的文件系统，每个有其自己的根目录。当后两个文件系统分别被装载到第一个文件系统的 /home 和 /usr 目录时，将得到一个目录树。

三、文件系统及其使用(二)


/
bin
dev
home
etc
usr
/
abc
liw
ftp
/
bin
etc
lib
/
bin
dev
home
etc
usr
abc
liw
ftp
bin
etc
lib

三、文件系统及其使用(二)

- mount 命令用于挂载文件系统
- 格式： mount 选项 参数
- 选项：
  - -t fstype 指定文件系统类型
  - -o options
    - ro 以只读属性装载该分区
    - rw 以只读一写属性装载该分区（缺省值）
- 参数包括要装载的分区以及要装载到的目录。
- 其中要装载到的目录称为**挂载点**。

三、文件系统及其使用(二)

- 例如： #mount –o ro /dev/hda3 /usr

以只读方式装载/dev/hda3 分区到/usr 目录下

三、文件系统及其使用(二)

- 7、文件系统的卸载
- 当一个文件系统不再需要时，可以用 umount 命令卸载它。它的参数是设备文件或安装点。如，要卸载前面例子的目录，可以用：
- #umount /dev/hda3，或
- #umount /usr

第二章 UNIX 环境

- 本章主要内容包括：
- 第一节、系统启动和关闭
- 第二节、用户管理
- 第三节、文件系统管理
- 第四节、进程管理
- 第五节、作业调度
- 第六节、输入输出及重定向

第四节、进程管理

- 本节主要内容包括：
- 一、 UNIX/Linux 系统进程
  - 进程，前台、后台、悬挂进程，作业和守护进程，进程的状态及转换
- 二、查看进程
- 三、进程控制

一、UNIX/Linux 系统进程

- 1、进程、作业和守护进程
- (1) **进程**是一个程序在指定的数据集上的一次执行过程，是系统分配资源的基本单位。
- 一个进程通常包括：
  - 可执行程序代码（代码段）
  - 数据段
  - 打开的文件
  - 挂起的信号
  - 地址空间

一、UNIX/Linux 系统进程

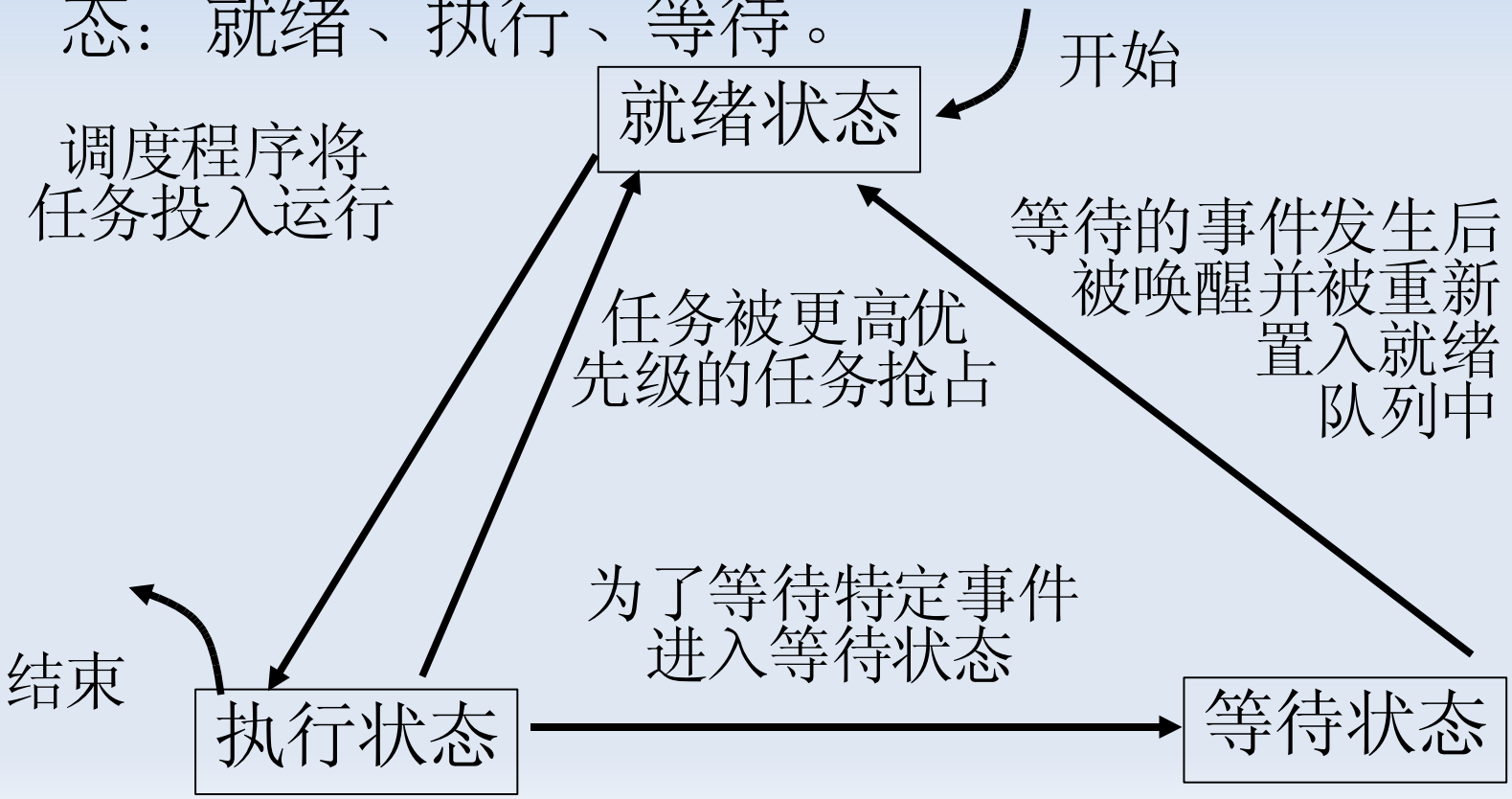
- (2) 前台、后台和悬挂进程
- 前台进程： shell 正在执行的进程，并且在它的执行过程中，不能访问 shell ，也不能执行其他程序
- 后台进程：也是 shell 正在执行的进程，但其执行过程中，可以访问 shell ，并执行其他程序
- 悬挂进程：被暂停的进程，还可以返回继续执行

一、UNIX/Linux 系统进程

- (3) 作业和守护进程
- 作业指被挂起或在后台运行的进程。
- 守护进程 **(Daemon)** 是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。

一、UNIX/Linux 系统进程

- 2、进程的状态
- 不同操作系统的规定不同，但至少应具备 3 种状态：就绪、执行、等待。



二、查看进程

- 可以使用 ps 命令查看系统种正在运行的进程的状态。其格式为：
- ps 选项
- 用于选择进程的选项：
  - -A/-e 列出所有活动进程
  - -a 列出在某终端上运行的所有进程
  - -r 仅列出正在运行的进程
- 用于控制输出格式的选项：
  - -f 以完全列表的形式输出
  - -l 以长格式输出

二、查看进程

- 进程状态码：
  - R　正在运行
  - S　正在睡眠
  - T　已经停止
  - <　高优先级进程
  - N　低优先级进程

二、查看进程

- 例如：
- #ps –ef

为列出当前系统中的活动进程的详细信息，分别有：进程所有者的用户 ID 号 (UID) 、进程的 ID 号 (PID) 、当前进程的父进程号 (PPID) 、使用 CPU 情况 (C) 、进程的启动事件 (STIME) 、终端标示号 (TTY) 、进程占用 CPU 时间 (TIME) 、与进程对应的命令名 (COMMAND) 。

```
root@dai-laptop:/home/dai# ps -ef
UID   PID  PPID  C  STIME  TTY   TIME      CMD
root   1    0     0  18:25   ?    :00:01    init [2]
```

三、进程控制

- 1 、在后台执行进程
- 将 & 操作符放在命令的最后，就可以使命令在后台执行，屏幕将立即显示此命令对应的进程 ID 号（用户可以使用进程 ID 号对进程进行跟踪和管理），并返回提示符状态。

```
root@dai-laptop:/home/dai# ps -ef &
[1] 12792
root@dai-laptop:/home/dai# UID        PID  PPID  C STIME TTY          TIME CMD
root   1     0      0  18:25   ?    :00:01     init [2]
```

三、进程控制

- 2 、将后台进程转为前台执行
- fg　[% 作业代号]
- 不代作业号时，将当前作业（正在使用 CPU 的作业）切换到前台。
-

三、进程控制

- 例如:
- $find / -name foo -print>foo.path 2>/dev/null &
- [1]13590
- $find / -name foobar -print>foobar.path 2>/dev/null &
- [2]13591

上述两条语句分别在后台执行 find 命令，得到作业号分别为 1 和 2，进程号分别为 13590 和 13591

三、进程控制

- $fg
- find / -name foo -print>foo.path 2>/dev/null
- <Ctrl+c>
- $fg %2
- find / -name foobar -print>foobar.path 2>/dev/null

两条语句分别将当前作业和作业号为 2 的作业切换到前台

三、进程控制

- 3 、进程休眠
- sleep 命令使进程暂停执行一段时间。参数的单位是秒，此命令多用于 shell 程序设计，使两条命令执行之间停顿一段时间。例如：
- #sleep 60 等待 60 秒后重新回到 $ 提示符

三、进程控制

- 4 、等待后台进程结束
- wait 命令用于等待后台进程结束。多用于 shell 程序设计，用来控制进程之间的时间顺序 。
- 例如：
- #wait 等待自己创建的后台进程全部结束
- #wait 2060 等待进程号为 2060 的后台进程的结束

三、进程控制

- 5 、 kill 命令
- kill 命令的用途是送一个信号 (signal) 给某一个进程 (process) 。因为大部份送的都是用来杀掉进程的 SIGKILL 或 SIGHUP ， 因此称为 kill 。
- kill 的用法为 : kill [-SIGNAL] pid ...
- SIGNAL 为一个的数字， 从 0 到 31 ， 其中 9 是 SIGKILL ， 也就是一般用来杀掉一些无法正常终止的信号。
- 你也可以用 kill -l 来查看可代替 signal 号码的数目字。

三、进程控制

- 例如:
- kill -9

杀掉系统内所有属于自己的 process

- kill 13590

杀掉进程号为 13590 的进程

第二章 UNIX 环境

- 本章主要内容包括：
- 第一节、系统启动和关闭
- 第二节、用户管理
- 第三节、文件系统管理
- 第四节、进程管理
- 第五节、作业调度
- 第六节、输入输出及重定向

第五节、作业调度

- 本节主要内容包括：
- 一、作业调度
- 二、操作 cron 守护进程
- 三、配置 cron

一、作业调度

每个小时去执行一遍 **/etc/cron.hourly** 内的脚本

系统提供的服务，程序员指定的
程序。这个服务就是 cron 守护进程。

01 * * * * root run-parts /etc/cron.hourly

结果送用户邮箱

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

crontab 文件

每分钟读一次

cron 守护进程

二、操作 cron 守护进程

- 可以使用以下命令操作 cron 服务：
  - /sbin/service crond start // 启动服务
  - /sbin/service crond stop // 关闭服务
  - /sbin/service crond restart // 重启服务
  - /sbin/service crond reload // 重新载入配置

三、配置 cron

- 1 、用 crontab 命令配置
- crontab 命令用来设定 cron 服务，其语法为:
- crontab [-n name/-l/-r/-e]
- 参数含义为:
  - -u name// 设定某个用户的 cron 服务，
  - crontab -l // 列出 cron 服务的详细内容
  - crontab -r // 删除某个用户的 cron 服务
  - crontab -e // 编辑某个用户的 cron 服务

三、配置 cron

- 例如：
- crontab -u root -l

root 查看自己的 cron 设置

- crontab -u fred -r

删除 fred 的 cron 设置

- crontab -u root -e

进入 vi 编辑模式，编辑 root 的 cron 设置

三、配置 cron

- 进入编辑模式后，可能看到如下内容:

```
# m h  dom mon dow   command
0 6 * * * echo "Good morning." >> /tmp/test.txt
```

- 每行代表一个要执行的任务，分为两部分，分别表示指定的时间和要执行的命令。

三、配置 cron

- 对日期的规定:
- 5 个数字，分别表示：
  - 分钟 (0-59)
  - 小時 (0-23)
  - 日期 (1-31)
  - 月份 (1-12)
  - 星期 (0-6)//0 代表星期天

三、配置 cron

- 3 个符号，分别表示:
  - / 代表每的意思 //*/5 表示每 5 个单位
  - - 代表从某个数字到某个数字
  - , 分开几个离散的数字

三、配置 cron

- 配置举例:
- 0 6 * * * echo "Good morning." >> /tmp/test.txt

每天早上 6 点输出 Good morning.

注意单纯 echo 从屏幕上看不到任何输出，因为 cron 把任何输出都 email 到 root 的信箱了

- 0 /2 * * * echo "Have a break now." >> /tmp/test.txt

每两个小时，输出 Have a break now.

三、配置 cron

- 0 23-7/2， 8 * * * echo "Have a good dream：） " >> /tmp/test.txt

晚上 11 点到早上 7 点之间每两个小时以及早上八点

- 0 11 4 * 1-3 command line

每个月的 4 号和每个礼拜的礼拜一到礼拜三的早上 11 点

- 0 4 1 1 * command line

1 月 1 日早上 4 点

三、配置 cron

- 2 、编辑 /etc/crontab 文件配置 cron
- cron 服务每分钟不仅要读一次 /var/spool/cron 内的所有文件，还需要读一次 /etc/crontab, 因此我们配置这个文件也能运用 cron 服务做一些事情。用 crontab 配置是针对某个用户的，而编辑 /etc/crontab 是针对系统的任务。

三、配置 cron

- 这个文件的内容可能是:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root // 如果出现错误，或者有数据输出，数据作为邮件发给这个帐号
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly // 每个小时去执行一遍 /etc/cron.hourly 内的脚本
02 4 * * * root run-parts /etc/cron.daily // 每天去执行一遍 /etc/cron.daily 内的脚本
22 4 * * 0 root run-parts /etc/cron.weekly // 每星期去执行一遍 /etc/cron.weekly 内的脚本
42 4 1 * * root run-parts /etc/cron.monthly // 每个月去执行一遍 /etc/cron.monthly 内的脚本
```

第二章 UNIX 环境

- 本章主要内容包括：
- 第一节、系统启动和关闭
- 第二节、用户管理
- 第三节、文件系统管理
- 第四节、进程管理
- 第五节、作业调度
- 第六节、输入输出及重定向

第五节、输入输出及重定向

- 本节主要内容包括：
- 一、常用输入输出方式
  - echo 、 printf 、 read 、 tee 、管道
- 二、标准输入、输出和错误
- 三、输入输出重定向

一、常用输入输出方式

- 1、echo
- 使用 echo 命令可以显示文本行或变量，或者把字符串输入到文件。它的一般形式为：
- echo string /$varname
- 其中， string 是想要显示的字符串，其中可以包含空格，也可以包含标点符号和格式化转义序列； varname 是一个变量名。

一、常用输入输出方式

说明:

- 可以包含的标点符号包括：？、：，、！等。
- 格式化转义序列是具有特殊含义的字符序列，shell 遇到它时，会使用另外的字符代替它。这里主要使用：
  - \n 代表换行
  - \t 代表 tab 制表符
  - \c 代表不换行
- 使用转义序列时要加双引号。
- 转义序列在 bash 、 zsh 中不起作用

一、常用输入输出方式

- 例如:

```
$echo Good Morning!// 显示字符串
$FRUIT=apple// 定义变量
$echo $FRUIT// 显示变量值
apple
```

一、常用输入输出方式

- 2 、 printf 命令
- 是 c 语言中 printf 函数的 shell 版本，提供了高度灵活的格式化输出。其格式为：
- printf 格式 参数
- 其中**格式**是一个包含一个或多个格式序列的字符串，**参数**是符合格式所规定的格式的字符串，这和 C 语言的 printf 函数格式是相似的。

一、常用输入输出方式

- 3、read
- 可以使用 read 语句从键盘或文件的某一行文本中读入信息，并将其赋给一个变量。其格式为：
- read varible1 varible2
- (1) 如果只指定了一个变量，那么 read 将会把所有的输入赋给该变量，直至遇到第一个文件结束符或回车。

```
root@dai-laptop:/home/dai/course# read NAME
zhang wang li zhao
root@dai-laptop:/home/dai/course# echo $NAME
zhang wang li zhao
```

一、常用输入输出方式

- (2) 如果给出了多个变量， shell 将以输入中的空格为界，将空格间的内容依次赋给各变量

```
root@dai-laptop:/# read NAME USERNAME
daizhe dai
root@dai-laptop:/# echo $NAME
daizhe
root@dai-laptop:/# echo $USERNAME
dai
```

一、常用输入输出方式

- (3) 如果输入文本域过长， shell 将所有的超长部分赋予最后一个变量

```
root@dai-laptop:/# read NAME USERNAME
zhang wang li zhao
root@dai-laptop:/# echo $NAME
zhang
root@dai-laptop:/# echo $USERNAME
wang li zhao
```

一、常用输入输出方式

- 4 、 tee
- tee 命令的作用可以用字母 T 来形象地表示。它把输出的一个副本输送到标准输出，另一个副本拷贝到相应的文件中。如果希望在看到输出的同时，也将其存入一个文件，那么这个命令再合适不过了。它的一般形式为：
- tee -a files
- 其中， -a 表示追加到文件末尾。

一、常用输入输出方式

- 例如：下面的命令将 who 的结果传给 tee， tee 将结果一方向传给标准输出及终端，一方向传给 who.out 文件

```
$ who | tee who.out
louise     pts/1     May 20 12:58 (193.132.90.9)
matthew    pts/0     May 20 10:18 (193.132.90.1)

cat who.out
louise     pts/1     May 20 12:58 (193.132.90.9)
matthew    pts/0     May 20 10:18 (193.132.90.1)
```

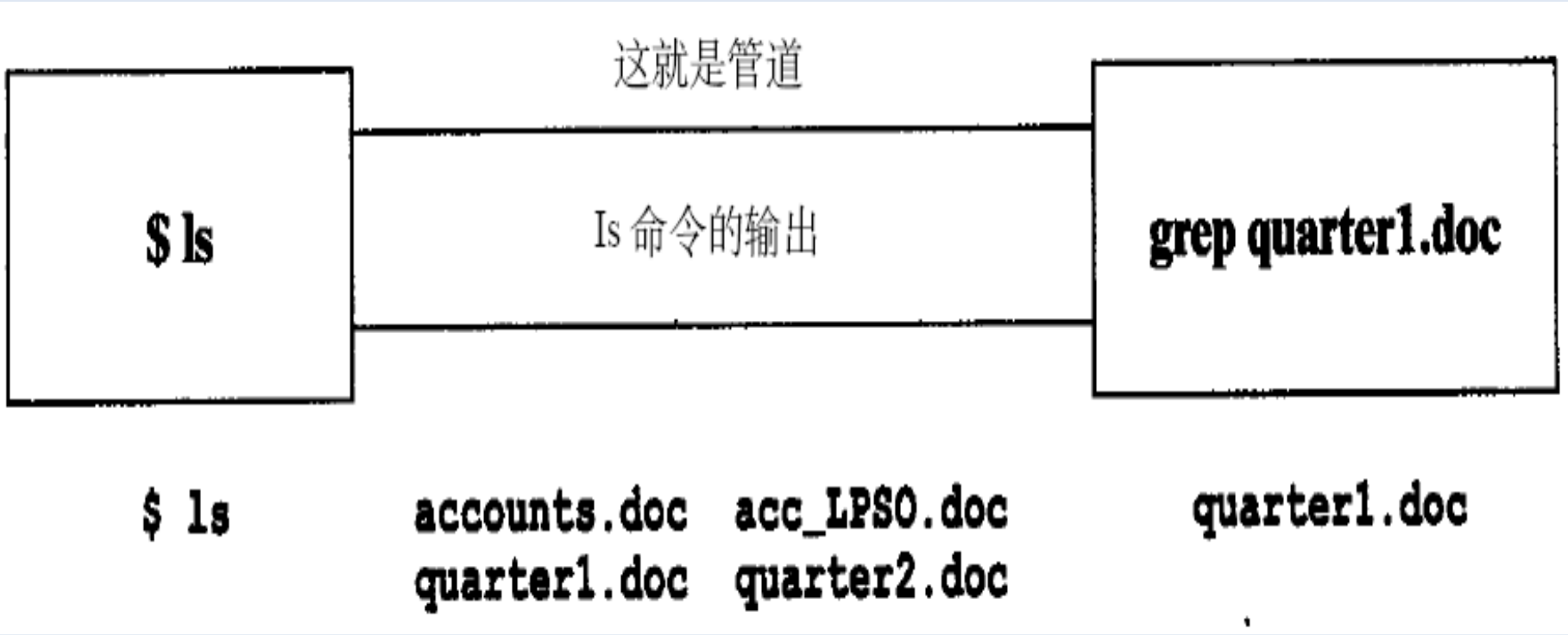
一、常用输入输出方式

- 5、管道
- 在 shell 编程中，使用一个程序来过滤另一个程序的输出是最常见的任务之一。可以使用管道将一个命令的输出重定向到另一个命令的输入，管道可以将好几个命令联系在一起，表示为：
- cmd1|cmd2|cmd3|.....

一、常用输入输出方式

- 下面的例子中， grep 命令在文件列表中搜索 quarter1.doc

root@dai-laptop:/# ls | grep quarter1.doc



一、常用输入输出方式

- 6、标准输入、输出和错误
- 当我们在 shell 中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。由于文件描述符不容易记忆，shell 同时也给出了相应的文件名。

文件	文件描述符
输入文件—标准输入	0
输出文件—标准输出	1
错误输出文件—标准错误	2

一、常用输入输出方式

- (1) 标准输入
- 标准输入是文件描述符 0 。它是命令的输入，缺省是键盘，也可以是文件或其他命令的输出。
- (2) 标准输出
- 标准输出是文件描述符 1 。它是命令的输出，缺省是屏幕，也可以是文件。
- (3) 标准错误
- 标准错误是文件描述符 2 。这是命令错误的输出，缺省是屏幕，同样也可以是文件。

一、常用输入输出方式

- 7、文件重定向
- 在执行命令时，可以指定命令的标准输入、输出和错误，要实现这一点就需要使用文件重定向。
- **在对标准错误进行重定向时，必须要使用文件描述符，但是对于标准输入和输出来说，这不是必需的。**
- (1) 重定向标准输出
- 可以把命令的输出重定向到文件中，而不是输出到标准输出。输出重定向符有：＞和＞＞。

一、常用输入输出方式

- 如果希望把标准输出重定向到文件中，可以用 >filename 。在下面的例子中， ls 命令的所有输出都被重定向到 ls.out 文件中：
- $ls >ls.out
- 如果希望追加到已有的文件中 (在该文件不存在的情况下创建该文件) ，那么可以使用 >>filename 。

一、常用输入输出方式

- 让我们来看一个标准输出的例子。在下面的命令中，把 /etc/passwd 文件中的用户 ID 域按照用户命排列。该命令的输出重定向到 sort.out 文件中。要提醒注意的是，在使用 sort 命令的时候 (或其他含有相似输入文件参数的命令)，重定向符号一定要离开 sort 命令两个空格，否则该命令会把它当作输入文件。
- $ cat passwd | awk -F: '{print $1}' | sort 1>sort.out

一、常用输入输出方式

- (2) 重定向标准输入
- 有些情况下需要为命令提供非交互方式的输入，可以将输入信息存储在某个文件中，通过重定向标准输入来传递给命令。下面给出一个这样的例子：
- $ sort < name.txt
- 在上面的命令中， sort 命令的输入是采用重定向的方式给出的，不过也可以直接把相应的文件作为该命令的参数：
- $ sort name.txt

一、常用输入输出方式

- 在上面的例子中，还可以更进一步地通过重定向为 sort 命令指定一个输出文件 name.out 。这样屏幕上将不会出现任何信息 (除了错误信息以外) :
- $ sort <name.txt >name.out

一、常用输入输出方式

- 重定向操作符 command << delimiter 是一种非常有用的命令， shell 将分界符 delimiter 之后直至下一个同样的分界符之前的所有内容都作为输入，遇到下一个分界符， shell 就知道输入结束了。这一命令对于自动或远程的例程非常有用。可以任意定义分界符 delimiter ，最常见的是 EOF 。 Delimiter 中不能包括空格或 tab 。也可以使用如下格式:
- cmd > file << delimiter
- document
- delimiter

一、常用输入输出方式

- (3) 重定向标准错误
- 为了重定向标准错误，可以指定文件描述符 2 。让我们先来看一个例子。在这个例子中， grep 命令在文件 missiles 中搜索 trident 字符串：
- $grep “trident” missiles
- Gerp:missiles:No such file or directory
- grep 命令没有找到该文件，缺省地向终端输出了一个错误信息。现在让我们把错误重定向到文件 /dev/null 中 (实际就上是系统的垃圾箱) ：
- $ grep "trident" missiles 2>/dev/null

一、常用输入输出方式

- 如果你在对更重要的文件进行操作，可能会希望保存相应的错误。下面就是一个这样的
- 例子，这一次错误被保存到 grep.err 文件中：
- $ grep "trident" missiles 2>grep.err
- $ pg grep.err
- Gerp:missiles:No such file or directory

一、常用输入输出方式

- 还可以把错误追加到一个文件中。在使用一组命令完成同一个任务时，这种方法非常有用。在下面的例子中，两个 grep 命令把错误都输出到同一个文件中；由于我们使用了 >> 符号进行追加，后面一个命令的错误 (如果有的话) 不会覆盖前一个命令的错误。
- $ grep “Lpso” * 2>>account.err
- $ grep “Sito” * 2>>account.err

本章小结

- 本章主要介绍 UNIX/Linux 系统的环境。
- 在系统的启动和关闭部分，详细介绍了系统的启动过程，系统在引导程序的引导下将内核载入内存，启动 init 进程，该进程安装 inittab 文件的规定启动跟当前运行级别相应的服务，另外一些服务在 rcn 和 rcn.d 的脚本中，启动这些服务后， init 创建终端，等待用户登录。
- 系统的关闭主要有 4 个命令 halt 、 reboot 、 shutdown 、 init 。

本章小结

- 在用户管理部分，介绍了 UNIX/Linux 系统对用户进行管理的规定，以及对用户和用户组进行管理的常用命令。
- 在文件系统管理部分，介绍了 UNIX/Linux 中的文件和文件类型，文件的访问权限，文件系统的概念及文件访问过程，文件系统的使用。
- 在进程管理部分，介绍了 UNIX/Linux 系统进程的概念，以及对进程进行管理的常用命令。

本章小结

- 在作业调度部分，介绍了 cron 守护进程的使用和配置方法。
- 最后，介绍 UNIX/Linux 系统的输入输出和重定向。

作业

- 1、写出 UNIX/Linux 系统启动的详细过程
- 2、写出创建组 dai，创建用户帐户 dai，其属组为 dai，主目录为 /home/dai，默认 shell 为 bash，修改其密码为 123 的命令
- 3、用两种方式写出将上述用户的主目录 /home/dai 的权限修改为对所有者可读、可写、可执行，同组和其他用户可读的命令。

作业

- 4、写出任意一 find 命令，将其转为后台执行
- 5、用 crontab 命令为当前用户添加一条每周 1 早上 8 点执行的输出“hello” 的计划任务