**查询cpu**

## 1、top

这个命令很常用，在第三行有显示CPU当前的使用情况。

[root@li676-235 ~]# top -bn 1 -i -c

top - 14:19:51 up 138 days, 7:15, 1 user, load average: 0.20, 0.33, 0.39

Tasks: 115 total, 1 running, 114 sleeping, 0 stopped, 0 zombie

Cpu(s): 4.5%us, 3.8%sy, 0.0%ni, 91.0%id, 0.6%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 1014660k total, 880512k used, 134148k free, 264904k buffers

Swap: 262140k total, 34788k used, 227352k free, 217144k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

12760 root 20 0 15084 1944 1632 R 2.0 0.2 0:00.01 top -bn 1 -i -c

如上所示，top命令可以看到总体的系统运行状态和cpu的使用率 。

%us：表示用户空间程序的cpu使用率（没有通过nice调度）

%sy：表示系统空间的cpu使用率，主要是内核程序。

%ni：表示用户空间且通过nice调度过的程序的cpu使用率。

%id：空闲cpu

%wa：cpu运行时在等待io的时间

%hi：cpu处理硬中断的数量

%si：cpu处理软中断的数量

%st：被虚拟机偷走的cpu

## 2、vmstat

vmstat命令是最常见的Linux/Unix监控工具，可以展现给定时间间隔的服务器的状态值,包括服务器的CPU使用率，内存使用，虚拟内存 交换情况,IO读写情况。相比top，通过vmstat可以看到整个机器的 CPU,内存,IO的使用情况，而不是单单看到各个进程的CPU使用率和内存使用率。

一般vmstat工具的使用是通过两个数字参数来完成的，第一个参数是采样的时间间隔数，单位是秒，第二个参数是采样的次数。

root@vm-199:~# vmstat 2 1

procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----

 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa

 0  0  97640  53884 192800 578212    0    0     3    20    1   12  1  2 93  3

2表示每个两秒采集一次服务器状态，1表示只采集一次。

实际上，在应用过程中，我们会在一段时间内一直监控，不想监控直接结束vmstat就行了,例如:

root@vm-199:~# vmstat 2

procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----

 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa

 0  1  97640  50348 192812 578352    0    0     3    20    1   12  1  2 93  3

 0  0  97640  54636 192812 578360    0    0     0    14  126  147  2  4 86  8

 0  3  97640  53908 192816 578356    0    0     0    62   96  110  2  4 86  9

 0  0  97640  54156 192816 578360    0    0     0    14  113  118  3  5 83 11

 0  0  97640  53908 192816 578360    0    0     0    16  107  103  2

这表示vmstat每2秒采集数据，一直采集，直到我结束程序。

**Linux 内存监控vmstat命令输出分成六个部分：**

(1)进程procs：

    r：在运行队列中等待的进程数 。

    b：在等待io的进程数 。

(2)Linux 内存监控内存memoy：

    swpd：现时可用的交换内存（单位KB）。

    free：空闲的内存（单位KB）。

    buff: 缓冲去中的内存数（单位：KB）。

    cache：被用来做为高速缓存的内存数（单位：KB）。

(3) Linux 内存监控swap交换页面

    si: 从磁盘交换到内存的交换页数量，单位：KB/秒。

    so: 从内存交换到磁盘的交换页数量，单位：KB/秒。

(4)Linux 内存监控 io块设备:

    bi: 发送到块设备的块数，单位：块/秒。

    bo: 从块设备接收到的块数，单位：块/秒。

(5)Linux 内存监控system系统：

    in: 每秒的中断数，包括时钟中断。

    cs: 每秒的环境（上下文）转换次数。

(6)Linux 内存监控cpu中央处理器：

    cs：用户进程使用的时间 。以百分比表示。

    sy：系统进程使用的时间。 以百分比表示。

    id：中央处理器的空闲时间 。以百分比表示。

常见诊断：

1、假如 r 经常大于4 ，且 id 经常小于40，表示中央处理器的负荷很重。

2、假如 bi，bo 长期不等于0，表示物理内存容量太小。

**每个参数的具体意思如下：**

**r** 表示运行队列(就是说多少个进程真的分配到CPU)，我测试的服务器目前CPU比较空闲，没什么程序在跑，当这个值超过了CPU数目，就会出现CPU瓶颈 了。这个也和top的负载有关系，一般负载超过了3就比较高，超过了5就高，超过了10就不正常了，服务器的状态很危险。top的负载类似每秒的运行队 列。如果运行队列过大，表示你的CPU很繁忙，一般会造成CPU使用率很高。

**b** 表示阻塞的进程,这个不多说，进程阻塞，大家懂的。

**swpd** 虚拟内存已使用的大小，如果大于0，表示你的机器物理内存不足了，如果不是程序内存泄露的原因，那么你该升级内存了或者把耗内存的任务迁移到其他机器。

**free**   空闲的物理内存的大小，我的机器内存总共8G，剩余3415M。

**buff**   Linux/Unix系统是用来存储，目录里面有什么内容，权限等的缓存，我本机大概占用300多M

**cache** cache直接用来记忆我们打开的文件,给文件做缓冲，我本机大概占用300多M(这里是Linux/Unix的聪明之处，把空闲的物理内存的一部分拿来做文件和目录的缓存，是为了提高 程序执行的性能，当程序使用内存时，buffer/cached会很快地被使用。)

**si**  每秒从磁盘读入虚拟内存的大小，如果这个值大于0，表示物理内存不够用或者内存泄露了，要查找耗内存进程解决掉。我的机器内存充裕，一切正常。

**so**  每秒虚拟内存写入磁盘的大小，如果这个值大于0，同上。

**bi**  块设备每秒接收的块数量，这里的块设备是指系统上所有的磁盘和其他块设备，默认块大小是1024byte，我本机上没什么IO操作，所以一直是0，但是我曾在处理拷贝大量数据(2-3T)的机器上看过可以达到140000/s，磁盘写入速度差不多140M每秒

**bo** 块设备每秒发送的块数量，例如我们读取文件，bo就要大于0。bi和bo一般都要接近0，不然就是IO过于频繁，需要调整。

**in** 每秒CPU的中断次数，包括时间中断

**cs** 每秒上下文切换次数，例如我们调用系统函数，就要进行上下文切换，线程的切换，也要进程上下文切换，这个值要越小越好，太大了，要考虑调低线程或者进程的 数目,例如在apache和nginx这种web服务器中，我们一般做性能测试时会进行几千并发甚至几万并发的测试，选择web服务器的进程可以由进程或 者线程的峰值一直下调，压测，直到cs到一个比较小的值，这个进程和线程数就是比较合适的值了。系统调用也是，每次调用系统函数，我们的代码就会进入内核 空间，导致上下文切换，这个是很耗资源，也要尽量避免频繁调用系统函数。上下文切换次数过多表示你的CPU大部分浪费在上下文切换，导致CPU干正经事的 时间少了，CPU没有充分利用，是不可取的。

**us** 用户CPU时间，我曾经在一个做加密解密很频繁的服务器上，可以看到us接近100,r运行队列达到80(机器在做压力测试，性能表现不佳)。

**sy** 系统CPU时间，如果太高，表示系统调用时间长，例如是IO操作频繁。

**id**  空闲 CPU时间，一般来说，id + us + sy = 100,一般我认为id是空闲CPU使用率，us是用户CPU使用率，sy是系统CPU使用率。

**wt** 等待IO CPU时间。

## 3、sar

sar命令语法和vmstat一样。命令不存在时需要安装sysstat包，这个包很有用。

### CPU使用率

例如每1秒采集一次CPU使用率，共采集5次。

[root@li676-235 ~]# sar -u 1 5

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

02:41:25 PM CPU %user %nice %system %iowait %steal %idle

02:41:26 PM all 64.71 0.00 18.63 0.00 0.98 15.69

02:41:27 PM all 29.47 0.00 22.11 14.74 0.00 33.68

02:41:28 PM all 67.33 0.00 31.68 0.99 0.00 0.00

02:41:29 PM all 7.00 0.00 2.00 0.00 0.00 91.00

02:41:30 PM all 69.00 0.00 23.00 0.00 0.00 8.00

Average: all 47.79 0.00 19.48 3.01 0.20 29.52

和top一样，可以看到所有cpu的使用情况。如果需要查看某颗cpu的使用可以用-P参数。例如指定显示0号cpu 的使用情况。

[root@li676-235 ~]# sar -P 0 -u 1 5

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

02:45:14 PM CPU %user %nice %system %iowait %steal %idle

02:45:15 PM 0 44.00 0.00 52.00 4.00 0.00 0.00

02:45:16 PM 0 9.28 0.00 26.80 62.89 1.03 0.00

02:45:17 PM 0 3.06 0.00 14.29 81.63 1.02 0.00

02:45:18 PM 0 4.12 0.00 22.68 72.16 1.03 0.00

02:45:19 PM 0 4.12 0.00 22.68 72.16 1.03 0.00

Average: 0 13.09 0.00 27.81 58.28 0.82 0.00

### 进程队列长度和平均负载状态

例如每1秒采集一次，共采集5次。

[root@li676-235 ~]# sar -q 1 5

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

02:48:47 PM runq-sz plist-sz ldavg-1 ldavg-5 ldavg-15

02:48:48 PM 1 133 0.34 0.43 0.41

02:48:49 PM 2 132 0.34 0.43 0.41

02:48:50 PM 1 133 0.34 0.43 0.41

02:48:51 PM 2 134 0.31 0.42 0.40

02:48:52 PM 1 133 0.31 0.42 0.40

Average: 1 133 0.33 0.43 0.41

输出项：

runq-sz：运行队列的长度（等待运行的进程数）

plist-sz：进程列表中进程（processes）和线程（threads）的数量

ldavg-1：最后1分钟的系统平均负载（System load average）

ldavg-5：过去5分钟的系统平均负载

ldavg-15：过去15分钟的系统平均负载

### 进程创建的平均值和上下文切换的次数

例如每1秒收集一次，共收集5次。

[root@li676-235 ~]# sar -w 1 5

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

02:54:03 PM proc/s cswch/s

02:54:04 PM 1.01 156.57

02:54:05 PM 1.00 132.00

02:54:06 PM 2.00 201.00

02:54:07 PM 2.02 126.26

02:54:08 PM 2.00 114.00

Average: 1.61 145.98

sar命令也可以获取过去指定日期的性能参数。

[root@li676-235 ~]# sar -u -f /var/log/sa/sa20

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

01:10:01 PM CPU %user %nice %system %iowait %steal %idle

01:20:02 PM all 25.78 0.00 8.88 3.59 0.15 61.60

01:30:01 PM all 26.06 0.00 9.96 3.33 0.10 60.55

01:40:01 PM all 25.73 0.00 9.17 3.70 0.09 61.32

01:50:01 PM all 25.70 0.00 9.50 2.79 0.12 61.89

02:00:01 PM all 26.70 0.00 9.73 2.20 0.10 61.28

02:10:01 PM all 26.16 0.00 9.56 4.34 0.11 59.82

02:20:01 PM all 25.49 0.00 9.61 2.76 0.07 62.07

02:30:01 PM all 26.47 0.00 9.94 0.64 0.30 62.65

02:40:02 PM all 27.32 0.00 10.37 3.86 0.15 58.30

02:50:02 PM all 26.98 0.00 10.38 4.56 0.13 57.95

Average: all 26.24 0.00 9.71 3.18 0.13 60.74

## 4、mpstat

这个命令也在sysstat包中，语法类似。

例如每1秒收集一次，共5次。

[root@li676-235 ~]# mpstat 1 5

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

03:01:18 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %idle

03:01:19 PM all 52.53 0.00 23.23 0.00 0.00 0.00 0.00 0.00 24.24

03:01:20 PM all 21.00 0.00 4.00 0.00 0.00 0.00 0.00 0.00 75.00

03:01:21 PM all 53.00 0.00 18.00 0.00 0.00 0.00 0.00 0.00 29.00

03:01:22 PM all 26.00 0.00 3.00 0.00 0.00 0.00 0.00 0.00 71.00

03:01:23 PM all 46.00 0.00 18.00 0.00 0.00 0.00 0.00 0.00 36.00

Average: all 39.68 0.00 13.23 0.00 0.00 0.00 0.00 0.00 47.09

cpu使用情况比sar更加详细些，也可以用-P指定某颗cpu 。

## 5、iostat

这个命令主要用来查看io使用情况，也可以来查看cpu，个人感觉不常用。

[root@li676-235 ~]# iostat -c 1 2

Linux 3.18.5-x86\_64-linode52 (li676-235) 07/20/2015 \_x86\_64\_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle

4.53 0.01 3.81 0.63 0.04 90.99

avg-cpu: %user %nice %system %iowait %steal %idle

38.00 0.00 14.00 0.00 0.00 48.00

## 6、dstat

每秒cpu使用率情况获取

[root@li676-235 ~]# dstat -c

----total-cpu-usage----

usr sys idl wai hiq siq

5 4 91 1 0 0

27 11 62 0 0 0

52 11 37 0 0 0

26 10 63 0 0 1

50 13 37 0 0 1

21 6 72 0 0 0

最占cpu的进程获取

[root@li676-235 ~]# dstat --top-cpu

-most-expensive-

cpu process

mysqld 1.5

php-fpm: pool 24

mysqld 59

**查看io**

## 1. iostat 查看磁盘IO情况

### 安装

yum install sysstat -y

### 语法

-d 表示显示设备（磁盘）使用状态

-x 表示显示和io相关的扩展数据

-k 表示某些使用block为单位的列强制使用Kilobytes为单位

-c 查看cpu信息

### 使用

#iostat -d -k 1 10 #查看TPS和吞吐量信息

tps：该设备每秒的传输次数（Indicate the number of transfers per second that were issued to the device.）

"一次传输"意思是"一次I/O请求"。多个逻辑请求可能会被合并为"一次I/O请求"。"一次传输"请求的大小是未知的。

kB\_read/s：每秒从设备（drive expressed）读取的数据量；

kB\_wrtn/s：每秒向设备（drive expressed）写入的数据量；

kB\_read：读取的总数据量；

kB\_wrtn：写入的总数量：这些单位都为Kilobytes，即KB

#iostat -d -x -k 1 10 #查看设备使用率（%util）、响应时间（await）

以上可以看到，磁盘的平均响应时间较小，为0，磁盘使用率平均为3.0左右，比较小。

#iostat -c 1 10 #查看cpu状态

iostat -x -k

如果%util接近100%,说明产生的I/O请求太多,I/O系统已经满负载,该磁盘可能存在瓶颈。

iostat -d sda 2

监控指定设备

在使用LINUX时经常会遇到系统IO占用很高，系统IO占用可以使用iostat -x 1查看。

linux系统中，管理员可以使用top来监控进程的cpu和内存的使用情况，但是对于磁盘的i/o则只能使用iostat笼统地进行监视，不能对应进程这对于管理员来说很不方便。比如你的服务器负载很低，内存使用也很少，但是硬盘狂转，但你就是无法准确确定是哪个进程在搞鬼。

## 2. iotop 磁盘IO 定位负载来源进程

iotop 是一个用来监视磁盘 I/O 使用状况的 top 类工具。但很多时候知道磁盘IO负载高，但并不知道是什么程序占用的，是PHP，还是MYSQL,还是其它的，这就不好查看了。iotop工具可以实现

iotop 可以清楚地知晓是什么程序在读写磁盘，速度以及命令行，pid 等信息。

iotop可以显示磁盘读写的速率，交换分区进出情况和整体磁盘性能状况，这些都是按照每个进程使用情况来统计。进程列表按照I/O使用状态排序并每秒刷新一次。

### 语法

-p 指定进程ID，显示该进程的IO情况

-u 指定用户名，显示该用户所有的进程IO情况

## 3. vmstat 测硬盘IO

vmstat命令的含义为显示虚拟内存状态（“Viryual Memor Statics”），但是它可以报告关于进程、内存、I/O等系统整体运行状态

### 使用方法

vmstat [-V] [-n] [delay [count]]

-V 显示vmstat的版本；

-n causes the headers not to be reprinted regularly.

-a 显示所有激活和未激活内存的状态；print inactive/active page stats.

-d 显示硬盘统计信息；prints disk statistics

-D 显示硬盘分区表；prints disk table

-p 显示硬盘分区读写状态等；prints disk partition statistics

-s 显示内存使用情况；prints vm table

-m prints slabinfo

-S 定义单位，k K

delay 是两次刷新时间间隔；

单位体积： k:1000 K:1024 m:1000000 M:1048576 (默认是 K)

count 刷新次数；

### 显示参数

r :The number of processes waiting for run time. 等待运行时间的进程数，即等待的进程数、

b :The number of processes in uninterruptible sleep. 在等待io的进程数

swpd: 虚拟内存的使用量

free：空闲内存量

buff：缓冲区中的内存

cache：被用来做为高速缓存的内存数

si: 从磁盘交换到内存的交换页数量，单位：KB/秒。

so: 从内存交换到磁盘的交换页数量，单位：KB/秒。

bi: 发送到块设备的块数，单位：块/秒。

bo: 从块设备接收到的块数，单位：块/秒。

in: 每秒的中断数，包括时钟中断。

cs: 每秒的环境（上下文）转换次数。

us：用户进程使用的时间 。以百分比表示。

sy：系统进程使用的时间。 以百分比表示。

id：中央处理器的空闲时间 。以百分比表示。

wa：io等待时间

st：Time stolen from a virtual machine

## iftop 测网络IO

### 安装

* 第一步：安装EPEL源 yum install epel-release
* 第二部：安装iftop yum install iftop

### 使用

* 1、iftop界面相关说明

界面上面显示的是类似刻度尺的刻度范围，为显示流量图形的长条作标尺用的。

中间的<= =>这两个左右箭头，表示的是流量的方向。

TX：发送流量

RX：接收流量

TOTAL：总流量

Cumm：运行iftop到目前时间的总流量

peak：流量峰值

rates：分别表示过去 2s 10s 40s 的平均流量

* 2、iftop相关参数
* 常用的参数

-i设定监测的网卡，如：# iftop -i eth1

-B 以bytes为单位显示流量(默认是bits)，如：# iftop -B

-n使host信息默认直接都显示IP，如：# iftop -n

-N使端口信息默认直接都显示端口号，如: # iftop -N

-F显示特定网段的进出流量，如# iftop -F 10.10.1.0/24或# iftop -F 10.10.1.0/255.255.255.0

-h（display this message），帮助，显示参数信息

-p使用这个参数后，中间的列表显示的本地主机信息，出现了本机以外的IP信息;

-b使流量图形条默认就显示;

-f这个暂时还不太会用，过滤计算包用的;

-P使host信息及端口信息默认就都显示;

-m设置界面最上边的刻度的最大值，刻度分五个大段显示，例：# iftop -m 100M

* 进入iftop画面后的一些操作命令(注意大小写)

按h切换是否显示帮助;

按n切换显示本机的IP或主机名;

按s切换是否显示本机的host信息;

按d切换是否显示远端目标主机的host信息;

按t切换显示格式为2行/1行/只显示发送流量/只显示接收流量;

按N切换显示端口号或端口服务名称;

按S切换是否显示本机的端口信息;

按D切换是否显示远端目标主机的端口信息;

按p切换是否显示端口信息;

按P切换暂停/继续显示;

按b切换是否显示平均流量图形条;

按B切换计算2秒或10秒或40秒内的平均流量;

按T切换是否显示每个连接的总流量;

按l打开屏幕过滤功能，输入要过滤的字符，比如ip,按回车后，屏幕就只显示这个IP相关的流量信息;

按L切换显示画面上边的刻度;刻度不同，流量图形条会有变化;

按j或按k可以向上或向下滚动屏幕显示的连接记录;

按1或2或3可以根据右侧显示的三列流量数据进行排序;

按<根据左边的本机名或IP排序;

按>根据远端目标主机的主机名或IP排序;

按o切换是否固定只显示当前的连接;

按f可以编辑过滤代码，这是翻译过来的说法，我还没用过这个！

按!可以使用shell命令，这个没用过！没搞明白啥命令在这好用呢！

按q退出监控。

**测cache**

## 1. valgrind的cachegrid (+slurm)

valgrind --tool=cachegrind ~/GeminiGraph-master/toolkits/pagerank

* Cachegrind 模拟您的程序与机器缓存等级和（可选）分支预测单元的互动。它跟踪模拟的一级指令和数据缓存的用量以便探测不良代码与这一级缓存的互动；最高一级，可以是二级或者三级缓存，用来跟踪对主内存的访问。因此，使用 Cachegrind 运行的程序速度比正常运行时要慢 20-100 倍。

是否用slurm跑的cache miss tate的区别：

是不是slurm因为用到了网络通信，所以进行的内存访问更多的是在网络通信的程序上，而不是数据访问上

迭代100次和迭代1次，cache缺失的次数都很接近。说明cachegrind并没有记录到，使用slurm之后访问数据的cache缺失

## 2. perf

perf stat -e cache-references,cache-misses ./toolkits/pagerank ./../data\_sbb/enwiki-2013.bin 4206757 100

这个没有cachegrind那么严格，是采用采样的方法测试的，速度比较快，测得的是近似值，不过已经可以达到目的了

在测得时候，把迭代次数设置成80次，这样能够确保算法收敛，测出来才有意义哈

这个perf是采用采样的方式，所以有些误差，没有cachegrind采用的完全模拟的方式准确，不过也有参考价值。可以多测几次取平均。

## 3. 直接用mpi跑

mpiexec -n 10 ./../GeminiGraph-master/toolkits/pagerank ./../data\_sbb/amazon-2008.bin 735322 10

**查内存泄漏 valgrid**

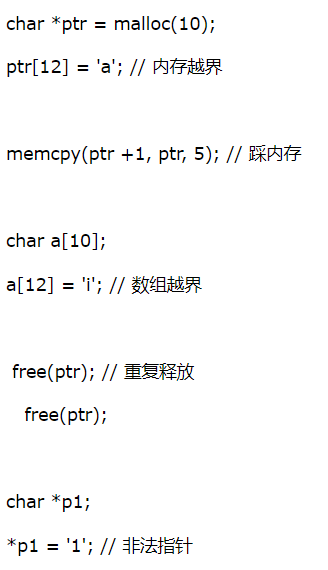
Valgrind使用起来非常简单，你甚至不需要重新编译你的程序就可以用它。当然如果要达到最好的效果，获得最准确的信息，还是需要按要求重新编译一下的。比如在使用memcheck的时候，最好关闭优化选项。

       valgrind命令的格式如下：valgrind [valgrind-options] your-prog [your-prog options]



## **Memcheck**

最常用的工具，用来检测程序中出现的内存问题，所有对内存的读写都会被检测到，一切对malloc、free、new、delete的调用都会被捕获。所以，它能检测以下问题：堆中的内存越界、踩内存、栈中的内存越界、非法指针使用、重复free。



       1、对未初始化内存的使用；

       2、读/写释放后的内存块；

       3、读/写超出malloc分配的内存块；

       4、读/写不适当的栈中内存块；

       5、内存泄漏，指向一块内存的指针永远丢失；

       6、不正确的malloc/free或new/delete匹配；

       7、memcpy()相关函数中的dst和src指针重叠。

1、leak-check

    --leak-check=<no|summary|yes|full> [default: summary]

    用于控制内存泄漏检测力度。

    no，不检测内存泄漏；

    summary，仅报告总共泄漏的数量，不报告具体泄漏位置；

    yes/full，报告泄漏总数、泄漏的具体位置。

2、show-reachable

    --show-reachable=<yes|no> [default: no]

    用于控制是否检测控制范围之外的泄漏，比如全局指针、static指针等。

#include <stdlib.h>

#include <stdio.h>

//char \*gptr = NULL;

int main(void)

{

    gptr = (char \*)malloc(10);

    return 0;

}

对应以上代码，若--show-reachable为no，则valgrind不报告内存泄漏，否则会报告。

3、undef-value-errors

--undef-value-errors=<yes|no> [default: yes]

用于控制是否检测代码中使用未初始化变量的情况。

对应以下代码：

    int a;

    printf("a = %d \n", a);

若 --undef-value-errors＝no，则valgrind不报告错误，否则报告“Use of uninitialised value ...”的错误。

4、其他选项

    --log-file=filename 将结果输出到文件。

    --log-socket=192.168.0.1:12345 输出到网络。

    --trace-children=<yes|no> [default: no]

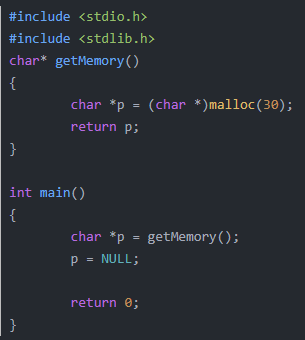
    --track-fds=<yes|no> [default: no]

    --log-fd=<number> [default: 2, stderr]

    --xml=<yes|no> [default: no]

    --num-callers=<number> [default: 12]

    --show-below-main=<yes|no> [default: no]



[root@xxx ~/valgrind-3.8.1/bin]# **./valgrind --tool=memcheck --leak-check=yes --show-reachable=yes ./a.out**

==19226== Memcheck, a memory error detector

==19226== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.

==19226== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info

==19226== Command: ./a.out

==19226==

==19226==

==19226== HEAP SUMMARY:

==19226== in use at exit: 30 bytes in 1 blocks

==19226== total heap usage: 1 allocs, 0 frees, 30 bytes allocated

==19226==

==19226== 30 bytes in 1 blocks are definitely lost in loss record 1 of 1

==19226== at 0x4C278FE: **malloc** (vg\_replace\_malloc.c:270)

==19226== by 0x4005B5: **getMemory()** (in /root/valgrind-3.8.1/bin/a.out)

==19226== by 0x4005CC: main (in /root/valgrind-3.8.1/bin/a.out)

==19226==

==19226== LEAK SUMMARY:

==19226== definitely lost: 30 bytes in 1 blocks

==19226== indirectly lost: 0 bytes in 0 blocks

==19226== possibly lost: 0 bytes in 0 blocks

==19226== still reachable: 0 bytes in 0 blocks

==19226== suppressed: 0 bytes in 0 blocks

==19226==

==19226== For counts of detected and suppressed errors, rerun with: -v

==19226== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 6 from 6)

[root@xxx ~/valgrind-3.8.1/bin]#

 不过， 这样看着也挺蛋疼的， 如果代码过多， 肉眼据不太好分析了， 能不能把内存泄漏的代码行给找出来呢？ 当然能！ 回想一下我们之前介绍过得core dump定位到代码行的问题， 两个必要条件是:  编译时必须有-g参数； 编译后不能strip.  我们一起再看看：

[root@xxx ~/valgrind-3.8.1/bin]# g++ -g test.cpp

[root@xxx ~/valgrind-3.8.1/bin]#

[root@xxx ~/valgrind-3.8.1/bin]# **./valgrind --tool=memcheck --leak-check=yes --show-reachable=yes ./a.out**

==20448== Memcheck, a memory error detector

==20448== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.

==20448== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info

==20448== Command: ./a.out

==20448==

==20448==

==20448== HEAP SUMMARY:

==20448== in use at exit: 30 bytes in 1 blocks

==20448== total heap usage: 1 allocs, 0 frees, 30 bytes allocated

==20448==

==20448== 30 bytes in 1 blocks are definitely lost in loss record 1 of 1

==20448== at 0x4C278FE: malloc (vg\_replace\_malloc.c:270)

==20448== by 0x4005B5: getMemory() (**test.cpp:5**)

==20448== by 0x4005CC: main (**test.cpp:11**)

==20448==

==20448== LEAK SUMMARY:

==20448== definitely lost: 30 bytes in 1 blocks

==20448== indirectly lost: 0 bytes in 0 blocks

==20448== possibly lost: 0 bytes in 0 blocks

==20448== still reachable: 0 bytes in 0 blocks

==20448== suppressed: 0 bytes in 0 blocks

==20448==

==20448== For counts of detected and suppressed errors, rerun with: -v

==20448== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 6 from 6)

[root@xxx ~/valgrind-3.8.1/bin]#

## **Cachegrind**

       Cache分析器，它模拟CPU中的一级缓存I1，Dl和二级缓存，能够精确地指出程序中cache的丢失和命中。如果需要，它还能够为我们提供cache丢失次数，内存引用次数，以及每行代码，每个函数，每个模块，整个程序产生的指令数。这对优化程序有很大的帮助。

**运行中的程序调试**

## 断点（BreakPoint）

在代码的指定位置中断，这个是我们用得最多的一种。设置断点的命令是break，它通常有如下方式：

* break <function>    在进入指定函数时停住
* break <linenum>    在指定行号停住。
* break +/-offset    在当前行号的前面或后面的offset行停住。offiset为自然数。
* break filename:linenum    在源文件filename的linenum行处停住。
* break ... if <condition>    ...可以是上述的参数，condition表示条件，在条件成立时停住。比如在循环境体中，可以设置break if i=100，表示当i为100时停住程序。

可以通过info breakpoints [n]命令查看当前断点信息。此外，还有如下几个配套的常用命令：

* delete    删除所有断点
* delete breakpoint [n]    删除某个断点
* disable breakpoint [n]    禁用某个断点
* enable breakpoint [n]    使能某个断点

## 一、多线程调试

多线程调试可能是问得最多的。其实，重要就是下面几个命令：

**info thread** 查看当前进程的线程。

**thread <ID>** 切换调试的线程为指定ID的线程。

**break file.c:100 thread all**  在file.c文件第100行处为所有经过这里的线程设置断点。

**set scheduler-locking off|on|step，**这个是问得最多的。在使用step或者continue命令调试当前被调试线程的时候，其他线程也是同时执行的，怎么只让被调试程序执行呢？通过这个命令就可以实现这个需求。

**off** 不锁定任何线程，也就是所有线程都执行，这是默认值。

**on** 只有当前被调试程序会执行。

**step** 在单步的时候，除了next过一个函数的情况(熟悉情况的人可能知道，这其实是一个设置断点然后continue的行为)以外，只有当前线程会执行。

## 多线程所有的线程id

查看thread id的方法有：  
1. sh-3.2# ps -efL | grep process，  
ps命令指定-L命令选项可以用来查看进程下所包含的所有线程。

## 二、调试宏

在GDB下，我们无法print宏定义，因为宏是预编译的。但是我们还是有办法来调试宏，这个需要GCC的配合。

在GCC编译程序的时候，加上-ggdb3参数，这样，你就可以调试宏了。

另外，你可以使用下述的GDB的宏调试命令 来查看相关的宏。

info macro – 你可以查看这个宏在哪些文件里被引用了，以及宏定义是什么样的。

macro – 你可以查看宏展开的样子。

1、首先获得程序的PID

ps -ef | grep xxxxx 

2、进入调试程序

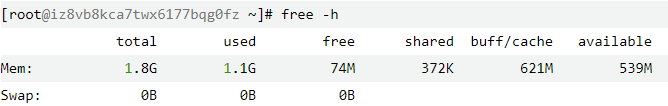
1. gdb attach PID

3、gcore命令生成CORE文件

4、进程信息可以用info  proc显示

5、寄存器信息可以用info reg显示

**free buff/cache**



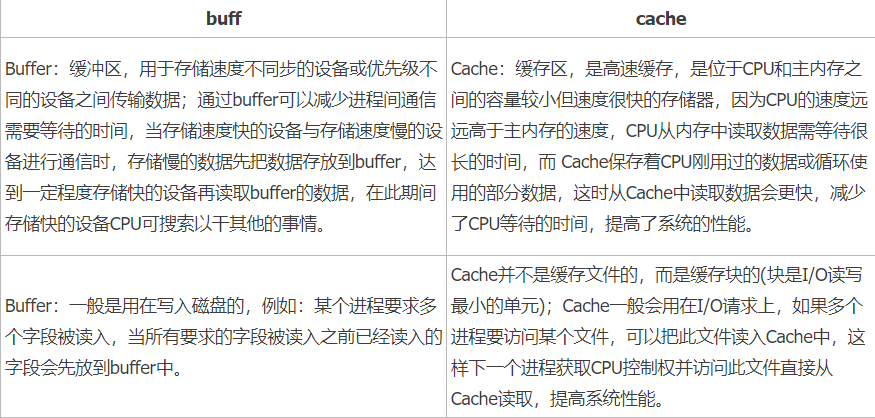
shared：用于进程之间相互共享数据。  
Used：已使用内存。  
total：内存总量。  
free：未使用的内存。  
available：开启一个新程序能够使用的最大内存。

buffer：在内存中还没有被写入到磁盘的something。

作用：为了解决CPU向磁盘中存储something时的速度差，将积攒一大波something再向磁盘中存储一次，buffer用于积攒something。

cache：已经从磁盘读出到内存的something。 作用：cpu使用磁盘中的something时，将something存入内存，cpu在较短时间内再次使用something时，直接从cache中调取something，不需要从磁盘调取，节省调取时间。

计算内存使用情况：

totaled=used+free+buff/cache

**$ pidstat -d 1**

命令的含义：展示I/O统计，每秒更新一次

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

Linux下， 获取文件大小 ls -l、wc

查看Linux日志tail、cat、tac、head、echo

**dmesg**

使用dmesg命令可以快速查看最后一次系统引导的引导日志

通常它的内容会很多，所以我们往往使用如下命令以分页的方式显示引导信息：

# dmesg | more

1、cat命令：

功能：1）显示整个文件。

示例:$ cat fileName

2）把文件串连接后传到基本输出，如将几个文件合并为一个文件或输出到屏幕。

示例：$ cat file1 file2 > file

说明：把档案串连接后传到基本输出(屏幕或加 > fileName 到另一个档案)

cat参数详解：

-n 或 –number 由 1 开始对所有输出的行数编号

-b 或 –number-nonblank 和 -n 相似，只不过对于空白行不编号

-s 或 –squeeze-blank 当遇到有连续两行以上的空白行，就代换为一行的空白行

-v 或 –show-nonprinting

2、more命令：

以百分比的形式查看日志。

3、less命令：

跟more功能差不多，只不过less支持前后翻阅文件。

4、head命令：

功能：从文本文件的头部开始查看，head 命令用于查看一个文本文件的开头部分。

示例如下：

head example.txt 显示文件 example.txt 的前十行内容；

head -n 20 example.txt 显示文件 example.txt 的前二十行内容；

head详解：

-n 指定你想要显示文本多少行。

-n number 这个参数选项必须是十进制的整数，它将决定在文件中的位置，以行为单位。

-c number 这个参数选项必须是十进制的整数，它将决定在文件中的位置，以字节为单位。

5、tail命令：

功能：tail 命令用于显示文本文件的末尾几行。

示例如下：

tail example.txt 显示文件 example.txt 的后十行内容；

tail -n 20 example.txt 显示文件 example.txt 的后二十行内容；

tail -f example.txt 显示文件 example.txt 的后十行内容并在文件内容增加后，自动显示新增的文件内容。

tail -n 50 -f example.txt 显示文件 example.txt 的后50行内容并在文件内容增加后，自动显示新增的文件内容。

注意：

最后一条命令非常有用，尤其在监控日志文件时，可以在屏幕上一直显示新增的日志信息。

tail详解：

-b Number 从 Number 变量表示的 512 字节块位置开始读取指定文件。

-c Number 从 Number 变量表示的字节位置开始读取指定文件。

-f 如果输入文件是常规文件或如果 File 参数指定 FIFO（先进先出），

那么 tail 命令不会在复制了输入文件的最后的指定单元后终止，而是继续

从输入文件读取和复制额外的单元（当这些单元可用时）。如果没有指定 File 参数，

并且标准输入是管道，则会忽略 -f 标志。tail -f 命令可用于监视另一个进程正在写入的文件的增长。

-k Number 从 Number 变量表示的 1KB 块位置开始读取指定文件。

-m Number 从 Number 变量表示的多字节字符位置开始读取指定文件。使用该标志提供在单字节和双字节字符代码集环境中的一致结果。

-n Number 从首行或末行位置来读取指定文件，位置由 Number 变量的符号（+ 或 - 或无）表示，并通过行号 Number 进行位移。

-r 从文件末尾以逆序方式显示输出。-r 标志的缺省值是以逆序方式显示整个文件。如果文件大于 20,480 字节，那么-r标志只显示最后的 20,480 字节。 -r 标志只有

与 -n 标志一起时才有效。否则，就会将其忽略。

linux查看日志文件内容命令

tail -f test.log  
你会看到屏幕不断有内容被打印出来. 这时候中断第一个进程Ctrl-C,

---------------------------  
linux 如何显示一个文件的某几行(中间几行)

从第3000行开始，显示1000行。即显示3000~3999行  
cat filename | tail -n +3000 | head -n 1000

显示1000行到3000行  
cat filename| head -n 3000 | tail -n +1000

\*注意两种方法的顺序  
分解：  
tail -n 1000：显示最后1000行  
tail -n +1000：从1000行开始显示，显示1000行以后的  
head -n 1000：显示前面1000行

用sed命令  
sed -n '5,10p' filename 这样你就可以只查看文件的第5行到第10行。

例：cat mylog.log | tail -n 1000 #输出mylog.log 文件最后一千行

---------------------------  
cat主要有三大功能：  
1.一次显示整个文件。$ cat filename  
2.从键盘创建一个文件。$ cat > filename   
只能创建新文件,不能编辑已有文件.  
3.将几个文件合并为一个文件： $cat file1 file2 > file  
参数：  
-n 或 --number 由 1 开始对所有输出的行数编号  
-b 或 --number-nonblank 和 -n 相似，只不过对于空白行不编号  
-s 或 --squeeze-blank 当遇到有连续两行以上的空白行，就代换为一行的空白行  
-v 或 --show-nonprinting  
例：  
把 textfile1 的档案内容加上行号后输入 textfile2 这个档案里  
cat -n textfile1 > textfile2

把 textfile1 和 textfile2 的档案内容加上行号（空白行不加）之后将内容附加到 textfile3 里。  
cat -b textfile1 textfile2 >> textfile3  
  
把test.txt文件扔进垃圾箱，赋空值test.txt  
cat /dev/null > /etc/test.txt   
注意：>意思是创建，>>是追加。千万不要弄混了。  
------------------------------------------  
tac (反向列示)  
tac 是将 cat 反写过来，所以他的功能就跟 cat 相反， cat 是由第一行到最后一行连续显示在萤幕上，  
而 tac 则是由最后一行到第一行反向在萤幕上显示出来！

------------------------------------------  
在Linux中echo命令用来在标准输出上显示一段字符，比如：  
echo "the echo command test!"

这个就会输出“the echo command test!”这一行文字！

echo "the echo command test!">a.sh  
这个就会在a.sh文件中输出“the echo command test!”这一行文字！   
该命令的一般格式为： echo [ -n ] 字符串其中选项n表示输出文字后不换行；字符串能加引号，也能不加引号。  
用echo命令输出加引号的字符串时，将字符串原样输出；  
用echo命令输出不加引号的字符串时，将字符串中的各个单词作为字符串输出，各字符串之间用一个空格分割。

怎么查看堆栈里的值

bt

打印当前的函数调用栈的所有信息。

bt <-n>

-n表一个负整数，表示只打印栈底下n层的栈信息

如果你要查看某一层的信息，你需要在切换当前的栈，一般来说，程序停止时，最顶层的栈就是当前栈，如果你要查看栈下面层的详细信息，首先要做的是切换当前栈。

F <n>是一个从0开始的整数，是栈中的层编号。比如：frame 0，表示栈顶，frame 1，表示栈的第二层。

up

表示向栈的上面移动n层，可以不打n，表示向上移动一层。

down

表示向栈的下面移动n层，可以不打n，表示向下移动一层。

Linux内存管理机制：分段分页

cmd显示不是内部或外部命令：

输入C:\WINDOWS\SYSTEM32\

从网上下载到服务器并解压：

$ wget https://repo.continuum.io/archive/Anaconda2-4.2.0-Linux-x86\_64.sh  
$ tar -xvf mpich-3.2.tar.gz  
  
激活环境变量：（关掉重新进也可以）  
vim ./.bashrc  
source ./.bashrc  
  
安装到指定目录（在解压后文件目录下运行）：  
$ ./configure --prefix=/usr/bin  
  
测试是否能访问外网：  
$ ping www.baidu.com  
  
换到个人超级用户：  
$ su sc

$ cd /home/sc

切换到root用户：

sudo su root

查看所有目录：

$ ls -al

查看pagerank正在运行的进程号：

ps -A|grep pagerank

杀死pagerank进程：

kill pagerank

一次性杀死所有pagerank进程：

pkill pagerank

打开文件：

vim configure

可以关闭终端后继续跑：

nohup 原始命令 > cxx.txt &

查看操作系统：

lsb\_release -a

如果没有装lsb：yum install redhat-lsb

bash: rpmbuild: command not found

[userid@hostname ~]# sudo yum install rpm-build

安装fio：

sudo apt-get install fio

^x Exit

按Ctrl+x

configure: error: no acceptable C compiler found in $PATH

解决方案： yum install gcc

将文档a复制成b（相当于备份并改名）。

cp -i a b

或，

cp a b

配置用lscpu查看

error: can't open cache simulation output file

解决方案：一般是文件路径不对，看下是不是进错用户，或者集群里有节点没有装对应的软件或者数据集

添加磁盘分区：

[具体方法1](http://blog.csdn.net/tianlesoftware/article/details/5642883)

[具体方法2](http://wqmsl.blog.51cto.com/847418/305313)

[具体方法3](http://www.cnblogs.com/alylee/p/Linux_disk_management_partition_mount.html)

1.df -sh 当前磁盘分区挂载点使用量  
2.fdisk -l 列出磁盘设备的分区表（磁盘总量、已分配的）  
3.fdisk /dev/sdb 进入磁盘分区操作  
p 打印  
n 新建  
选择编号：1  
选择p主/e扩展：p  
分配大小：回车全部分配  
w 保存  
Disk /dev/sdb 分区生产了/dev/sdb1  
partprobe 立即生效  
4.mkfs.ext3 /dev/sdb1 直接格式化/dev/sdb1 或  
mkfs -t ext3 /dev/sdb1  
5./etc/fstab 添加 /dev/sdb1 /home ext3 defaults 0 0  
6.mount /dev/sdb1 /home/tt/app 挂载

7.lsblk 查看系统块设备信息

input/output error：

如果cd /home报上面那个错的话，先

df -h看/home挂载在哪个文件系统，然后xfs\_repair -d  /dev/mapper/cl-home 再重启服务器

-d表示用最高级别去修复系统

shift+g 跳转到最后一行

gg跳到第一行

选y/n的时候按多了建：

Ctrl+回退

退出：

|  |
| --- |
| 按ESC，然后输入:q！ |

删目录grapglab及其所有子目录：

rm graphlab/ -fr

删用户sbb：

[root@localhost ~]userdel -r sbb

改用户密码：

先进root用户：sudo su root

passwd sbb

然后输入新密码，会输两次

然后就可以回到自己的用户了：su sbb

回到自己的根目录：cd ~

从windows传文件到linux：

先装rz/sz：sudo apt-get install lrzsz 或 yum install lrzsz

再在linux下运行rz会弹出windows下的目录

-bash: lsb\_release: command not found

yum install -y redhat-lsb

解压到指定目录（先到指定目录下，然后再解压包带路径）：

~/test01/lib$ tar -zxvf ~/webgraph-deps.tar.gz

看磁盘满了没：

df -h

看文件夹大小：

du -h --max-depth=1

在当前目录下有大文件，看是哪一个（total显示的是文件夹这个文件，不是文件夹里面文件的总大小）：

ll -h

跳转到338节点过程：  
1.  
202.114.10.146端口号2013账号jumper密码jumptocgcl  
2.  
ssh liao@202.114.10.172 或者ssh liao@load11  
password: liao  
3.  
ssh node338

新建用户：

sudo adduser sbb

如果没有权限：

su

然后输密码

就到了root下，建完用户后输入exit切回

把用户sbb加入sudoer：

sudo su root

vim /etc/sudoers

按yyp复制光标行并粘贴到其下一行

改完之后保存：按esc和：wq！

回到sbb：su sbb

查看文件大小

#查看当前目录下的所有目录以及子目录的大小

$ du -h

$ du -ah

linux中df命令的功能是用来检查linux服务器的文件系统的磁盘空间占用情况。可以利用该命令来获取硬盘被占用了多少空间，目前还剩下多少空间等信息。

df –i输出iNode的信息

df –h 输出系统剩余空间

#查看当前目录及其指定深度目录的大小

du -h –-max-depth=0

1. Linux查看CPU和内存使用情况

Top htop

1. 查看当前路径 pwd
2. 当前文件夹文件的权限修改：chmod修改权限，chown修改文件的所有者
3. 连服务器的方式：ssh(应用xshell putty vnc 远程桌面)，teamviewer

ls -l > a.txt，列表的内容写入文件a.txt中（覆盖写）

ls -al >> aa.txt，列表的内容追加到文件aa.txt的末尾

输出linux某一列

grep -a  '.'  x.txt | tr -s ' ' ' ' |cut -d  ' ' -f 2

对命令的解释，grep -a  '.'   x.txt | 作用是将x.txt中的文本安行读出，并管道方式传给tr命令.

tr -s ' ' ' '  | 用于合并连续的空格为一个空格，并将结果以管道方式传给cut命令.

cut -d  ' ' -f 2  用于获取第二列数据

find和grep

grep是查找匹配条件的行，find是搜索匹配条件的文件。

find，搜索文件及目录

atime的意思是access time，即文件的最近的一次访问时间，+n意思为查找n天以前的文件，-n为查找n天以内的文件

mtime比较好理解，为modify time，即文件数据最新的修改时间，指的就是文件内容的最新修改时间。

ctime +10：十小时前更改过的文件或目录

find /opt -iname "\*" -atime 1 -type f  
【找出 /opt 下一天前访问过的文件】

**grep**，在文本中查询内容，常用grep –sIrHn “查找的字符串”，输出当前目录及子目录包含字符串的文件和行号（包括行信息）。

ldd查看程序运行调用的库文件

strace追踪程序的系统调用

cp ‐r dir1 dir2，递归复制命令

rm ‐rf \*，删除所有内容，包含目录和文件，r表示递归，f表示强制

top命令用于查看最活跃进程的实时信息，而ps提供的是进程的快照。要查看具体某个进程是用什么命令启动的，可以通过ps命令来查看。

$ ps –ef

可以看到输出中用java命令启动java程序A的记录。将ps的输出用grep继续过滤，就能看到所有java进程的信息了。

$ ps -ef | grep java

端口占用情况

lsof -i:端口号 查看某个端口是否被占用

使用netstat -anp|grep 80 80端口是否被占用 listen表示正在运行

杀死某端口号的进程步骤：

1 netstat -nlp 查看占用端口号的服务；2找到该端口号的进程；3找到该进程id；4 kill它。

kill [‘]netstat –nlp |grep:80|awk ‘{print $7}’|awk –F ”/” ‘{print $1}’ [‘]

正则表达式：1[3|5|7][0-9]{9}

**管道命令**操作符是：”|”,它仅能处理经由前面一个指令传出的正确输出信息，也就是 standard output 的信息，对于 stdandard error 信息没有直接处理能力。然后，传递给下一个命令，作为标准的输入 standard input.

管道与重定向的区别是：

1、左边的命令应该有标准输出 | 右边的命令应该接受标准输入  
   左边的命令应该有标准输出 > 右边只能是文件  
   左边的命令应该需要标准输入 < 右边只能是文件

2、**管道**触发两个子进程执行"|"两边的程序；而**重定向**是在一个进程内执行

**重定向**

打开一个终端，创建一个管道文件1.p,将”hello”字符串重定向到1.p中，echo是指回显到屏幕上。

[root@localhost day01]# mkfifo 1.p

[root@localhost day01]# echo "hello" > 1.p

打开另外一个终端，执行，字符串会回显到屏幕上。

[zwe@localhost day01]$ cat 1.p

# linux文件覆盖问题：权限和时间戳

stat命令中出现的三个时间戳：Access（atime）,Modify（mtime）,Change（ctime）。

atime:访问时间，读取文件或者执行文件时会更改，命令如cat,vi

mtime:修改时间，文件的内容被改变时，就会更改，如vi

ctime:改变时间，文件的标签如属性、用户、用户组、权限、内容等被改变时，就会更改，如chmod,chown,vi6。

在linux系统中，许多场合都使用时间戳的方式来表示时间。

①用标准时间的格式显示当前日期：

[root@localhost day01]# date

Fri Nov 17 19:04:46 PST 2017

②用时间戳的方式显示当前日期：

[root@localhost day01]# date **+%s**

1510974328

③时间戳转化为标注时间：

date: invalid date `+%s'

[root@localhost day01]# date -d @1510974328

Fri Nov 17 19:05:28 PST 2017

**两台Linux系统之间传输文件的几种方法**

1 scp传输速度较慢,但使用ssh通道保证了传输的安全性

* 将本地文件拷贝到远程

scp 文件名 –用户名@计算机IP或者计算机名称:远程路径

* 从远程将文件拷回本地

scp –用户名@计算机IP或者计算机名称:文件名 本地路径

**2. rcp**  
目标主机需要事先打开rcp功能，并设置好rcp的权限：把源主机加入到可信任主机列表中，否则无法在源主机上使用rcp远程复制文件到目标主机。

rcp -r local\_dir remote\_hostname:remote\_dir

**将当前目录下的 test1 复制到名为 webserver1 的远程系统：**

rcp test1 webserver1:/home/root/test3

**将远程系统 webserver1中的 test2**

rcp webserver1:/home/root/test2

3从网上下载wget [参数] ftp://<目标机器ip或主机名>/<文件的绝对路径>

# 4 rsync差异化传输(支持断点续传,数据同步)

“rsync”的一个优点就是，不会拷贝全部的文件，只会拷贝本地目录和远程目录中有区别的文件。而且它还使用很高效的压缩算法，这样拷贝的速度就很快。

rsync [参数] <源地址（用户名@IP地址或主机名）>:<文件路径> @<目的地址（用户名 @IP 地址或主机名）>:<文件路径>

# 5 管道传输(降低IO开销)

gzip -c sda.img | ssh root@192.168.1.110 "gunzip -c - > /image/sda.img"

对sda.img使用gzip压缩,-c参数表示输出到stdout,即通过管道传送

gunzip -c - 中的”-“表示接收从管道传进的sdtin

# cut和grep的区别

 cut是将一行信息当中，取出某部分我们想要的，信息处理一行为单位。而grep则是分析一行信息， 若当中有我们所需要的信息，就将该行拿出来。

|  |
| --- |
| [root@www ~]# cut -d'分隔字符' -f fields <==用于有特定分隔字符  [root@www ~]# cut -c 字符区间            <==用于排列整齐的信息  选项与参数：  -d  ：后面接分隔字符。与 -f 一起使用；  -f  ：依据 -d 的分隔字符将一段信息分割成为数段，用 -f 取出第几段的意思；  -c  ：以字符 (characters) 的单位取出固定字符区间； |

　　范例一：将 PATH 变量取出，我要找出第五个路径。

|  |  |
| --- | --- |
| 1  2  3  4 | [root@www ~]# echo $PATH | cut -d ':' -f 5  # 如同上面的数字显示，我们是以『 : 』作为分隔，因此会出现 /usr/local/bin  # 那么如果想要列出第 3 与第 5 呢？，就是这样：  [root@www ~]# echo $PATH | cut -d ':' -f 3,5 |

 　　范例二：将 export 输出的信息，取得第 12 字符以后的所有字符串

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | [root@www ~]# export  declare -x HISTSIZE="1000"  declare -x INPUTRC="/etc/inputrc"  declare -x KDEDIR="/usr"  declare -x LANG="zh\_TW.big5"  .....(其他省略).....  # 注意看，每个数据都是排列整齐的输出！如果我们不想要『 declare -x 』时，  # 就得这么做：    [root@www ~]# export | cut -c 12-  HISTSIZE="1000"  INPUTRC="/etc/inputrc"  KDEDIR="/usr"  LANG="zh\_TW.big5" |

 　　CUT -d命令适合操作具有固定分割符的文本，用 -c 可以处理比较具有格式的输出数据，还可以指定某个范围的值，例如第 12-20 的字符，就是 cut -c 12-20。cut 主要的用途在于将『同一行里面的数据进行分解！』最常使用在分析一些数据或文字数据的时候！ 这是因为有时候我们会以某些字符当作分割的参数，然后来将数据加以切割，以取得我们所需要的数据。

|  |
| --- |
| [root@www ~]# grep [-acinv] [--color=auto] '搜寻字符串' filename  选项与参数：  -a ：将 binary 文件以 text 文件的方式搜寻数据  -c ：计算找到 '搜寻字符串' 的次数  -i ：忽略大小写的不同，所以大小写视为相同  -n ：顺便输出行号  -v ：反向选择，亦即显示出没有 '搜寻字符串' 内容的那一行！  --color=auto ：可以将找到的关键词部分加上颜色的显示喔！ |

　　具体的操作实例如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | 范例一：将 last 当中，有出现 root 的那一行就取出来；  [root@www ~]# last | grep 'root'    范例二：与范例一相反，只要没有 root 的就取出！  [root@www ~]# last | grep -v 'root'    范例三：在 last 的输出信息中，只要有 root 就取出，并且仅取第一栏  [root@www ~]# last | grep 'root' |cut -d ' ' -f1  # 在取出 root 之后，利用上个命令 cut 的处理，就能够仅取得第一栏啰！    范例四：取出 /etc/man.config 内含 MANPATH 的那几行  [root@www ~]# grep --color=auto 'MANPATH' /etc/man.config  ....(前面省略)....  MANPATH\_MAP     /usr/X11R6/bin          /usr/X11R6/man  MANPATH\_MAP     /usr/bin/X11            /usr/X11R6/man  MANPATH\_MAP     /usr/bin/mh             /usr/share/man |

# 有你名字这一行的信息用什么命令调出来

文件| grep '我的名字'

# 每条信息用逗号隔开取所有同学名字，用cut怎么截取

# 查看访问次数在前100里的ip

获取最多访问的前10个IP地址及访问次数。

既然是统计，那么awk是必不可少的，好用而高效。

命令如下：

**awk '{a[$1] += 1;} END {for (i in a) printf("%d %s\n", a[i], i);}' 日志文件 | sort -n | tail**

首先用awk统计出来一个列表，然后用sort进行排序，最后用tail取最后的10个。

以上参数可以略作修改显示更多的数据，比如将tail加上-n参数等，另外日志格式不同命令也可能需要稍作修改。

# 当前WEB服务器中联接次数最多的ip地址

#netstat -ntu |awk '{print $5}' |sort | uniq -c| sort -nr

# 查看日志中访问次数最多的前10个IP

#cat access\_log |cut -d ' ' -f 1 | sort |uniq -c | sort -nr | awk '{print $0 }' | head -n 10 | less

# 查看日志中出现100次以上的IP

#cat access\_log |cut -d ' ' -f 1 | sort |uniq -c | awk '{if ($1 > 100) print $0}'｜sort -nr | less

# 查看最近访问量最高的文件

#cat access\_log | tail -10000 | awk '{print $7}' | sort | uniq -c | sort -nr | less

# 查看日志中访问超过100次的页面

#cat access\_log | cut -d ' ' -f 7 | sort |uniq -c | awk '{if ($1 > 100) print $0}' | less

# 统计某url，一天的访问次数

#cat access\_log | grep '12/Aug/2009' | grep '/images/index/e1.gif' | wc | awk '{print $1}'

# 前五天的访问次数最多的网页

#cat access\_log | awk '{print $7}' | uniq -c | sort -n -r | head -20

# 从日志里查看该ip在干嘛

#cat access\_log | grep 218.66.36.119 | awk '{print $1"\t"$7}' | sort | uniq -c | sort -nr | less

# 列出传输时间超过 30 秒的文件

#cat access\_log | awk '($NF > 30){print $7}' | sort -n | uniq -c | sort -nr | head -20

# 列出最最耗时的页面(超过60秒的)

#cat access\_log | awk '($NF > 60 && $7~/\.php/){print $7}' | sort -n | uniq -c | sort -nr | head -100

# Awk（统计）（打印列求最大值最小值平均值）

1、打印文件的第一列(域)                 ： awk '{print $1}' filename  
2、打印文件的前两列(域)                 ： awk '{print $1,$2}' filename  
3、打印完第一列，然后打印第二列  ： awk '{print $1 $2}' filename  
4、打印文本文件的总行数                ： awk 'END{print NR}' filename  
5、打印文本第一行                          ：awk 'NR==1{print}' filename  
6、打印文本第二行第一列                ：sed -n "2, 1p" filename | awk 'print $1'

test.txt文件内容：  
9  
11  
35  
21  
42  
118

求最大值：

awk 'BEGIN {max = 0} {if ($1+0 > max+0) max=$1} END {print "Max=", max}' test.txt  
Max= 118

求最小值：

awk 'BEGIN {min = 65536} {if ($1+0 < min+0) min=$1} END {print "Min=", min}' test.txt  
Min= 9

求和：

cat test.txt|awk '{sum+=$1} END {print "Sum= ", sum}'  
Sum=  236

求平均值：

cat test.txt|awk '{sum+=$1} END {print "Avg= ", sum/NR}'  
Avg=  39.3333

一. 命令awk简介

　　　　1. awk是一种编程语言，用于对文本和数据进行处理的

　　　　2. 具有强大的文本格式化能力

　　　　3. 利用命令awk，可以将一些文本整理成为我们想要的样子

　　　　4. 命令awk是逐行进行处理的

二. grep、sed、awk的简单比较

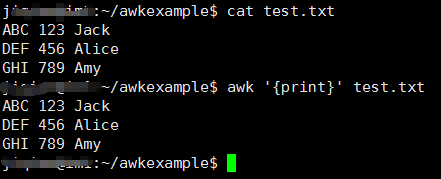
　　　　1. 命令grep，更适合单纯的查找或匹配文本

　　　　2. 命令sed，更适合对匹配到的文本进行编辑

　　　　3. 命令awk，更适合文本格式化，对文本进行较复杂的格式处理

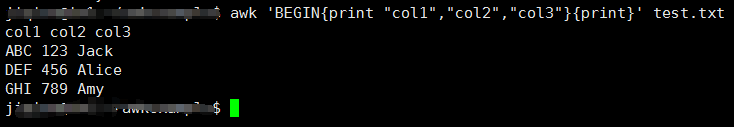
awk [options] 'pattern{action}' file

　　　1. 在没有options和pattern的情况下，使用命令awk

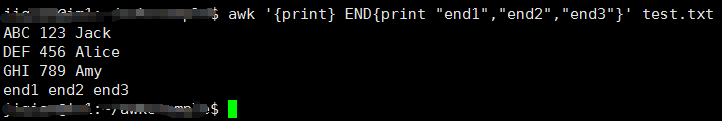


　　 　2. pattern包括两种特殊模式，分别是BEGIN和END

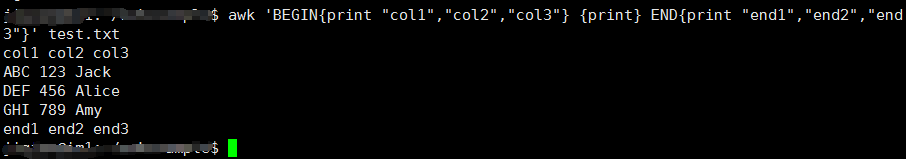
　　　　（1）BEGIN模式，是指命令在处理文本之前执行



　　　　（2）END模式，是指命令在处理文本之后执行

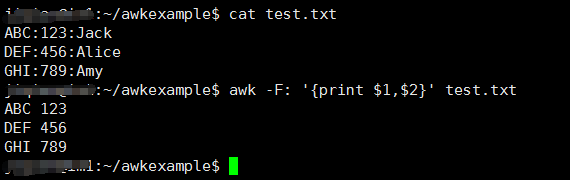


　　　　（3）BEGIN模式和END模式同时存在时，其中，BEGIN与END之间的{}相当于一个循环体，对文件中的每一行进行处理

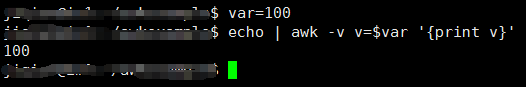


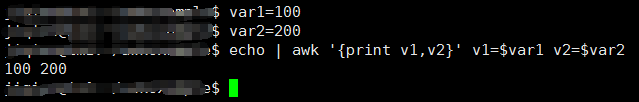
　　3. 常用的参数

　　　　（1）-F，用于指定输入分隔符

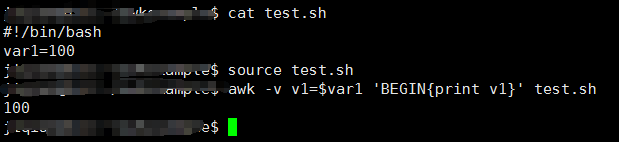


　　　　（2）-v，用于设置变量的值





　　　　　　从文件中输入变量



四. 变量

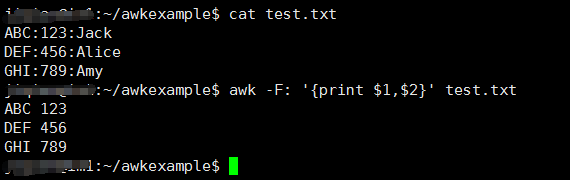
　　awk中的变量分为内置变量和自定义变量两种

　　1. 内置变量

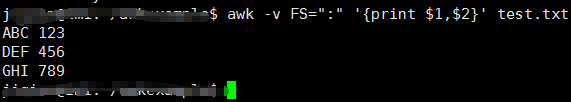
|  |  |
| --- | --- |
| FS | 输入字段分隔符，默认为空白字符 |
| OFS | 输出字段分隔符，默认为空白字符 |
| RS | 输入记录分隔符（输入换行符），指定输入时的换行符 |
| ORS | 输出记录分隔符（输出换行符），指定输出时的换行符 |
| NF | 当前行的字段数（当前行被分隔符分割成了几段） |
| NR | 当前行的行号 |
| FNR | 不同文件分别计数 |
| FILENAME | 当前文件名 |
| ARGV | 数组，保存的是命令行所给定的各参数 |
| ARGC | ARGC数组的个数 |

　　（1）FS：以“：”为字段输入分隔符，输出第1列和第2列

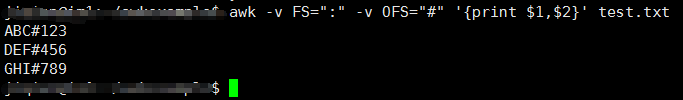
　　　　　　1）使用-F选项指定输入分隔符



　　　　　　2）使用内置变量FS指定输入分隔符，需要注意的是，使用变量时，要使用-v选项来指定对应的变量

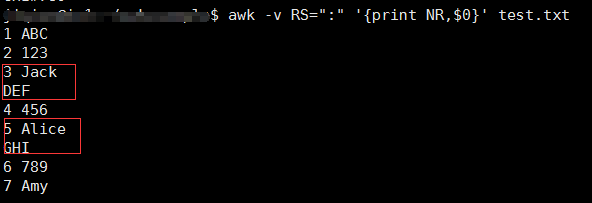


　　　　（2）OFS：以“#”为字段输出分隔符，输出第1列和第2列



　　　　（3）RS：以“：”为行输入分隔符，输出对应的行号和当前行内容。

　　　　　　需要注意的是，两个红框中的内容，由于Jack和DEF、Alice和GHI之间没有“：”，所以在awk中被认作为同一行



　　　　（4）ORS：以“---”为行输出分隔符

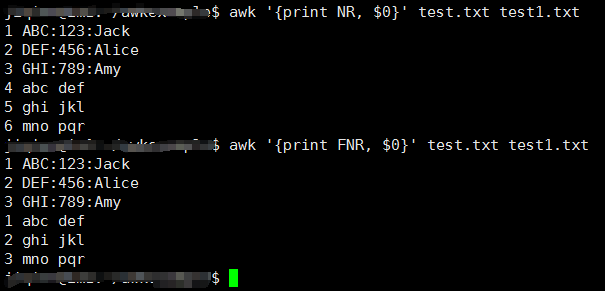
https://images2017.cnblogs.com/blog/1127068/201712/1127068-20171201170340289-1836281852.png

　　　　（5）NF：当前行的字段数

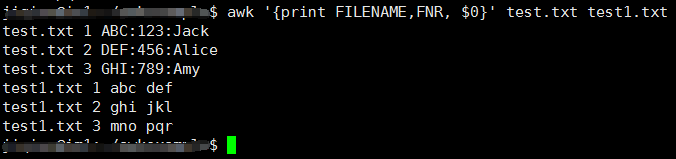
　　　　　　其中，$NF表示的是最后一个字段的内容，$(NF-1)表示的是倒数第二个字段的内容

　　　　（6）NR：当前行号

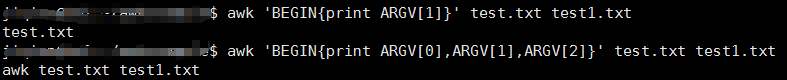
　　　　（7）FNR：不同文件分别计数



　　　　（8）FILENAME：显示当前行的文件名



　　　　（9）ARGV：数组，保存的是命令行所给定的各参数



　　　　（10）ARGC：保存的是ARGV数组的个数

https://images2017.cnblogs.com/blog/1127068/201712/1127068-20171201171905664-27494933.png

　　2. 自定义变量

　　　　（1）使用-v来自定义变量（在上第三节已介绍）

　　　　（2）在awk中直接定义

https://images2017.cnblogs.com/blog/1127068/201712/1127068-20171201172239383-324006650.png

五. 格式化中，awk使用printf时需要注意的问题

　　1. 使用printf输出的文本不会换行，如果需要换行，可以在对应的“格式替换符”后加入“\n”进行转义

　　2. 使用printf输出时，“指定的格式”与“被格式化的文本”之间，要用“，”隔开

　　3. 使用printf输出时，“格式”中的“格式替换符”必须与“被格式化的文本”一一对应（个数要相同）