

```

1  写一个脚本完成一下任务 (line3245.sh)
2  说明：此脚本能于同一个repo文件中创建多个Yum源的指向
3  1、接受一个文件名作为参数，此文件存放至/etc/yum.repos.d目录中，且文件名以.repo为后缀：
4      要求，此文件不能事先存，否则，报错
5  2、在脚本中，提醒用户输入repo id:如果为quit，则退出脚本；否则，继续完成一下步骤：
6  3、repo name以及baseurl的路径，而后以repo文件的格式将其保存至指定的文件中：
7  4、enabled默认为1，而gpgckeck默认设定为0；
8  5、此脚本会循环执行多次，除非用户为 repo id指定为quit
9
10 #!/bin/bash
11 #by zhangshuo 20180126
12
13 if [ $# -eq 0 ];then
14     echo "usage !!!!!!"
15     exit 2
16 fi
17
18 REPONAME=$1
19 REPOPATH=/etc/yum.repos.d/
20
21
22 if [ -e $REPOPATH${REPONAME}.repo ];then
23     echo "the file is exisit"
24     exit 3
25 fi
26
27 read -p "input your repoid : " REPOID
28
29 until [ $REPOID == 'quit' ];do
30     echo "[$REPOID]" >> $REPOPATH${REPONAME}.repo
31     read -p "repo name: " REPOIDNAME
32     echo "name=$REPOIDNAME" >> $REPOPATH${REPONAME}.repo
33     read -p "baseurl: " BASEURL
34     echo "baseurl=$BASEURL" >> $REPOPATH${REPONAME}.repo
35     echo "enabled=1" >> $REPOPATH${REPONAME}.repo
36     echo "gpgcheck=0" >> $REPOPATH${REPONAME}.repo
37
38     read -p "input your repoid : " REPOID
39 done
40
41
42
43 While ,until, for
44
45 提前结束循环
46 break: 退出循环
47 continue: 退出本次循环
48
49 #!/bin/bash
50 #by zhangshuo 20180126
51
52 declare -i SUM=0
53 let I=0
54
55 while [ $I -lt 100 ];do
56     let I++
57     if [ ${I%2} -eq 0 ];then
58         continue
59     fi
60     let SUM+=I
61     if [ $SUM -gt 1000 ];then
62         break
63     fi
64 done
65
66 echo $I
67
68 echo $SUM
69
70 While的特殊用法一：相当于c语言的while (1)
71 while : ; do
72
73

```

```

74
75 done
76
77 #!/bin/bash
78 #by zhangshuo 20180126
79
80 read -p "input a user name : " USERNAME
81
82 while ;;do
83     if who | grep "$USERNAME" &> /dev/null ;then
84         echo "$USERNAME is online"
85         break
86     else
87         echo "waiting $USERNAME"
88         sleep 3
89     fi
90 done
91
92
93 While的特殊用法二:
94 while read line;do
95
96
97
98
99 done < /path/to/somefile
100
101 #!/bin/bash
102 #by zhangshuo 20180126
103
104 while read LINE;do
105     [ $(echo $LINE | awk -F : '{print $3}') -lt 500 ] && continue
106     [ $(echo $LINE | awk -F : '{print $7}') == '/bin/bash' ] && echo -e "`awk -F :
    '{print $1,$3}'`"
107 done < /etc/passwd
108
109

```

110 写一个脚本

- 111 1、判断一个指定的bash脚本是否有语法错误：如果有错误，则提醒用户键入q或者Q无视错误并退出，其它任何键可以通过vim打开这个指定的脚本；
- 112 2、如果用户通过vim打开编辑后保存退出时仍然有错误，则重复第一步中的内容；否则，就正常关闭退出。

```

113
114 Until bash -n $1 &> /dev/null;do
115     read -p "syntax error,[Qq] to quit, others for editing:" CHOICE
116     case $CHOICE in
117         q|Q)
118             echo "something wrong,quiting"
119             exit 5
120             ;;
121         *)
122             vim + $1
123             ;;
124     esac
125 done
126
127

```

130 脚本编程之函数

132 函数：功能，function 和c语言基本类似

134 函数的主要功能是实现代码重用：

136 库：.so也是直接被调用的和函数目的类似，实现代码共用

138 函数定义的方法有两种：

```

139
140 function funcname{
141     command
142 }
143

```

```

144 funcname() {
145     command
146 }
147
148 如果要取得函数执行返回值则只需在函数中使用echo输出最后结果，在主调函数中使用命令替换即可
149
150 如果要返回执行状态结果，则需要使用return返回程序执行状态
151
152 自定义执行状态返回值：
153 使用return #
154 0-255
155
156 #!/bin/bash
157 #by zhangshuo 20180127
158 #实现函数参数传递，及其函数执行结果的引用
159
160 TWOSUM() {
161     echo ${1+$2}
162 }
163
164 TWOSUM 5 6
165
166 for I in `seq 1 1 10`;do
167     J=${I+1}
168     echo "$I plus $J is `TWOSUM $I $J`"
169 done
170
171
172 #!/bin/bash
173 #by zhangshuo 20180127
174 #通过函数实现参数的传递，函数的执行状态返回
175 ADDUSER() {
176     USERNAME=$1
177     if id $USERNAME &> /dev/null;then
178         return 1
179     else
180         echo "add user $USERNAME"
181     fi
182 }
183
184 ADDUSER $1
185 if [ $? -eq 0 ];then
186     echo "finish"
187 else
188     echo "failuer"
189 fi
190
191 #!/bin/bash
192 #by zhangshuo 20180127
193 #使用while循环来实现函数的多次调用
194
195 ADDUSER() {
196     USERNAME=$1
197     if id $USERNAME &> /dev/null;then
198         return 1
199     fi
200 }
201
202 for I in {1..10};do
203
204     ADDUSER user$I
205     if [ $? -eq 0 ];then
206         echo "user$I finish"
207     else
208         echo "user$I failuer"
209     fi
210 done
211
212 练习：写一个脚本，判定192.168.0.200-192.168.0.254之间的主机哪些在线。要求：
213 1、使用函数来实现一台主机的判定过程；
214 2、在主程序中来调用此函数判定指定范围内的所有主机的在线情况。
215

```

```

216 #!/bin/bash
217 #by zhangshuo 20180127
218
219 ONLINETEST(){
220     if ping -c 1 -W 1 $1 &> /dev/null;then
221         return 0
222     else
223         return 1
224     fi
225 }
226
227 for I in {1..254};do
228
229     ONLINETEST 192.168.1.$I
230
231     if [ $? -eq 0 ];then
232         echo "192.168.1.$I online"
233     #else
234     #     echo "192.168.1.$I no online"
235     fi
236 done

```

练习：

写一个脚本：使用函数完成

1、函数能接受一个参数，参数为用户名

判断一个用户是否存在

如果存在，就返回此用户的shell和uid；并返回正常状态值：

如果不存在，就说此用户不存在；并返回错误状态值

2、在主程序中调用函数

扩展1:在主程序中，让用户自己输入用户名后，传递给函数来进行判断：

扩展2:在主程序中，输入用户名判断后不退出脚本，而是提示用户继续输入下一个用户名：如果用户输入的用户不存在，请用户重新输入，但如果用户输入的是q或Q就退出；

```

248
249 #!/bin/bash
250 #by zhangshuo 20180127
251
252 USERTEST(){
253     if id $1 &> /dev/null;then
254         echo $(cat /etc/passwd | grep "$1" | awk -F : '{print $3,$7}')
255         return 0
256     else
257         echo "no such user"
258         return 1
259     fi
260 }
261
262
263 read -p "input user name: " USERNAME
264
265 until [ $USERNAME == 'Q' -o $USERNAME == 'q' ];do
266     USERTEST $USERNAME
267     read -p "input user name: " USERNAME
268 done

```

进程的相关管理

内存的占用大小有

Vsz：虚拟的内存大小使用

Rss：表示常驻内存集大小：residentsize

每个任务都会有6个状态：停止、就绪、正在执行、可中断睡眠状态、不可中断睡眠状态、僵尸状态

linux中有140个优先级0-139，数字越小优先级越高

其中100-139是由用户可控制的

0-99：为内核控制，

对于CPU选择较高优先级然后进行调度的时间的标准为大O标准

O：

O(1)：常函数

O(N)：一次函数

287 0 (LOGN) : 、 、 、
288 0 (N^2) : 、 、 、
289 0 (2^N) : 、 、 、

290

291

292 优先级越高的程序

293 1、获得更多的cpu运行时间

294 2、更优先获得运行的机会

295

296 用户可调节的优先级为100-139

297

298 nice值: 优雅的, 友好的

299 nice值的取值为-20到19 默认nice值都为0

300

301 普通用户仅能调大自己的nice值

302

303 init: 进程号为1

304

305 ps: process state

306 该命令有两种不同的风格

307 systemv风格: -

308 BSD风格:

309

310 a: 显示所有跟终端相关的进程

311 u: 显示是由那个用户启动的

312 x: 所有与终端无关的进程

313

314 systemv风格

315

316 ps

317 -e: 显示所有线程

318 -l: 长格式

319 -F:

320

321

322 进程的分类

323 跟终端相关的进程:

324 跟终端无关的进程:

325

326 进程状态:

327 D: 不可中断的睡眠

328 R: 运行或就绪

329 S: 可中断的睡眠

330 T: 停止

331 Z: 僵尸状态

332

333 <: 高优先级进程

334 N: 低优先级进程

335 +: 前台进程组中的进程

336 l: 多线程进程

337 s: 会话进程首进程

338

339 Pstree:显示进程树

340 Pgrep: 类似grep只是利用正则表达式寻找进程

341 -u : 属主

342

343 Pidof: 根据进程名查看进程的进程号

344 pidof init

345

346 Top:

347 top - 16:32:19 up 54 min, 2 users, load average: 0.00 (1分钟), 0.00 (5分钟),

348 0.00 (15分钟)

349 Tasks: 95 total, 2 running, 93 sleeping, 0 stopped, 0 zombie

348 Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi (硬中断),

349 0.0%si (软中断), 0.0%st (被偷走的)

350 Mem: 1026064k total, 396016k used, 630048k free, 19580k buffers

351 Swap: 2064376k total, 0k used, 2064376k free, 249444k cached

352

353 M:根据驻留内存大小进行排序

354 P:根据cpu使用百分比进行排序

355 T:根据累计时间进行排序

356

357 l:是否显示平均负载和启动时间

358 t:是否显示进程和cpu状态相关信息

```

358 m:是否显示内存相关信息
359
360 c:是否显示完整的命令行信息
361 q:退出top
362 k:终止某个进程
363
364 l:查看多个cpu情况
365
366 Top -d 1:规定刷新时常
367 Top -d 1 -b: 以批模式进行刷新
368 Top -d 1 -b -n 3:在批模式下公显示多少批
369
370 进程间通信机制:
371 共享内存
372 信号:
373 旗语:
374
375
376 Kill -l: 显示所有可用信号
377 重要的信号:
378 1:SIGUP 让进程不用重启, 就可以重读其配置文件, 并让新的配置文件信息生效
379 2:SIGINT ctrl+c 中断一个进程
380 9:SIGKILL 杀死一个进程 直接立刻杀死一个进程, 强行杀死
381 15:SIGTERM 终止一个进程 通知进程马上要关闭, 请结束当前所有任务 默认使用此信号
382
383 指定一个信号:
384 信号号码: kill -l
385 信号名称: kill -SIGKILL
386 信号名称简写: kill -KILL
387
388 Kill pid: 直接杀死一个进程
389 Killall command: 杀死一组进程, 比如httpd进程会有很多进程, 可以使用此命令一起杀死
390
391 调整进程的nice值既调整优先级
392 1、调整已经启动的进程的nice值
393 renice ni pid
394 2、在启动时指定nice值
395 nice -n ni command
396
397 前台作业: 占据了命令提示符
398 后台作业: 启动之后, 释放命令提示符, 后续的操作在后台完成
399
400 前台进程送到后台:
401 ctrl+z: 把正在前台的作业送往后台
402 command &: 让命令在后台执行
403
404 Bg: 让后台的停止作业继续运行
405 bg jobid
406
407 Jobs: 查看后台的所有作业
408 作业号, 不同于进程号
409 +: 命令将默认操作的作业
410 -: 命令将第二个默认操作的作业
411
412 Fg: 将后台的作业调回前台
413 fg 【%jobid】
414 kill %jobid: 终止某作业
415
416 Vmstat: 系统状态查看命令
417
418 procs -----memory----- ---swap-- -----io----- --system-- -----cpu-----
419 r (运行队列长度) b (阻塞队列长度) swpd free buff cache si so
419 bi (磁盘块读入) bo (磁盘块读出) in (中断的次数) cs (上下文切换的次数)
419 us ( ) sy id wa st
420 0 0 0 368296 88068 388812 0 0 81 20 1029 45 0 0 99 0 0
421
422 procs -----memory----- ---swap-- -----io----- --system-- -----cpu-----
423 r b swpd free buff cache si so bi bo in cs us sy id wa st
424 0 0 0 368296 88068 388812 0 0 81 20 1029 45 0 0 99 0 0
425
426 Vmstat 1 5 : 每隔一秒钟显示一次但是只显示五次
427
428 uptime : 显示top的第一个字段

```

429
430
431 Linux系统启动流程
432
433 Post-->bios (boot sequence)-->MBR (bootloader, 446byte)-->Kernel-->initrd-->/sbin/init
434
435 Windows的启动会有几个模式，比如安全模式，其实就是不同模式所启动的服务程序不相同而已
436
437 同理linux下也有这样的模式，但是叫做运行级别：启动的服务不同对应的级别不同
438 运行级别：0-6
439 0:halt关机
440 1:单用户模式，类似windows的安全模式，直接以管理员身份切入
441 2:multi user mode，但是no nfs不启用网络文件系统
442 3:正常多用户模式，只有命令行不启用图形界面
443 4:reserved保留模式
444 5:正常多用户模式，但是会启动图形中断
445 6:reboot重启
446 这些运行级别是在用户空间所实现的所以由init进程所控制，各种运行状态取决于其配置文件 (/etc/inittab)
447
448 内核设计风格：
449 核心会动态加载对应设备的内核模块，初始化时只是初始化的核心本身
450 内核模块在：/lib/modules/"内核版本号命名的目录"
451 单内核：linux (lwp叫做轻量级线程)
452 模块化设计：核心与ko (kernel object)
453 微内核：windows, solaris (支持线程)
454
455 Linux内核的命名
456 Vmlinux-2.6.18
457
458 红帽5上实现根切换bash不能实现，所以要借助其他bash
459 所以使用：nash
460 switch—root命令完成根切换
461
462 内核启动时要借助中间设备完成驱动设备和文件系统的加载
463 这个文件系统是在内存中直接模拟的文件系统，即把内存当硬盘使用
464 叫
465 红帽5这个中间设备叫做：ramdisk-->这个中间文件系统的名字叫做：initrd
466 红帽6叫做ramfs (ubuntu也是用的这个文件系统)：-->initramfs
467
468 Chroot:chroot /path/to/temproot [command...]
469 chroot /root/myroot /bin/bash
470
471 Ldd /path/to/binary_file:显示二进制文件所依赖的共享库
472
473
474
475 详解启动过程
476 bootloader (mbr)
477 Linux的引导程序
478 1、lilo: linux loader 但是不能识别8g以后的分区
479 2、grub: grand unified bootloader: 统一的引导程序
480 因为mbr存储有限所以grub分三段
481 stage1:mbr
482 stage1.5:识别常见的文件系统
483 stage2: 位于/boot/grub
484
485 grub的配置文件/etc/grub.conf 该链接指向 /boot/grub/grub.conf
486
487 grub.conf中的信息
488
489 default=0:设定默认启动的title的编号，从0开始的内核
490 timeout=5: 等待用户选择的超时时常，单位是秒
491 splashimage=(hd0,0)/grub/splash.xpm.gz: 指定grub的背景图片
492 hiddenmenu: 隐藏菜单
493 password redhat
494 Password --md5 \$1\$2hcCi/\$s2E/UkTHpiUQhile1xoU3/ 该作用为编辑grub时需要密码
495 title CentOS (2.6.18-398.el5): 内核标题或操作系统的名称，纯字符串，可以任意修改
496 root
(hd0,0): 内核文件所在的设备，对gurb而言，所有类型硬盘一律hd: hd#表示第几个磁盘：最后的0表示对应磁盘的分区
497 kernel /vmlinuz-2.6.18-398.el5 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
指定内核文件路径，及传递给内核的参数

```
498     initrd /initrd-2.6.18-398.el5.img ramdisk文件路径
499     Password --md5 $1$2hcCi/$s2E/UkTHpiUQhile1xoU3/ 该位置为如果要启动该内核则需要密码
500
501 Cat /proc/cmdline中显示的和ro root=/dev/VolGroup00/LogVol00 rhgb quiet是一样的
502
503 grub-md5-crypt:使用该命令创建grub密码串
504
505 查看当前运行级别:
506 runlevel
507     N (上一次的级别) 3 (当前级别)
508
509 who -r
510
511 查看内核release号
512     uname -r
513
514 安装grub stage1
515 # grub 打开grub的命令行界面 一定要确定自己的磁盘号
516 grub> root (hd0,0) 指定内核所在的分区
517 grub> setup (hd0) 安装grub到指定磁盘
518
519 安装grub第二种方式:
520 # grub-install --root-directory=/ grub所在的目录即/boot的父目录
521
522 磁盘挂载到的目录下要挂在在boot目录下 mount /dev/sdb1 /media/boot/
523
524 grub-install --root-directory=/media /dev/sdb
525
526 安装完成后在grub目录下创建配置文件: grub.conf
527
528 如果grub配置文件损坏, 开机时会弹出grub的命令窗口, 此时只需要指定root和内核路径即可开机
529 如果不知道配置文件在哪里可以使用find (hd0,0) 去查找
530 Find (hdl,0)
531 grub>root (hd0,0)
532 grub>kernel /path/to/kernel_filename
533 grub>initrd /path/to/initrd_filename
534
535 Kernel初始化的过程:
536 1、设备探测
537 2、驱动初始化 (可能会从initrd (红帽六initramfs) 文件中装载驱动模块)
538 3、以只读方式挂载根文件系统:
539 4、装载第一个进程init (pid为1)
540
541 /sbin/init: (/etc/inittab)
542     upstart: ubuntu
543     systemd:
544
545 Inittab文件中的格式如下
546 每一行定义了一个要运行的服务或事进程
547
548 id:runlevels:action:process
549
550 Id:标识符
551 Runlevels: 在哪个级别运行此程序
552 Action: 要执行的动作
553 process: 要执行的进程
554
555 action:
556 initdefault: 设定默认运行级别
557 sysinit: 系统初始化
558 Wait: 等待级别切换至此级别时执行
559 respawn: 一旦程序终止, 会重新启动
560
561 /etc/rc.d/rc.sysinit系统初始化脚本完成的任务
562 1、激活udev和selinux
563 2、根据/etc/sysctl.conf文件, 来设定内核参数
564 3、设定系统时钟
565 4、装载键映射
566 5、启动交换分区
567 6、设置主机名
568 7、根文件系统检测, 并以读写方式重新挂载
569 8、激活raid和lvm设备
570 9、启动磁盘配额
```


571 10、根据/etc/fstab，检查并挂载其他文件系统
572 11、清理过期的锁和pid文件；
573
574 服务类脚本
575 /etc/rc.d/rc文件
576 根据运行级别启动相应的服务，该脚本的任务是按运行级别启动以s开头的文件，并关闭以k开头的文件。
577
578 /etc/rc.d/rc#.d
579 表示的为对应级别需要启动或关闭的服务，在这种文件夹下数字越小表示优先级越高
580
581 这种服务类脚本既/etc/rc.d/init.d文件中的脚本都应该能接受以下的参数
582 改类脚本都为sysv：风格
583 start|stop|restart|status
584 reload|configtest
585 这类脚本中每个脚本都有以下两行，：
586 Chkconfig：改命令用于自动创建以k或s开头的文件
587 Description：
588
589 # chkconfig: 345（在哪些级别下默认启动的） 26（ss启动的优先次序）
74（kk关闭的优先次序）
590
当chkconfig命令来为此脚本在rc#.d目录创建链接时，runlevels表示默认创建为s*开头的链接，除此之外的级别默认创建为k开头的链接：
591 s后面的启动优先级为ss所表示的数字：k后面关闭优先次序为kk所表示的数字：
592 # description: Listen and dispatch ACPI events from the kernel
用于说明此脚本的简单功能：\,续行
593
594 Chkconfig --list:列出当前系统的所有服务 查看所有独立守护服务的启动设定：
595 chkconfig --list service_name
596 自动创建链接的方法
597 Chkconfig --add service_name
598 Chkconfig --del service_name
599 Chkconfig --level 24 myservice {off|on}
这样可以实现不用修改脚本而使系统在改级别下系统服务关闭
600
601 如果省略级别指定，默认为2345级别
602
603 /etc/rc.d/rc.local:系统最后启动的一个服务，准确说，应该执行的一个脚本
604
605 /etc/inittab的任务
606 1、设定默认运行级别
607 2、运行系统初始化脚本
608 3、运行指定运行级别对应的目录下脚本
609 4、设定ctrl+atl+del组合键的操作
610 5、定义ups电源在电源故障/恢复时执行的操作
611 6、启动虚拟终端（2345）
612 7、启动图形终端（5）
613
614 守护进程的类型
615 独立守护进程
616 xinetd: 超级守护进程
617 yum install xinetd -y:安装xinetd程序
618 service xinetd start:启动超级守护进程
619 chkconfig --list:查看改超级守护进程
620 瞬时守护进程：不需要关联至运行级别
621
622
623 核心：/boot/vmlinuz-version
624 内核模块（ko）：/lib/modules/version/
625
626 内核设计：
627 单内核
628 模块化设计
629 微内核
630 装载模块的命令
631 insmod
632 modprobe
633
634 用户空间访问、监控内核的方式：
635 通过修改系统映射文件来和内核进行交互/proc,/sys
636
637 伪文件系统

```
638
639 /proc/sys: 此目录中的文件很多都是可读写的
640 /sys/:某些文件可写
641
642 清除系统的缓存
643 Echo 1 > /proc/sys/vm/drop_caches
644
645 设定内核参数值的方法:
646 Echo value > /proc/sys/to/somefile
647 Sysctl -w kernel.hostname=centos5
648 Sysctl -w vm.drop_caches=1
649
650 上面的方式能立即生效, 但无法永久有效:
651
652 永久有效的方法但不会立即生效: /etc/sysctl.conf
653
654 如果修改完/etc/sysctl.conf需要重新读取配置文件的话可以使用
655 Sysctl -p 来通知内核重读改配置文件
656
657 如果系统中有两块网卡, 期望一个网卡读取数据一个网卡发送数据可以修改下面的文件为1
658 /proc/sys/net/ipv4/ip_forward
659
660 Sysctl -a 显示所有内核参数及其值
661
662 内核模块管理
663 Lsmod:查看模块
664
665 modprobe modname: 装载某个模块
666 modprobe -r modename : 卸载某个模块
667
668 Modinfo modename:查看模块的详细信息
669
670 Insmod /path/to/module_file:装载模块
671 rmmod modename: 移除模块
672
673 depmod /path/to/modles_dir 生成模块依赖关系
674
675 内核中的功能除了核心功能之外, 在编译时, 大多功能都有三种选择:
676 1、不使用此功能
677 2、编译成内核模块
678 3、编译进内核
679
680 如何手动编译内核
681 1、make gconfig:gnome桌面环境下使用, 需要安装图形开发库:GNOME Software Development
682 2、make kconfig:kde桌面环境使用
683
684 Make meunconfig
685
686 Make
687 Make modules_install
688 Make install
689
690 Screen命令:
691 Screen -ls: 显示已经建立的屏幕
692 Screen: 直接打开一个新的屏幕
693 ctrl+a, d: 拆除一个屏幕
694
695 screen -r id: 还原回某屏幕
696
697 exit: 退出
698
699 二次编译时清理:
700 make clean
701 make mrproper
702
703 Grub-->kernel-->initrd-->rootfs(/sbin/init,/bin/bash)
704
705 搭建自己的linux系统
706 1、安装grub
707 2、复制内核文件即cp /boot/vmlinuz-2.6.18-398.el5 /media/boot/vmlinuz
708 3、创建initrd文件
709 mkinitrd /boot/initrd-`uname -r`.img `uname -r`
710
```

```

711 cpio -id < ../initrd-2.6.18-398.el5.img
712 修改init文件
713 find . | cpio -H newc --quiet -o | gzip -9 > /media/boot/initrd.gz
714
715 Bash自带截取字符串的功能
716
717 FILE=/usr/local/src
718 ${FILE#*/}:usr/local/src
719 ${FILE##*/}:src
720
721 ${FILE%/*}:/usr/local
722 ${FILE%%/*}:
723
724 ${parameter#*word}
725 ${parameter##*word}
726     The word is expanded to produce a pattern just as in pathname
        expansion. If the pattern matches the beginning of the value of
        parameter, then the
727     result of the expansion is the expanded value of parameter with the
        shortest matching pattern (the '#' case) or the longest matching
        pattern (the
728     '##' case) deleted. If parameter is @ or *, the pattern removal
        operation is applied to each positional parameter in turn, and the
        expansion is
729     the resultant list. If parameter is an array variable subscripted
        with @ or *, the pattern removal operation is applied to each member
        of the array
730     in turn, and the expansion is the resultant list.
731
732 ${parameter%word*}
733 ${parameter%%word*}
734     The word is expanded to produce a pattern just as in pathname
        expansion. If the pattern matches a trailing portion of the expanded
        value of paramete-
735     ter, then the result of the expansion is the expanded value of
        parameter with the shortest matching pattern (the '%' case) or the
        longest matching
736     pattern (the '%%' case) deleted. If parameter is @ or *, the
        pattern removal operation is applied to each positional parameter in
        turn, and the
737     expansion is the resultant list. If parameter is an array variable
        subscripted with @ or *, the pattern removal operation is applied to
        each member
738     of the array in turn, and the expansion is the resultant list.
739
740 #!/bin/bash
741 #by redhat 20180306
742
743 DEST=/media/sysroot
744 libcp(){
745     LIBPATH=${1%/*}
746     [ ! -d $DEST$LIBPATH ] && mkdir -p $DEST$LIBPATH
747     [ ! -e $DEST${1} ] && cp $1 $DEST$LIBPATH && echo "cp $1 finished"
748 }
749
750 bincp(){
751     CMDPATH=${1%/*}
752     [ ! -d $DEST$CMDPATH ] && mkdir -p $DEST$CMDPATH
753     [ ! -e $DEST${1} ] && cp $1 $DEST$CMDPATH
754
755     for LIB in `ldd $1 | grep -o "\\.lib\\(64\\)\\{0,1\\}/[^[:space:]]\\{1,\\}"`;do
756         libcp $LIB
757     done
758 }
759
760 read -p "Your Command:" CMD
761 until [ $CMD == 'q' ];do
762     ! which $CMD &> /dev/null && echo "wrong command" && read -p "input againe"
        CMD && continue
763     COMMAND=`which $CMD | grep -v "alias" | grep -o "[^[:space:]]\\{1,\\}"`
764     bincp $COMMAND
765     echo "copy $COMMAND finished"
766     read -p "continue: " CMD

```

767 done
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788