# 语法：

$\*表示所有这些参数都被双引号引住。若一个脚本接收两个参数，$\*等于$1$2

$@表示所有这些参数都分别被双引号引住，若一个脚本接收到两个参数，$@等价于$1$2

$#表示提供给脚本的参数号

运行test.sh 1 2 3后

$\*为"1 2 3"（一起被引号包住）

$@为"1" "2" "3"（分别被包住）

$#为3（参数数量）

# bash\_case

|  |
| --- |
| #!/bin/sh  #$#:传递给程序的总的参数数目  #$n:表示第几个参数，$1 表示第一个参数，$2 表示第二个参数 ...  #$0:当前程序的名称  #$\*:传递给程序的所有参数组成的字符串  #$@:以"参数1" "参数2" ... 形式保存所有参数  case "$1" in  "a" | "A" )  echo A  ;;  "b" | "B" )  echo B  ;;  "c" | "C" )  echo C  ;;  \*)  echo no parameters  ;;  esac |

# bash\_param

|  |
| --- |
| #!/bin/sh  #$#:传递给程序的总的参数数目  #$n:表示第几个参数，$1 表示第一个参数，$2 表示第二个参数 ...  #$0:当前程序的名称  #$\*:传递给程序的所有参数组成的字符串  #$@:以"参数1" "参数2" ... 形式保存所有参数  #compare strings  if [ "$1" = "abcde" ];then  echo $1 match "abcde"  elif [ "$1" = "none" ];then  echo $1 match "none"  else  echo $1 not match "abcde" or "none"  fi |

# 循环读cpu状态：

|  |
| --- |
| while true; do  echo " "  cat /proc/stat  sleep 1  done |

# 循环读寄存器值

|  |
| --- |
| while true; do  echo " "  busybox devmem 0xe0168000 32  busybox devmem 0xe01b0054 32  sleep 0.5  done |

# preview

|  |
| --- |
| #!/system/bin/sh  while [ true ]; do  echo "Goto take a picture..."  am start -n com.android.camera2/com.android.camera.CameraActivity  sleep 5  input keyevent HOME  sleep 1  echo "done"  done |

# take\_picture

|  |
| --- |
| #!/system/bin/sh  echo "Goto take a picture..."  am start -n com.android.camera2/com.android.camera.CameraActivity  sleep 5  input keyevent ENTER  sleep 3  input keyevent ENTER  sleep 3  input keyevent HOME  sleep 1  echo "done"  exit 0 |

# take\_video

|  |
| --- |
| #!/system/bin/sh  echo "Goto take a video..."  am start -n com.android.camera2/com.android.camera.CameraActivity  sleep 5  input keyevent DPAD\_DOWN  sleep 3  input keyevent ENTER  sleep 10  input keyevent ENTER  sleep 3  input keyevent HOME  sleep 1  echo "done"  exit 0 |

# bash\_freq

|  |
| --- |
| #!/system/bin/sh  case "$1" in  "1" )  freq=696000  echo 1 : ${freq}  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor  echo ${freq} > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  ;;  "2" )  freq=1008000  echo 2 : ${freq}  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor  echo ${freq} > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  ;;  "3" )  freq=1260000  echo 3 : ${freq}  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor  echo ${freq} > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  ;;  "4")  freq=1404000  echo 4 : ${freq}  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor  echo ${freq} > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  ;;  "5")  freq=1536000  echo 5 : ${freq}  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor  echo ${freq} > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  ;;  \*)  echo no parameters  echo 1 : 696000  echo 2 : 1008000  echo 3 : 1260000  echo 4 : 1404000  echo 5 : 1536000  ;;  esac |

|  |
| --- |
|  |

|  |
| --- |
|  |

# 2.2.1、基本的正则表达式：

**（1）元字符：**

. 匹配任意单个字符

fg：查找包含student且student后面带一个字符的行

grep ‘student.’ etcpasswd （模式可以用单引号和双引号，如果模式中要做变量替换时则必须用双引）

[] 匹配指定范围内的任意单个字符,[abc],[a-z],[0-9],[a-zA-Z]

fg：查找带有数字的行

grep ‘[0-9]’ etcpasswd

[^] 匹配指定范围外的任意单个字符

fg：查找没有小写字母的行。

grep ‘[^a-z]’ etcinittab

[space]表示空白字符

[punct]表示所有标点符号的集合

[lower]表示所有的小写字母

[upper]表示所有的大写字母

[alpha]表示大小写字母

[digit]表示数子

[alnum]表示数字和大小写字母-----使用格式[[alnum]]等

**（2）次数匹配：**

匹配其前面的字符任意次

fg：查找root出现0次或0次以上的行

grep ‘root’ etcpasswd

. 任意字符

fg：查找包含root的行

grep 'root.' etcpasswd

：匹配其前面的字符1次或0次

{m,n} 匹配其前字符最少m，最多n次）

**(3) 字符锚定：**

^锚定行首，此字符后面的任意内容必须出现在行首

fg：查找行首以#开头的行

grep '^#' etcinittab

$锚定行尾，此字符前面的任意内容必须出现在行尾

fg：查找行首以root结尾的行

grep 'root$' etcinittab

^$锚定空白行，可以统计空白行

或者b锚定词首，其后面的任意字符必须做为单词首部出现

fg查找root且root前面不包含任何字符的行

grep '

或者b锚定词尾，其前面的任意字符必须做为单词尾部出现

fg： 查找root单词

grep =grep brootb

# 2.2.2、扩展的正则表达式：

扩展的正则表达只是在基本的正则表达上作出了小小的一点修改，其修改如下：

在扩展的正则表达中把( ) 写成()、{ } 写成{ }，另外加入了+：次数匹配，匹配其前面的字符至少出现一次，无上限、 或者(二取一），其余的都一样， 基本正则表达式，使用( ) { } . 都需要转义,在扩展正则表达中不需要加，其详细信息如下：

(1) 字符匹配的命令和用法与基本正则表达式的用法相同，这里不再重复阐述。

(2) 次数匹配：

匹配其前面字符的任意次

？匹配其前面字符的0此或着1此

+ 匹配其前面字符至少1此

fg：至少一个空白符： '[[space]]+'

{m,n} 匹配其前面字符m到n次

(3) 字符锚定的用法和基本正则表达式的用法相同，在此不再阐述。

（4）特殊字符：

代表或者的意思。

fg：grep -E 'ccat' file：表示在文件file内查找包含c或者cat

.表示转义字符，此表示符号.

# End