目录

[**基础知识** 2](#_Toc48595035)

[**单引号、双引号、反引号** 2](#_Toc48595036)

[**命令 -help** 2](#_Toc48595037)

[**常用的命令** 2](#_Toc48595038)

[**ps** 2](#_Toc48595039)

[**find** 2](#_Toc48595040)

[**grep** 2](#_Toc48595041)

[**sed** 2](#_Toc48595042)

[**tail** 2](#_Toc48595043)

[**awk** 2](#_Toc48595044)

[**xargs** 3](#_Toc48595045)

[**scp** 3](#_Toc48595046)

[**shutdown** 3](#_Toc48595047)

[**wc** 3](#_Toc48595048)

[**top** 3](#_Toc48595049)

[**相关概念：** 3](#_Toc48595050)

[**chmod** 5](#_Toc48595051)

[**统计文本中关键字出现的次数** 5](#_Toc48595052)

[**查看指定端口进程** 5](#_Toc48595053)

[**硬盘查看** 5](#_Toc48595054)

[**shell适用的场景** 5](#_Toc48595055)

[**什么样的场景会导致线上CPU负载过高？如何查找原因** 5](#_Toc48595056)

**基础知识**

**单引号、双引号、反引号**

A = ‘you’

echo ‘I love $A’ 🡪 I love $A

echo “I love $A” 🡪 I love you

echo `ls` 🡪 会输出ls命令执行的结果

**命令 -help**

查看命令的使用方法

**常用的命令**

**ps**

查看进程

ps 进程id

ps aux|grep python

**find**

find ./ -name ‘\*.txt’

find ./ -size +500M 查找大小大于500M的文件

find ./ -perm 777 查找权限为777的文件或文件夹

**grep**

grep –i ‘java\*’ f.txt 搜索f.txt中匹配模式的每一行，不区分大小写

-v 🡪 不匹配的行 -n🡪显示匹配的行号

**sed**

sed -n 2p f.txt 打印文件第2行

sed -n 2,5p f.txt 打印文件第2到5行

sed ‘s/root/abc/g’ f,txt 将文件中所有的root替换为abc,不会改变原文件，只会输出改变后的样子，这里的s///可以使用其他分隔符，e.g. s@@@ s### g代表全局替换

sed -n ‘s/root/abc/p’ f.txt 替换…，不改变…,只输出发生改变的行

-n 🡪不输出模式空间内容到屏幕 p🡪打印当前模式空间内容，追加到默认输出之后

-n和p一起出现，表示只打印发生改变的行 -i🡪编辑文件

**tail**

tail -n 10 f.txt 查看文件的后10行

tail +20 f.txt 查看文件从第20行至文件末尾

tail -f f.txt 会把 f.txt文件里的最尾部的内容显示在屏幕上，并且不断刷新，只要 f.txt 更新就可以看到最新的文件内容。

**awk**

awk ‘{print $1,$4}’ f.txt 文本的每一行以空格分割，输出第1和4项  **必须用单引号**

awk -F, ‘{print $1}’ f.txt 指定分割符为逗号

awk -va=1 {print $1,$1+a} 指定变量a

awk ‘$1==2 {print $4}’ f.txt 第一项为2时，打印第4项

awk -f cal.awk f.txt 执行脚本cal.awk格式化f.txt

**xargs**

和管道结合使用，将数据转换成命令行参数，可将单行或多行文本转换为其他格式,e.g.多行变单行，单行变多行

find ./ -size +1M|ls -l 只会输出当前目录，等价于ls -l，不是我们想要的

find ./ -size +1M|xargs ls -l 会输出所有大于1M的文件，包括子目录下的

cat f.txt|xargs 多行变一行

cat f.txt|xargs -n3 三个元素为一行，输出

**scp**

两台电脑之间传输文件。比较慢，但是ssh通道保证了传输的安全性

本地拷贝至远程 scp 本地目录 远程用户名@远程主机ip：远程目录

[远程拷贝至本地 scp cb116@192.168.1.184/disk2/wyl/test.txt /home/wyl](mailto:远程拷贝至本地scp%20-r%20cb116@192.168.1.184/disk2/wyl/test.txt%20%20/home/wyl)

-r 拷贝目录

**shutdown**

shutdown -h now 立即关机

shutdown -r now 立即重启

**wc**

wc -c f.txt 统计字节数

-l统计行数 -m统计字符数 -w统计字数

例如统计当前目录下的文件数 ls -l|wc -l

**top**

**相关概念：**

cpu利用率：cpu的使用情况。

用户时间user time：cpu执行用户进程的时间，包括nices时间。通常期望越高越好。

系统时间system time：cpu在内核运行时间，包括IRQ和softirq时间。系统cpu占用率高，表明系统某部分存在瓶颈，通常期望越低越好。

等待时间waiting time：CPU在等待I/O操作完成所花费的时间。系不部应该花费大量时间等待I/O操作，否则说明I/O存在瓶颈。

空闲时间idle time：系统处于空闲期，等待进程运行。

nice时间 nice time：系统调整进程优先级所花费的时间。

硬中断处理时间 hard irq time：系统处理硬中断所花费的时间，

软中断处理时间 soft irq

丢失时间steal time：悲情值等待虚拟cpu的时间，此时hypercisor在为另一个虚拟处理器服务。。。？

top显示当前系统正在执行的进程的相关信息，包括进程ID、内存占用率、CPU占用率等

top -c 显示完整的command

top -p进程号 指定查看的进程

top -d 0.1 设置刷新时间间隔为0.1s，默认是3s

top - 14:45:40 up 10 days, 38 min, 4 users, load average: 0.00, 0.00, 0.00

任务: 334 total, 1 running, 207 sleeping, 20 stopped, 0 zombie

%Cpu(s): 2.6 us, 1.0 sy, 0.0 ni, 96.3 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 26402392+total, 26257793+free, 670152 used, 775832 buff/cache

KiB Swap: 48827388 total, 48589760 free, 237628 used. 26182446+avail Mem

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

3659 cb116 20 0 43700 3800 3084 R 11.8 0.0 0:00.03 top

1 root 20 0 225808 5000 3204 S 0.0 0.0 0:08.64 systemd

2 root 20 0 0 0 0 S 0.0 0.0 0:00.19 kthreadd

…………

第一行：14:45:40🡪当前系统时间up 10 days,38min🡪系统已运行了10天，在这期间未重启

4 useres🡪当前有4个用户登录系统 load average:0,0,0🡪3个数分别对应1min、5min、15min的负载情况(每5s检查一次活跃的进程，并按照特定算法计算，如果这个数除以逻辑CPU的数量后，结果>5，表明系统超负荷运转)

第二行：各状态进程数 zobie🡪僵尸状态

第三行：cpu状态信息 cpu占用率是根据/proc/stat文件的内容计算而来的。

(我的电脑有4个物理cpu，这里会显示16个逻辑cpu的情况)

cpu时间=user+system+nice+idle+iowait+hardirq+softirq+steal

2.6us🡪用户空间占用cpu的百分比=（user+nice）/cpu时间\*100% 1.0sy🡪内核空间。。=（system+hardirq+softirq）/cpu时间\*100% 0.0ni🡪改变过优先级的进程。。。 96.3id🡪空闲cpu百分比 0.0wa🡪IO等待。。。 0.0hi🡪硬中断。。。 0.0si🡪软中断

第四行：内存状态 单位是kB XX total🡪物理内存总量 XX used🡪使用中的内存总量(指的是现在系统内核控制的内存数，不见得都在使用，还包括过去使用过的，现在可以重复利用的内存) XX free🡪空闲内存总量 XX buffers🡪缓存的内存量

第五行：swap交换分区信息 单位是kB XX total🡪交换区总量 XX used🡪使用的交换区总量 XX free🡪空闲的交换区总量 XX cached🡪缓冲的交换区总量

计算可用内存数：内存free+内存buffer+swap cached

进行内存监控，时刻监督swap used，如果这个数值在不断发生变化，说明内核在不断进行内存和swap的数据交换，这是真正的内存不够用了。

第六行是空行

第七行：各进程的状态监控，包括进程号pid，进程所有者user，进程优先级pr，nice值nice(负值为高优先级，正值为低优先级)，进程使用的虚拟内存总量(kb)virt=swap+res，进程使用的、未被换出的物理内存大小(kb)res=code+data，共享内存大小(kb)shr，进程状态s(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)，从上次更新到现在 的CPU时间占用百分比%CPU，进程使用的物理内存百分比%MEM，进程使用的CPU时间总计(单位1/100秒)，进程名称(命令名/命令行)command。

**chmod**

sudo chmod （-R） 777 文件(夹)名（每个人都有读和写以及执行的权限）

sudo chmod -（代表类型）×××（所有者）×××（组用户）×××（其他用户）

三位数的每一位都表示一个用户类型的权限设置。取值是0～7，即二进制的[000]~[111]。 这个三位的二进制数的每一位分别表示读、写、执行权限。

**一些场景**

**统计文本中关键字出现的次数**

一个文本f.txt，一个关键字a🡪输出a出现的次数

grep -o a f.txt|wc -l

一个文本f.txt,两个关键字a,b🡪输出a或b出现的总次数

grep -o ‘a\|b’ f.txt|wc -l

grep -o 🡪一条数据里有多个相同，会统计相同的次数

grep🡪一条数据里有多个相同，会统计一次次数

当前文件夹下以.py为后缀的文件，每个文件中关键字a出现的次数，过滤不出现的文件，并排序：

find . -name ‘\*.py’|grep -c ‘a’|awk -F ‘:’sort -t ‘:’ -k 2,2nr

**查看指定端口进程**

netstate -tunpl|grep 端口号

-t🡪tcp -u🡪udp n🡪不解析名称 -p🡪展示pid或程序名 –

l🡪展示监听服务器的套接字 -a🡪所有

**硬盘查看**

查看硬盘和分区信息：fdisk -l 需要sudo权限

查看文件系统的磁盘空间占用情况：df -h

查看当前目录的大小 du -sh ./

**shell适用的场景**

1.将一些复杂的命令简单化(平时我们提交一次github代码可能需要很多步骤，但是可以用Shell简化成一步)

2.可以写一些脚本自动实现一个工程中自动更换最新的sdk(库)

3.自动打包、编译、发布等功能

4.清理磁盘中空文件夹

总之一切有规律的活脚本都可以尝试一下

**什么样的场景会导致线上CPU负载过高？如何查找原因**

1. 程序陷入死循环，不停地消耗CPU
2. 线程死锁，线程相互等待，导致假死状态，不停地消耗CPU。
3. 使用top 命令监控当前系统负载情况top命令，找到cpu负载较高的进程。

按下数字1可以看到每个cpu的负载情况，并且在下方会显示该进程开了多少个线程和每个线程的cpu负载等情况。或使用top -H -p pid 查看指定进程中每个线程的资源占用情况。

1. 根据进程id或线程id查看线程堆栈信息

jstack pid: 查看指定进程中线程的堆栈信息，这个命令最终会打印出指定进程的线程堆栈信息，而实际线上情况发生时，我们应当把快速把堆栈信息输出到日志文本中，保留日志信息，然后迅速先重启服务，达到临时缓解服务器压力的目的。e.g. jstack 线程id>xx.log。

然后分析.log文件，.log文件会定位到具体的代码行。

p.s. top打印的pid或id是十进制的，jstack是二进制的。