**Morning：**

1. 字符串截取及切割（不会影响原值）

${#arr} ‘查看$arr字符串的长度

1. 字串截取三种用法：

-${变量名:起始位置:长度} ，起始位从0开始

例如： echo $arr ‘显示abcdefg

echo ${arr:1:3} ‘显示bcd

echo ${arr::4} ‘显示abcd

1. 使用 expr substr

-expr substr "$变量名" 起始位置 长度 ，起始位从1开始

例如： expr substr “$arr” 1 6 ‘显示abcdef

1. 使用cut分割工具，起始位从0开始

-echo $变量名 | cut -b 起始位置-结束位置

关于位置： p1-p2代表连续；

-p2 代表从头截取到p2；

p1- 代表从p1截取到尾；

p1,p2,p3,p4代表不连续，按顺序显示；

例如： echo $arr | cut -b 3-5 ‘显示cde

echo $arr | cut -b 1,3,5 ‘显示ace

Tips：arr=$a$b$c ‘字符串合并

1. 字符串的替换（不会影响原值）
2. 只替换第1个字串

-${变量名/old/new}

例如： echo $arr ‘显示abcddcba

echo ${arr/d/3} ‘显示abc3dcba

1. 替换全部字串

-${变量名//old/new}

例如： echo ${arr//d/3} ‘显示abc33cba

1. 字符串的匹配删除（不会影响原值）

1）从左向右，最短匹配删除（掐头）

-${变量名#\*关键词}

例如： echo $arr ’显示root:x:0:0:root:/root:/bin/bash

echo ${arr#\*:} ‘显示x:0:0:root:/root:/bin/bash

2）从左向右，最长匹配删除

-${变量名##\*关键词}

例如： echo ${arr##\*:} ‘显示/bin/bash

1. 从右向左，最短匹配删除

-${变量名%关键词\*}

例如： echo ${arr%:\*} ‘显示root:x:0:0:root:/root

4）从右向左，最长匹配删除

-${变量名%%关键词\*}

例如： echo ${arr%%:\*} ‘显示root

1. 字符串初值的处理

-${var:-word}

例如：echo ${arr:-123} ‘若$arr有值，原值输出；否则初始化输出

Tips：不会自动赋值给变量，需要arr=${arr:-123}

**Afternoon：**

1. expect预期交互

1）mail的知识储备：

-mail -s test student #交互式，以.结尾

-echo “hello” | mail -s test student #非交互式，管道

-mail -s input root < /root/test.txt #从文件导入（有依赖）

-mail -s input root << EOF #EOF .... EOF（无依赖）

This is a test mail...

EOF

1. expect预期交互格式：

必须事先yum -y install expect

只能对spawn所指定的操作下进行预期交互

expect << EOF

spawn ssh -o StrictHostKeyChecking=no 172.25.0.10

expect "password" { send "redhat\n"} ‘关键字、命令

expect "#" { send "touch /123.txt\n"}

expect "#" { send "exit\n"}

expect ‘默认不执行

EOF

Tips： 1）expect脚本的最后一行默认不执行

2）如果不希望ssh时出现yes/no的提示，远程时使用如下选项:

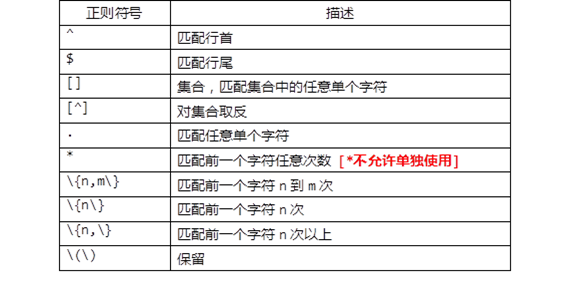
ssh -o StrictHostKeyChecking=no server0

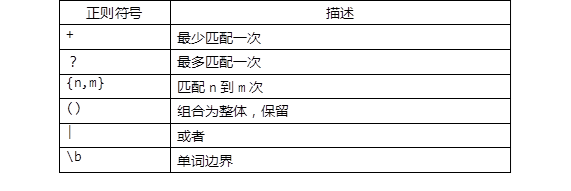
3）expect等待超时时间默认为10s，通过set timeout -1取消， 或者set timeout 20 设为20s

4）spawn 程序，下面的指令必须是建立在该进程的，否则不执行或者直接报错

1. 正则表达式

**-过滤、匹配数据。通用表达式**





1. grep -v(取反) -i(忽略大小写) ，^$代表空行
2. 使用正则表达式时最好使用””

**-正则表达式-元字符**

|  |  |  |
| --- | --- | --- |
| 1、字符类 | | |
| . | 匹配任意单个字符(除\n、\r) | |
| [xyz] | 匹配1个包含的字符。 | |
| [^xyz] | 匹配1个不包含的字符 | |
| [0-9] | 匹配1个范围内字符。 | |
| \d | 数字字符。等价于[0-9] | |
| \D | 非数字字符。等价于[^0-9] | |
| \w | 字母、数字、下划线。等价于[A-Za-z0-9\_] | |
| \W | 非字母、数字、下划线。等价于[^A-Za-z0-9\_] | |
| \s | 匹配1个空白字符。等价于[\f\n\r\t\v] | |
| \S | 匹配1个非空字符。 | |
| \f \n \r \t \v | 换页符、换行符、回车符、制表符、垂直制表符 | |
| 2、数量限定符 | | |
| \* | 匹配前面0次或多次。等价于{0,} | |
| + | 匹配前面1次或多次。等价于{1,} | |
| ? | 匹配前面0次或1次。等价于{0,1} | |
| {m} | 匹配前面m次 | |
| {m,} | 匹配前面≧m次 | |
| {,n} | 匹配前面≦n次 | |
| {m,n} | 匹配前面m~n次(包含m,n) | |
| 3、位置限定符 | | |
| ^ | 匹配行首 | |
| $ | 匹配行尾 | |
| \b | 匹配单词边界 | |
| \B | 匹配非单词边界 | |
| 4、特殊字符 | | |
| (abc) | 组合为整体。如IP匹配：([0-9]{1,3}\.){3}[0-9]{1,3} | |
| \ | 转义字符。普通字符->特殊、特殊字符->普通 | |
| | | 或者。如：n(o|either)匹配no或neither | |
| 5、未理解 | | |
| ? | | 当该字符紧跟在任何一个其他限制符 (\*, +, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串 "oooo"，'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。 |
| (?:pattern) | | 匹配 pattern 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例如， 'industr(?:y|ies) 就是一个比 'industry|industries' 更简略的表达式。 |
| (?=pattern) | | 正向肯定预查（look ahead positive assert），在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，"Windows(?=95|98|NT|2000)"能匹配"Windows2000"中的"Windows"，但不能匹配"Windows3.1"中的"Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。 |
| (?!pattern) | | 正向否定预查(negative assert)，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如"Windows(?!95|98|NT|2000)"能匹配"Windows3.1"中的"Windows"，但不能匹配"Windows2000"中的"Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。 |
| (?<=pattern) | | 反向(look behind)肯定预查，与正向肯定预查类似，只是方向相反。例如，"(?<=95|98|NT|2000)Windows"能匹配"2000Windows"中的"Windows"，但不能匹配"3.1Windows"中的"Windows"。 |
| (?<!pattern) | | 反向否定预查，与正向否定预查类似，只是方向相反。例如"(?<!95|98|NT|2000)Windows"能匹配"3.1Windows"中的"Windows"，但不能匹配"2000Windows"中的"Windows"。 |
| \cx | | 匹配由 x 指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。 |
| \xn | | 匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，'\x41' 匹配 "A"。'\x041' 则等价于 '\x04' & "1"。正则表达式中可以使用 ASCII 编码。 |
| \num | | 匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，'(.)\1' 匹配两个连续的相同字符。 |
| \n | | 标识一个八进制转义值或一个向后引用。如果 \n 之前至少 n 个获取的子表达式，则 n 为向后引用。否则，如果 n 为八进制数字 (0-7)，则 n 为一个八进制转义值。 |
| \nm | | 标识一个八进制转义值或一个向后引用。如果 \nm 之前至少有 nm 个获得子表达式，则 nm 为向后引用。如果 \nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的向后引用。如果前面的条件都不满足，若 n 和 m 均为八进制数字 (0-7)，则 \nm 将匹配八进制转义值 nm。 |
| \nml | | 如果 n 为八进制数字 (0-3)，且 m 和 l 均为八进制数字 (0-7)，则匹配八进制转义值 nml。 |
| \un | | 匹配 n，其中 n 是一个用四个十六进制数字表示的 Unicode 字符。例如， \u00A9 匹配版权符号 (?)。 |