

//-------------------------------------------------------------------------

${变量}或者$变量 引用变量值

$(pwd)或者`pwd` 调用命令（反引号）

\ 转义

' ' 所有都当做普通字符输出，正则表达式用的单引号

" " 所有都当做普通字符输出，除$, `, \外，如：$变量，`命令`，\$，\`，\\等

位置参数变量：$n: $0代表命令本身，$1~n代表参数，n>10,最好${10}

$#: 代表命令行参数个数，不包含脚本名命令本身参数

$\*: 代表命令行所有参数，视为一个整体

$@: 代表命令行所有参数，不视为一个整体

预定义变量： $?: 上一次进程退出的状态值（成功为0，反之不为0）

$$：查看Shell本身的PID，即当前进程id号

$!：Shell最后运行的后台最后一个进程id号

在shell中,$@和$\*都表示命令行所有参数(不包含$0),但是$\*将命令行的所有参数看成一个整体，而$@则区分各个参数

管道：|（有时候作条件 或）

通配符：？、 \* 、 []

? （匹配任意一个字符）

\* （匹配任意字符）

[] （匹配中括号中范围内的一个字符）

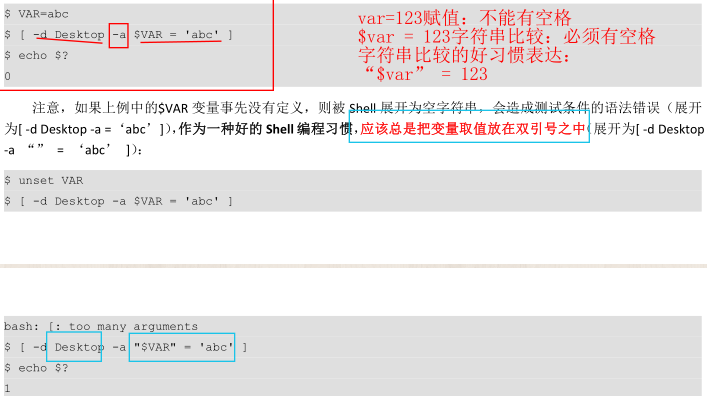
如： [0-9a-Za-Z] （匹配0~9及26个大小写字母任意一个字符）

[^a-z]（匹配a~z以外的一个字符）[!a-z]

算数运算：（（））、$（（））、$[]

a=10 执行算数操作： （（a++）），（（a+=1）），（（a+1））等操作

c=$(( $a + b$ )) 或者c=$((a+b)) 或者 c=$[a+b] （将a与b的值加起来赋给c）



【for结构的数值循环：（特殊）】

for (( i=0;i<10;i++ ))

do

echo $i

done

【管道在if语句使用：】

if [ $(ls | grep "case.sh") == "case.sh" ];then

echo 1

else

echo 2

fi

【函数：】先定义后使用，无返回值和形参

function fun1() {

xxxxx；

} #定义（ { 要有空格末尾有分号，function关键字可去 ）

fun1 #调用（不写括号）

【重定向输出】&>与> 以下&>>与>>同理

文件描述符我们常见的就是系统预留的0，1和2这三个，他们的意义分别有如下对应关系：

* 0 —— stdin（标准输入）
* 1 —— stdout （标准输出）
* 2 —— stderr （标准错误）

使用方式：

* 命令 >文件路径 （将一切结果覆盖方式保存）
* 命令 >>文件路径 （将一切结果追加方式保存）
* 命令 1>>文件1路径 2>>文件2路径（将命令正确结果保存于文件1，错误结果保存于文件2，追加方式）
* 命令 2>>/dev/null （将错误输出扔掉）

注：>等同于1>，2>&1（将标准错误输出重定向到标准输出）等同 &> 等同 >&

【常用指令】使用grep（BasicRE）匹配正则时，字符?、+、{}、|、()应解释为普通字符，要表示上述特殊含义则需要加\转义！！！

grep ‘ … ’ file1 匹配file1文件中的…（多文件匹配可在file1后加file2等文件）

grep -n ‘ … ’ file1 显示匹配及行号

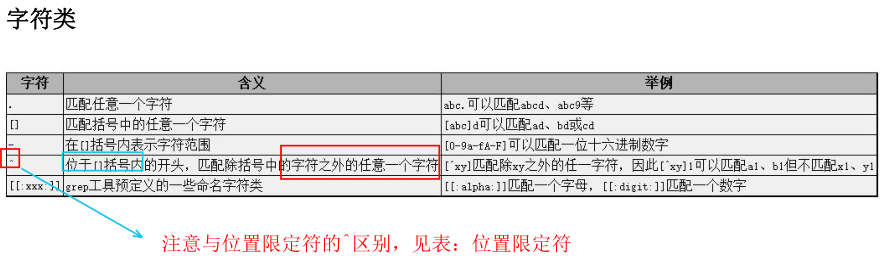
-v 显示不包括匹配行的所有行

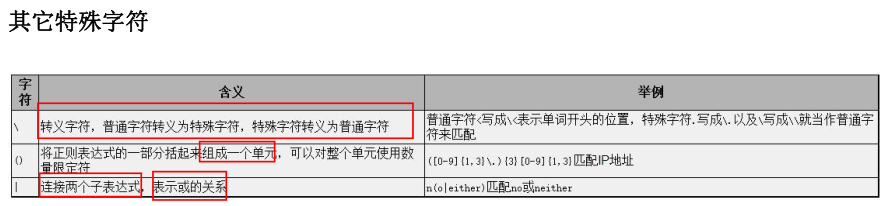
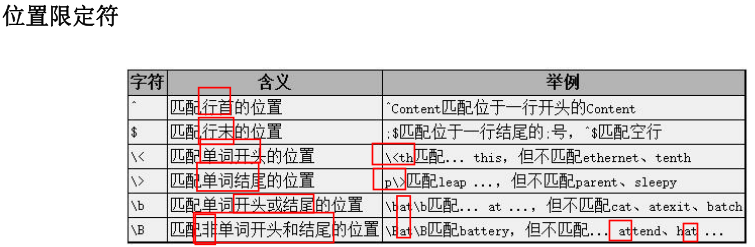
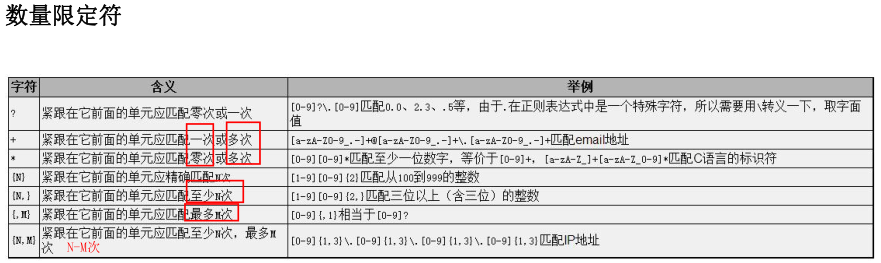
grep -E ‘…’ file1 正则模式字符串匹配=egrep

【基础正则:】支持的命令有：grep,sed,awk



【扩展正则:】支持的命令有:egrep, sed -r, awk





read -p “…” var1 显示提示信息…并读取键盘输入到var1变量

echo "…" >> file1 标准输出重定向追加到file1

echo -e "hello\n" 解析转义字符-e

find pathxx -name [-print –exec -ok] 按照名称查找，然后选择打印、执行的命令、安全执行的命令

如：find . -name "\*.sh" -exec ls -l {} \; #换为-ok也可以

find . -name "\*.sh" -print #打印

find . -name "\*" -print | xargs file # xarg调用file命令查看文件类型

cut -f 列号 -d "分割符" 文件路径 （以分割符提取第x列按列提起，cut只擅长处理“以一个字符间隔”的文本内容）

printf "%-10s %-8s %-.2f**\n**"  杨过 男  48.6543（%s，%f字符格式，-表示左对齐，不写表示右对齐，.2f只保留2位小数）

【sed】（流编辑器，以行为单位处理）：sed -n（只输入处理后的流结果，使用删除d就不要用-n参数，-i将修改的结果写入文件）

sed默认使用Basic正则，-r使用扩展正则

追加a "na xx" 在第n行后加入xx

插入i "ni xx" 在第n行前插入xx

行替换c "nc xx" 在第n行替换为xx

删除d "nd" n行

打印p "np" n行 #注："2,4p" 2到4行

字符替换s ns/旧字符1/新字符2/g #将第n行的字符串1替换为字符串2，n不写默认为全部

eg：sed "2,5d" testf #删除2-5行

sed -n ‘/-rwxr-xr-x/p’ a.txt #打印包含-rwxr-xr-x的行

sed -n ‘s/gj/guojian/g’ a.txt

sed 's/yes/no/g;s/static/dhcp/g' ./testfile #多字符替换，用;隔开，./表示当前文件夹

注意：.\*、.?、.+的贪婪匹配，如：去除html标签，要有终止条件

# sed 's/<.\*>//g' testf #不行

sed 's/<[^>]\*>//g' testf #可以，^符号后接普通字符，表示除了该字符外，接正则，表示行首

【awk】可以处理行、列，配合print $n; 语句，输出某一列

awk '{print $2;}' testf #$0全部内容，$1表示第一列

awk '$2<75 { printf "%s\t%s\n", $0, "REORDER"; } $2>=75 {print $0; }' testf

awk '/^ \*$/{ x=x+1; } END{ print x; }' testf #两个特殊的条件BEGIN、END

注：BEGIN：在处理文件之前执行一次后面紧跟的action

END：在处理文件完成后执行一次后面紧跟的action

awk 'BEGIN {FS=":"} { print $1;}' /etc/passwd # FS指定列分割符或者： awk -F ':'

 将要执行的命令写入文件**xx.awk**，然后-f调用，如写入BEGIN{ FS=":"} {print $1}

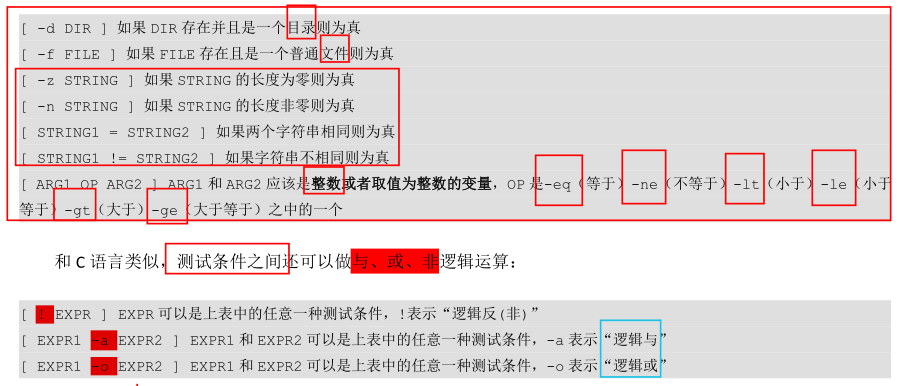
awk -f script.awk  file1 #**调用脚本文件作用于file1文件**

【典型错误】：+被当成拼接

var=1

var=$var+1

echo $var 结果：1+1 #只会拼接！！！



-------------------------------------------

cat /etc/shells（查看系统支持的shell）

echo $SHELL （查看当前进程或者当前shell类型）

exit （退出当前shell）

进入其他shell：命令行输入shell名，如sh，退出用exit

部分内接命令不需要shell解释器也能执行：man bash-butilins

env查看环境变量；export 变量名：将该本地变量提升为环境变量

unset：删除环境变量或本地变量

alias：查看命令别名；alias xx=“xxx” 给xxx指令取别名xx

alias （查看别名列表）

alias 别名='原命令'（设定命令别名）

unalias 别名 （删除别名）

（别名永久生效文件/root/.bashrc）

history （查看历史命令）

-c（清除历史命令）

-w（写入用户家目录下的.bash\_history文件中，/etc/profile默认只存放1000条）

-------------------------------------------