一些危险的命令（核武器），请谨慎使用

No 1. rm -rf /

[root@localhost ~]# rm -rf /

rm: it is dangerous to operate recursively on ‘/’

rm: use --no-preserve-root to override this failsafe

No2. wget http://某个不信任的源 -O- | sh

为图省事，程序员会直接执行下载的脚本。这是十分危险的，当这些脚本来自非可信任的源，那么这种做法无异于在刀剑上跳舞。这段脚本可能会带有类似rm -rf /的破坏指令，也可能会窃取你本机的隐私信息。

No3. 任意命令 > /dev/sda

这里的任意命令可以是如ls,top等带有输出的指令，这些输出的内容会直接写到/dev/sda磁盘的初始扇区，而这些扇区通常存的是引导系统和文件系统的元数据，直接将它们覆盖，系统也就基本启动不了。

[root@localhost /]# ls > /dev/sda

[root@localhost /]#

No4. mkfs.ext3 /dev/sda

Linux下的格式化磁盘

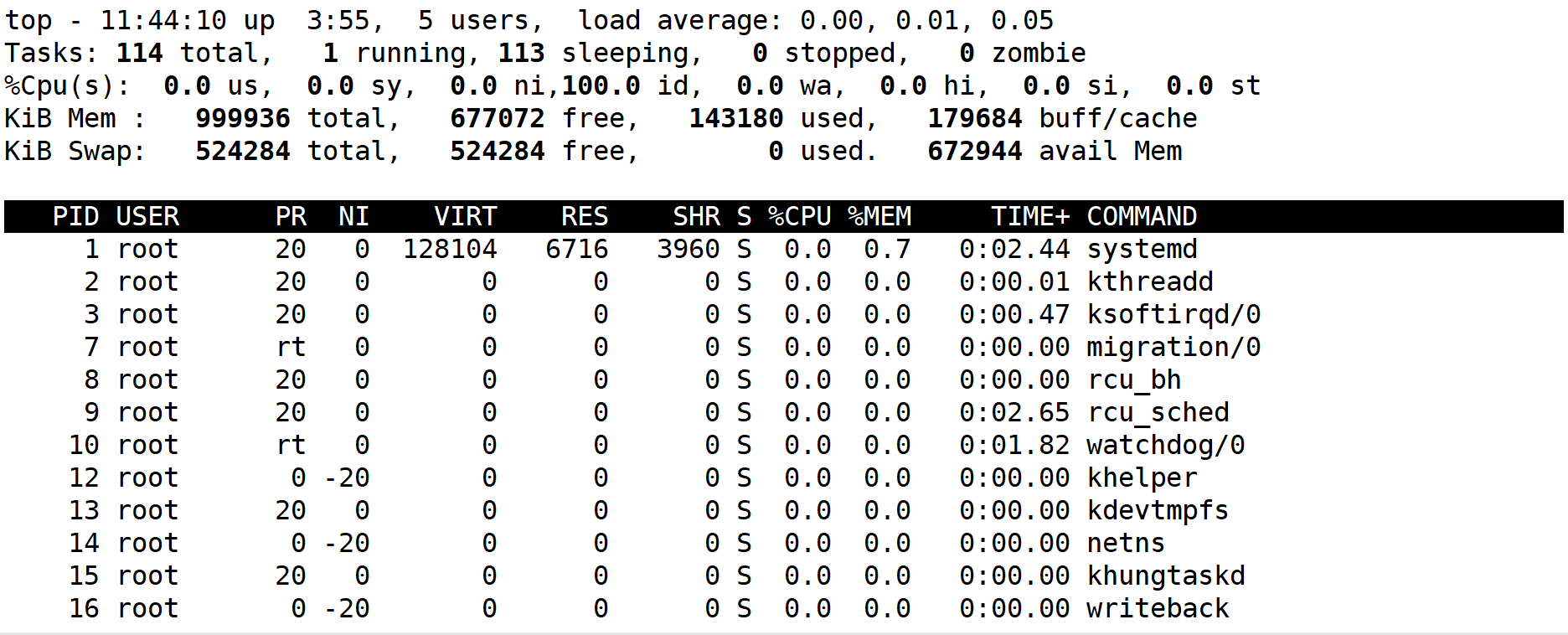
~~No5. mv /home/yourhomedirectory/\* /dev/null 致命指数：\*\*\*~~

~~Unix/Linux的/dev/null是一个特殊的设备文件，写入/dev/null设备文件的数据会被自动丢弃。~~

NO6. > file

清空文件里面的内容

top命令解释：按q退出。



其中第一行：

11:40:10 当前时间

3:55 系统运行时间，格式为时:分

5 user 当前登录用户数

load average: 0.06, 0.60, 0.48

系统负载，即任务队列的平均长度。 三个数值分别为 1分钟、5分钟、15分钟前到现在的平均值。

CPU状态信息：

Cpu(s)表示的是cpu信息。各个值的意思是：

us: user cpu time (or) % CPU time spent in user space

sy: system cpu time (or) % CPU time spent in kernel space

ni: user nice cpu time (or) % CPU time spent on low priority processes

id: idle cpu time (or) % CPU time spent idle

wa: io wait cpu time (or) % CPU time spent in wait (on disk)

hi: hardware irq (or) % CPU time spent servicing/handling hardware interrupts

si: software irq (or) % CPU time spent servicing/handling software interrupts

st: steal time - - % CPU time in involuntary wait by virtual cpu while hypervisor is servicing another processor (or) % CPU time stolen from a virtual machine

翻译一下：

us：用户态使用的cpu时间比

sy：系统态使用的cpu时间比

ni：用做nice加权的进程分配的用户态cpu时间比

id：空闲的cpu时间比

wa：cpu等待磁盘写入完成时间

hi：硬中断消耗时间

si：软中断消耗时间

st：虚拟机偷取时间

PID：进程id

USER：进程所有者的用户名

PR：优先级

NI：nice值。负值表示高优先级，正值表示低优先级

VIRT:进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES

SWAP：进程使用的虚拟内存中，被换出的大小，单位kb。

RES:进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA

SHR:共享内存大小，单位kb

%MEM：进程使用的物理内存百分比

%CPU：上次更新到现在的CPU时间占用百分比

TIME+：进程使用的CPU时间总计，单位1/100秒

S：进程状态。

D=不可中断的睡眠状态

R=运行

S=睡眠

T=跟踪/停止

Z=僵尸进程

COMMAND：命令名/命令行

Echo使用过程中，单引号’’和双引号””是有区别，单引号屏蔽一切特殊符号，双引号没有屏蔽$符号。

Shell编程语法：

1.脚本基本格式

#!/bin/bash

# 注释内容

Shell编程语句

2.脚本执行方式

(1)chmod a+x xx.sh

(2) bash xxx.sh

(3)source xxx.sh

(4) .xxx.sh

3.shell程序结构

(1)顺序结构

(2)选择结构

(3)循环结构

任何一种复杂的程序都是由这三个基本结构叠加组合而来

程序=算法+数据结构：数据结构用于描述一个对象，算法教对象如何做事。

1. 选择结构注意事项

if 和fi成对出现

if后面紧跟then

if后面的[]中的条件和[]之间要有空格，比如if [ xxxx ]

case语句中每一个模式（匹配）的结束 标志“;;”

case语句中默认匹配模式 \*)，case语句最后采用easc作为结束标志

5.循环结构注意事项

for语句中采用do…done包含循环体

For语句别忘记(())，双括号

For语句可以“离散”控制执行也可以“连续”控制执行

While语句还可以用于“读取一行，处理一行”

Until语句中，条件满足，程序停止循环

循环语句中，注意continue和break区别，continue结束当前循环，进入下一次循环；break语句直接跳出循环体，并结束循环。

1. 函数的使用

模块化设计思想。

实例中结合先前所学内容：基本命令、重定向、管道、序列号、AWK/SED、变量使用，循环/选择/顺序结构。

Shell函数传递参数。（通过位置变量进行）

可以参考源码（书本《运维之道》第2版）：

菜单实例 P157

位置变量实例 P111

Shell脚本编程实例（一）：

#!/bin/bash

#Date:2018-03-25

#Version:0.01

#Author:ZhangPeng

echo "this is the first shell script"

echo "generate files"

for name in {1..100}.txt

do

touch $name

done

echo "analyze documentation information"

file\_num=`ls -la | grep "^-"|wc -l`

echo "the num of file is $file\_num"

folder\_num=`ls -la | grep "^d"|wc -l`

echo "the num of folder is $folder\_num"

docu\_size=`du -sh . | awk '{print $1}'`

echo "the size of documentation is $docu\_size"

echo "backup files start..."

backup\_date=`date | awk '{print $6$2$3}'`

backup\_path=$backup\_date$RANDOM

backup\_name=$backup\_path+tar.gz

mkdir /$backup\_path

cp -r . /$backup\_path

cd /

tar -czf $backup\_name /$backup\_path

echo "backip files end"

exit 0

Shell脚本编程实例（二）：

#!/bin/bash

#Date:2018-03-25

#Version:0.01

#Author:ZhangPeng

# this program is used to manage linux system

echo "this is the second shell script"

echo "generate some user"

#need root

if [ "$(id -u)" == "0" ]

then

echo "under root"

else

echo "u need root,please input root passwd"

su

fi

echo "add some user,passwd see user2passwd"

for USER\_NAME in Ali Bob Carl Dave Eden Frank

do

useradd $USER\_NAME

PASSWD=$RANDOM

echo $USER\_NAME >>user2passwd

echo $PASSWD >>user2passwd

echo $PASSWD | passwd --stdin $USER\_NAME

done

echo "software management"

rpm -qai > software\_info

yum list > yum\_list

## 后续可以对上述两个文件进行相关信息提取

exit 0

Shell脚本编程实例（三）：

#!/bin/bash

#modify configuration files

#Date:2018-04-10

#Author:zp

FILE\_CONF='./hehe.conf'

# 单引号和双引号不一样，单引号屏蔽一切特殊符号

TMP\_FILE="./tmp.$$"

# make anonymous\_enable=yes

sed 's/^.\*anonymous\_enable=.\*/anonymous\_enable=YES/' $FILE\_CONF > $TMP\_FILE

echo $TMP\_FILE

mv -f $TMP\_FILE $FILE\_CONF

Shell脚本编程实例（四）

#!/bin/bash

#编写一个菜单脚本，里面融合函数定义、case案例，参数传递

HINT(){

read -p "Press Enter to continue:"

}

CPU\_INFO(){

echo "run cpu\_info"

echo "system working time"

#uptime -p | awk 'BEGIN{OFS="-"}{print $2,$3}'

uptime -p | awk '{print $2,$3,$4,$5}'| awk -F, '{print $1,$2}'| awk 'BEGIN{OFS="-"}{print $1,$2,$3,$4}'

echo "system load average:"

#uptime | awk -F, '{print $3,$4,$5}' | awk -F: '{print $2}'

uptime | awk -F: '{print $5}'

echo "user numbers:"

uptime | awk -F, '{print $2}'

}

MEM\_INFO(){

echo "run mem\_info"

USED\_MEM=`free -m | awk '/Mem/{print $3}' `

echo "used-mem : $USED\_MEM M"

TOTAL\_MEM=`free -m | awk '/Mem/{print $2}'`

echo "total-mem : $TOTAL\_MEM M"

REMAIN\_MEM=$((TOTAL\_MEM-USED\_MEM))

echo "remain-mem : $REMAIN\_MEM M"

}

PROGRESS\_INFO(){

echo "run progress\_info"

echo "the num of progress:"

NUM=$(ps -aux | wc -l)

PROGRESS\_NUM=$((NUM - 1))

echo "$PROGRESS\_NUM"

echo "the num of runing progress:"

ps -aux | awk '{print $8}' | grep R | wc -l

}

while true

do

clear

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "1.Display CPU info:"

echo "2.Display Mem info:"

echo "3.Display Progress info:"

echo "4.Exit Script:"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

read -p "please select an iterm(1-4):" U\_SELECT

case $U\_SELECT in

1)

CPU\_INFO

HINT

;;

2)

MEM\_INFO

HINT

;;

3)

PROGRESS\_INFO

HINT

;;

4)

exit

;;

\*)

read -p "please select 1-4,press enter to continue:"

esac

done

Shell脚本编程实例（五）

#!/bin/bash

#这是一个测试脚本，用于测试CPU 内存 磁盘空间占用

#Date:2018.04.12

#Author:zp

HINT(){

read -p "Press Enter to continue:"

}

TEST\_CPU(){

echo "run test\_cpu"

for((i=0;i<50000;i++))

do

cd /tmp

echo "generate directory"

PATH1=$RANDOM

mkdir ./test2test$PATH1

cd ./test2test$PATH1

echo "generate file"

echo "heheda$RANDOM" >nihao$PATH1

echo "delete directory and file"

rm -rf /tmp/test2test\*

done

}

TEST\_MEM(){

echo "run test\_mem"

}

TEST\_DISK(){

echo "run test\_disk"

for ((i=0;i<2000;i++))

do

echo "generate $i 300M"

dd if=/dev/zero of=tmp bs=1M count=300

rm -f ./tmp

echo "delete $i 300M"

done

}

while true

do

clear

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "1.TEST CPU:"

echo "2.TEST MEM:"

echo "3.TEST DISK:"

echo "4.Exit Script:"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

read -p "please select an iterm(1-4):" U\_SELECT

case $U\_SELECT in

1)

TEST\_CPU

HINT

;;

2)

TEST\_MEM

HINT

;;

3)

TEST\_DISK

HINT

;;

4)

exit

;;

\*)

read -p "please select 1-4,press enter to continue:"

esac

done