补充

[脚本组成 1](#_Toc323590626)

[管道 1](#_Toc323590627)

[标准IO&重定向 2](#_Toc323590628)

[脚本调试 3](#_Toc323590629)

[AND&OR 4](#_Toc323590630)

[引号 6](#_Toc323590631)

#### 脚本组成

|  |
| --- |
| 结构 |
| #!/bin/bash  do something  exit 0 |
| 步骤 |
| 编写: 使用vim，emacs或其他文本编辑器  命名: 通常以.sh结尾,并能比较清晰的说明脚本的作用,如backuplog.sh,mysqlbackup.sh等  权限: 脚本文件需要有可执行权限 chmod +x backuplog.sh  运行: ./backuplog.sh, bash backuplog.sh, bash -x backuplog.sh  使用: 结合场景可将脚本放在cron里面周期运行 |
| 优良的脚本 |
| 一个脚本应该无错运行并完成任务  程序的逻辑结构定义清晰而且明显,并有良好的注释  脚本可以重用 |

#### 管道

|  |
| --- |
| 管道 |
| 管道用“|”表示  管道符前后是两个命令, 前边命令的输出作为后边命令的输入  eg.  ls –l | grep ^d |

#### 标准IO&重定向

|  |
| --- |
| 标准I/O |
| 简介 |
| Linux 默认提供了三个I/O 通道：  进程启动后自动打开3个标准文件  Standard Input（标准输入：/dev/stdin，文件描述符：0） – 默认是键盘  0<stdin（可以重定向输入到文件等）  Standard Output（标准输出：/dev/stdout，文件描述符：1） – 默认是终端  1>stdout（可以重定向输出到文件等）  Standard Error（标准错误：/dev/stderr，文件描述符：2） – 默认是终端  2>stderr（可以重定向输出到文件等）  标准输出和标准错误可以强制重定向/dev/null |
| 示意图 |
| http://t1.gstatic.com/images?q=tbn:ANd9GcRMsxJmtkKe-aE07I5DfWu_JBseQCUxlMnOqbPbBAqKmqsaSKU0tg |
| 操作符 |
| > 输出重定向  >> 追加输出重定向  < 输入重定向  << 追加输入重定向 |
| 例子 |
| 1.重定向：复杂一点的例子  (  echo $header  cat ./data/myfile  echo $footer  ) >$htmlfile  exec </dev/null (or: exec <&-)  exec >$HOME/myapp/logs/myapp.log 2>&1  while read line; do  # do something  done < /tmp/myfile  2.重定向：恐怖的例子  exec **5**>>config.log  {  /bin/uname  /bin/hostname  } >&**5**  cat >&**5** <<\_ACEOF  ## ----------- ##  ## Core tests. ##  ## ----------- ##  \_ACEOF  3.特殊用法(常用) 2>dev/null 丢弃错误输出  4.重定向组合：输出、错误输出到同一文件 >outfile 2>&1(&表示替换，注意符号的顺序)  5.  command > filename　　　　　　把标准输出重定向到一个新文件中  command >> filename 　　　　　把标准输出重定向到一个文件中(追加)  command 1 > fielname　　　　　把标准输出重定向到一个文件中  command 1 >> fielname　　　　 把标准输出重定向到一个文件中(追加)  command 2 > filename　　　　　把标准错误重定向到一个文件中  command 2 >> filename　　　 　把标准错误重定向到一个文件中(追加)  command < filename　　　　　　把filename文件作为标准输入  command << delimiter　　　　　从标准输入中读入，直至遇到delimiter分界符  command <&m　　　　　　　　把把文件描述符m作为标准输入  command >&m 　　　　　　　　把标准输出重定向到文件描述符m中  command <&-　　　　　　　　　把关闭标准输入  不常用：  n<&- 表示将 n 号输入关闭  <&- 表示关闭标准输入（键盘）  n>&- 表示将 n 号输出关闭  >&- 表示将标准输出关闭 |

#### 脚本调试

|  |  |  |
| --- | --- | --- |
| 调试脚本 | | |
| 命令行选项 | Set选项 | 说明 |
| sh –n <script> | set –o noexec  set -n | 只检查语法错误，不执行命令 |
| sh –v <script> | set –o verbose  set –v | 在执行命令之前回显它们 |
| sh –x <script> | set –o xtrace  set –x | 在处理命令之后回显它们 |
| sh –u <script> | set –o nounset  set -u | 如果使用了未定义的变量，就给出出错信息 |
| -o选项启用设置 +o选项取消设置 | | |

更强大详细的调试方法：

<http://www.ibm.com/developerworks/cn/linux/l-cn-shell-debug/>

#### AND&OR

|  |
| --- |
| AND列表 |
| 作用：只有前面所有的命令都执行成功的情况下才执行最后一条命令  Statement1 && statement2 && statement3 && …..  从左到右顺序执行每条命令，如果一条命令返回true，右边的下一条命令才能执行  &&命令作用是检查前一条命令的返回值  if [ -f file\_one ] && echo “hello” && [ -f file\_two ] && echo “ there”  then  ……  fi |
| OR列表 |
| 作用：持续执行一系列命令，知道有一条命令成功为止  Statement1 || statement2 || statement3 || ….  从左顺序开始执行每条命令，若是一条命令返回false，它右边的下一条命令才能够被执行，如此持续到知道有一条命令返回true,或者列表中所有命令都执行完毕  if [ -f file\_one ] || echo “hello” || echo “ there” |

Bash

|  |
| --- |
| 为什么用 bash 而不是 sh |
| GNU Bash 主页  http://www.gnu.org/software/bash/  GNU Bash 手册  http://www.gnu.org/software/bash/manual/  Greg’s Wiki about GNU Bash <http://wooledge.org:8000/BashFAQ> http://wooledge.org:8000/BashGuide    更多的特性  $((3 + 4)) 而不需要 expr 3 + 4  /usr/{bin,local/bin} 而不需要 /usr/bin /usr/local/bin  ${str/src/dst} 而不需要 echo $str | sed ”s/$src/$dst/”  更方便的语法  for (( expr1; expr2; expr3 )); do  commands done  for (( i = 0; i < 100; i++ )); do … done  echo a{b,c,d}e ==> abe ace ade |
| 表达式求值 |
| $[] []$中间可以加表达式 eg: echo $[$a+$b]  $(()) (())中间可以加表达式。Eg： total=$(($a\*$b))  $[base#n]不同进制，n可能是0-32禁止 eg：$[10#8+1] 8进制的10=8+1=9 |
| 长度 |
| ${#变量名}得到字符串长度  test='I love china'  echo ${#test} |
| 截取字符串 |
| ${变量名:起始:长度}得到子字符串  $ test='I love china'  $ echo ${test:5}  e china  $ echo ${test:5:10}  e china |
| 字符串删除 |
| ${变量名#substring正则表达式}从字符串开头开始配备substring,删除匹配上的表达式。  ${变量名%substring正则表达式}从字符串结尾开始配备substring,删除匹配上的表达式。  注意：${test##\*/},${test%/\*} 分别是得到文件名，或者目录地址最简单方法  $ test='c:/windows/boot.ini'  $ echo ${test#/}  c:/windows/boot.ini  $ echo ${test#\*/}  windows/boot.ini  $ echo ${test##\*/}  boot.ini  $ echo ${test%/\*}  $ echo ${test%%/\*} |
| 字符串替换 |
| ${变量/查找/替换值} 一个“/”表示替换第一个，”//”表示替换所有,当查找中出现了：”/”请加转义符”\/”表示  $ test='c:/windows/boot.ini'  $ echo ${test/\//\\}  c:\windows/boot.ini  $ echo ${test//\//\\}  c:\windows\boot.ini |
| 正则表达式 |
| bash的正则表达式  str='hello, world'  if [[ $str =~ '\s+world$' ]]; then  echo match!  fi  if echo "$str" | grep -E '[ ]+world$'; then  echo match!  fi |

#### 引号

|  |
| --- |
| 组成 |
| 双引号：可以除了字符$`\外地任何字符或字符串  单引号：忽略任何引用值，将引号里的所有字符作为一个字符串  反引号：设置系统命令输出到变量  单引号告诉shell忽略所有特殊字符，而双引号只要求忽略大多数，具体说，括在双引号中的三种特殊字符不被忽略：$,\,` ,即双引号会解释字符串的特别意思,而单引号直接使用字符串.  echo $? 显示的是上一条指令退出状态  echo "$?" 效果同上  echo '$?' 显示的是$?  echo \$? 显示的是$?  echo "\$?" 显示的是$?  双引号对$符号不起作用而单引号可以将特殊字符的的特殊意义屏蔽掉,使其能显示为字符本身,反斜杠也可以将特殊字符的特殊含义屏蔽掉,使特殊字符失去特殊含义. |