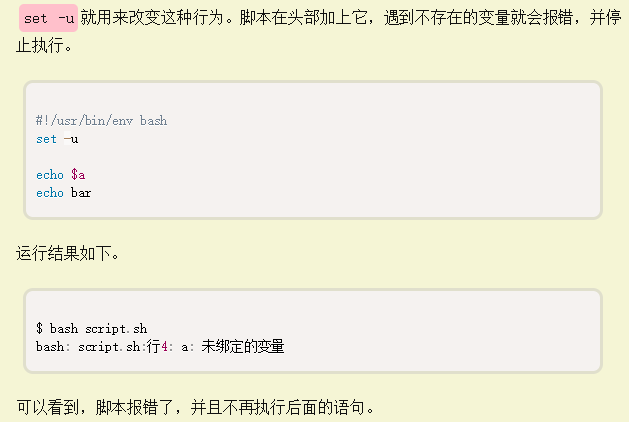
# set 命令

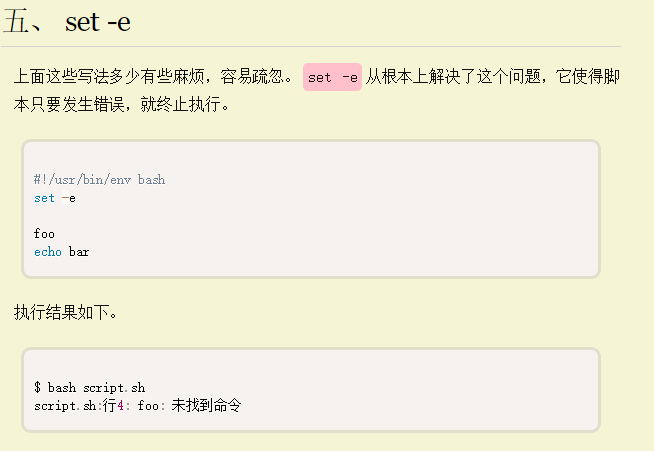
set –u



Set –x



Set –e



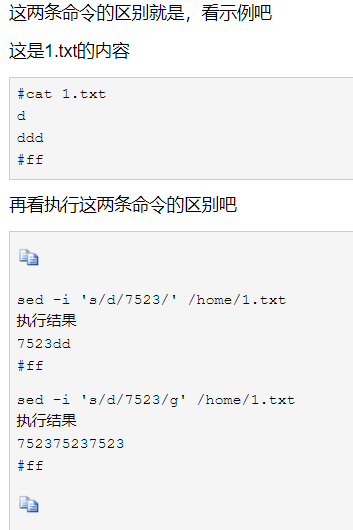
# **sed -i**

sed -i 就是直接对文本文件进行操作的。

sed -i 's/原字符串/新字符串/' /home/1.txt

sed -i 's/原字符串/新字符串/g' /home/1.txt

就是把 home目录下 1.txt 中 的 “原字符” 替换成 “新字符”。



# shell脚本if判断变量介绍

| **变量** | **含义** |
| --- | --- |
| [ -a FILE ] | 如果 FILE 存在则为真 |
| [ -b FILE ] | 如果 FILE 存在且是一个块特殊文件则为真 |
| [ -c FILE ] | 如果 FILE 存在且是一个字特殊文件则为真 |
| [ -e FILE | 如果 FILE 存在则为真 |
| [ -f FILE ] | 如果 FILE 存在且是一个普通文件则为真 |
| [ -g FILE ] | 如果 FILE 存在且已经设置了SGID则为真 |
| [ -h FILE ] | 如果 FILE 存在且是一个符号连接则为真 |
| [ -k FILE ] | 如果 FILE 存在且已经设置了粘制位则为真 |
| [ -r FILE ] | 如果 FILE 存在且是可读的则为真 |
| [ -s FILE ] | 如果 FILE 存在且大小不为0则为真 |
| [ -t FD ] | 如果文件描述符 FD 打开且指向一个终端则为真 |
| [ -u FILE ] | 如果 FILE 存在且设置了SUID (set user ID)则为真 |
| [ -w FILE ] | 如果 FILE 如果 FILE 存在且是可写的则为真 |
| [ -x FILE ] | 如果 FILE 存在且是可执行的则为真 |
| [ -O FILE ] | 如果 FILE 存在且属有效用户ID则为真 |
| [ - G FILE ] | 如果 FILE 存在且属有效用户组则为真 |
| [ -L FILE ] | 如果 FILE 存在且是一个符号连接则为真 |
| [ -N FILE ] | 如果 FILE 存在 and has been mod如果ied since it was last read则为真 |
| [ -S FILE ] | 如果 FILE 存在且是一个套接字则为真 |
| [ -z STRING ] | “STRING” 的长度为零则为真 |

## 字符串判断

| **变量** | **含义** |
| --- | --- |
| str1 = str2 | 两个字符串完全相等为真 |
| str1 != str2 | 两个字符串不完全相等为真 |
| -n str1 | 当串的长度大于0时为真(串非空) |
| -z str1 | 当串的长度为0时为真(空串) |
| str1 | 当串str1为非空时为真 |

## 数字判断

| **变量** | **含义** |
| --- | --- |
| int1 -eq int2 | 两数字相等为真 |
| int1 -ne int2 | 两数字不相等为真 |
| int1 -gt int2 | int1大于int2为真 |
| int1 -ge int2 | int1>= int2为真 |
| in1 -lt int2 | int1<int2 为真 |
| int1 -le int2 | int1 <= int2为真 |

## 文件判断

| **变量名** | **含义** |
| --- | --- |
| -r file | 用户可读为真 |
| -w file | 用户可写为真 |
| -f file | -文件为正规文件为真 |
| -x file | 用户可执行为真 |
| -d file | 文件为目录为真 |
| -c file | 闻见味特殊字符文件为真 |
| -s file | 文件大小非0为真 |
| -b file | 文件为块特殊文件为真 |
| -t file | 文件描述符(默认1)指定的设备为终端时为真 |

## 复杂逻辑

| **变量名** | **含义** |
| --- | --- |
| -a | 与 |
| -o | 或 |
| ! | 非 |
| != | 不等于 |

# 比较两个字符串是否相等的办法是：

if [ "$test"x = "test"x ]; then

这里的关键有几点：

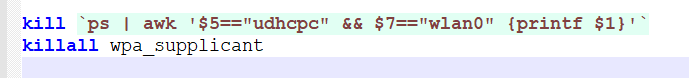
1 使用单个等号

2 注意到等号两边各有一个空格：这是unix shell的要求

3 注意到"$test"x最后的x，这是特意安排的，因为当$test为空的时候，上面的表达式就变成了x = testx，显然是不相等的。而如果没有这个x，表达式就会报错：[: =: unary operator expected

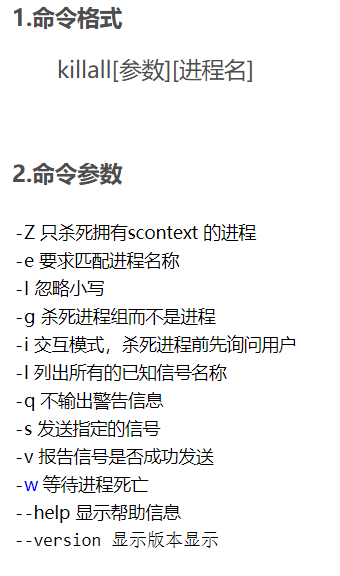
# awk

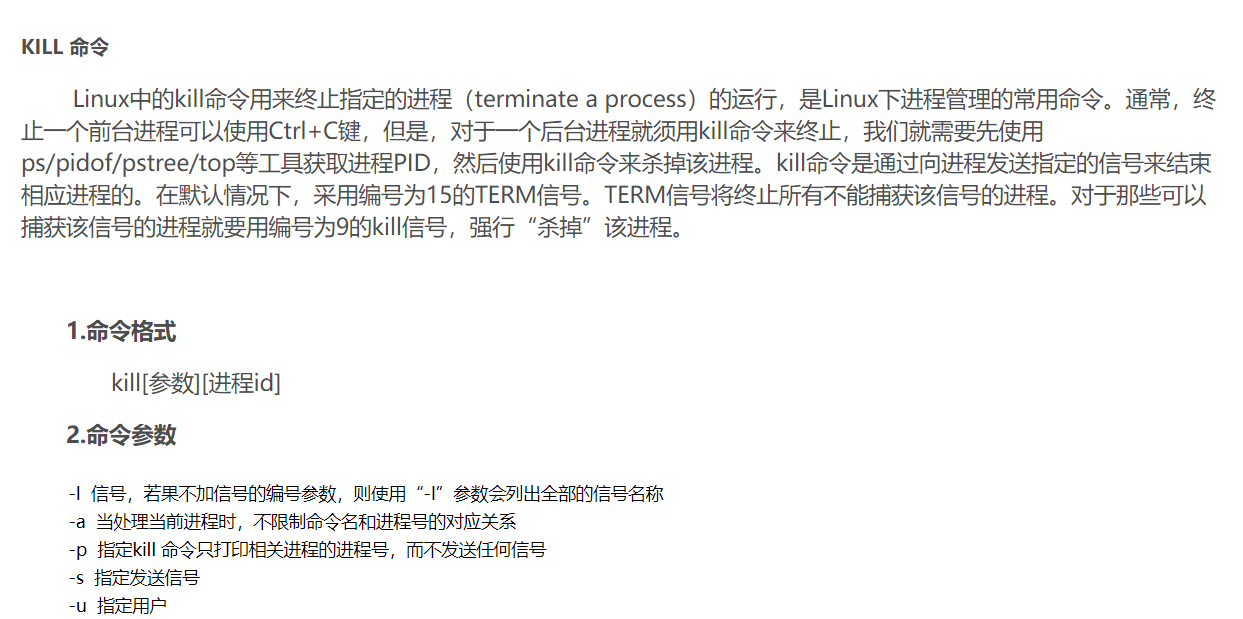
awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。



# killall kill 命令

Linux系统中的killall命令用于杀死指定名字的进程（kill processes by name）。我们可以使用kill命令杀死指定进程PID的进程，如果要找到我们需要杀死的进程，我们还需要在之前使用ps等命令再配合grep来查找进程，而killall把这两个过程合二为一，是一个很好用的命令。





# Make 命令

参考：https://www.cnblogs.com/mazhenyu/p/11069688.html

一、make命令  
make命令后接参数，称为目标；

1 常见目标  
make all：编译所有目标

make install：安装已编译的程序

make uninstall：卸载已安装的程序

make clean：删除由make命令产生的文件，通常删除目标文件.o

make distclean：删除由./configure产生的文件

make check：测试刚编译的软件

make installcheck：检查安装的库和程序

make dist：重新打包成packname-version.tar.gz

执行make命令时，需要一个Makefile文件，以告诉make命令如何编译和链接程序；

2 参数  
-B：重新建立所有目标

-d：打印调试信息

-C：切换到指定路径下寻找Makefile

-f：将指定文件看做Makefile

-j：同时运行命令的个数，即多线程执行Makefile，后接的个数可由nproc命令返回值来指定

注：nproc命令打印当前进程可用的处理数（线程数）；

二、程序的编译和链接  
一般在C/C++开发中，首先将源文件编译成目标文件（Windows下.obj文件，Unix下.o文件）——编译compile，再将目标文件合成执行文件——链接link；

三、make命令如何工作？  
1 make在当前目录下寻找“Makefile”或“makefile”文件

2 若找到，查找文件中的第一个目标文件.o

3 若目标文件不存在，根据依赖关系查找.s文件

4 若.s文件不存在，根据依赖关系查找.i文件

5 若.i文件不存在，根据依赖关系查找.c文件，此时.c文件一定存在，于是生成一个.o文件，再去执行

# Strip 命令

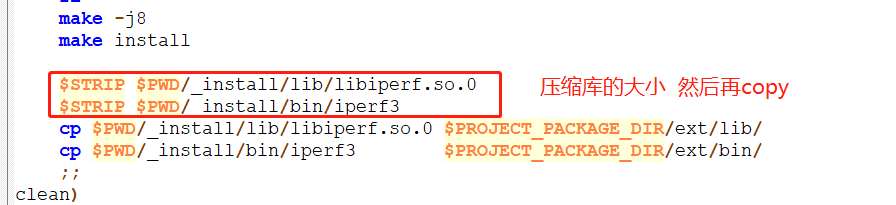
 strip用于脱掉文件的衣服， 文件会变小， 其中的符号信息会失去。 那这个strip有什么用呢？ 很有用的！ 原来的a.out比较大， 可以执行。 在strip之后， 文件变小了， 仍然可以执行， 这就就节省了很多空间。

 其实， strip不仅仅可以针对可执行文件， 还能针对目标文件和动态库等。

 在实际的开发中， 经常需要对动态库.so进行strip操作， 减少占地空间。 而在调试的时候（比如用addr2line）， 就需要符号了。 因此， 通常的做法是： strip前的库用来调试， strip后的库用来实际发布， 他们两者有对应关系。 一旦发布的strip后的库出了问题， 就可以找对应的未strip的库来定位。

 最后啰嗦一句， 某某动态库strip前是18M左右， strip后是3M左右， 可见， 脱脱衣服还是有明显好处的。

 补充： 后来发现， 在调试过程中， 经常涉及到传库， 库太大时， 很耗费传输时间， 所以还是用strip来搞一下吧。



# dd 命令：

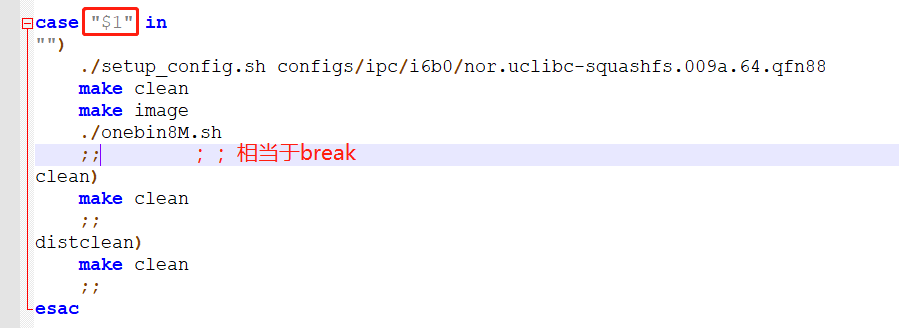
1.  dd if=infile.log  of=outfile.log bs=1M count=1

  拷贝 infile.log内容 到 outfile.log. 一次读写比特数为1M , 每次读写一个块（1M，由bs决定）

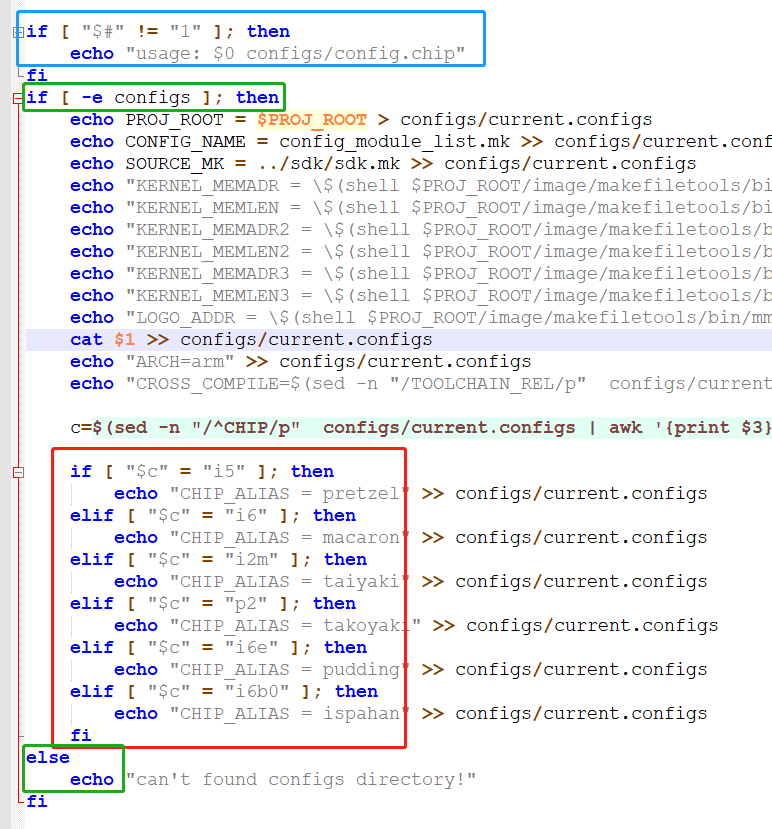


# case esac

1. ;;相当于其它语言中的break
2. 每个pattern之后记得加)
3. 最后记得加esac (即反的case)



# If else



# While



# Killall -0

