Linux常用命令知识积累(持续更新)

By Panda

Published 2016-02-25

写在前面

虽然平时大部分工作都是和Java相关的开发, 但是每天都会接触Linux系统, 尤其是使用了Mac之后, 每天都是工作在黑色背景的命令行环境中. 自己记忆力不好, 很多有用的Linux命令不能很好的记忆, 现在逐渐总结一下, 以便后续查看.

基本操作

Linux关机,重启

```
# 关机
shutdown -h now

# 重启
shutdown -r now
```

查看系统,CPU信息

```
# 查看系统内核信息
uname -a

# 查看系统内核版本
cat /proc/version

# 查看当前用户环境变量
env

```

Contents

```
cat /proc/cpuinfo

# 查看有几个逻辑cpu, 包括cpu型
cat /proc/cpuinfo | grep n

# 查看有几颗cpu,每颗分别是几核
cat /proc/cpuinfo | grep p

# 查看当前CPU运行在32bit还是6
getconf LONG_BIT

# 结果大于0, 说明支持64bit计算
cat /proc/cpuinfo | grep f
```


建立软连接

```
ln -s /usr/local/jdk1.8/ jd
```

rpm相关

```
# 查看是否通过rpm安装了该软件
rpm -qa | grep 软件名
```

sshkey

```
# 创建sshkey
ssh-keygen -t rsa -C your_email@example.com

#id_rsa.pub 的内容拷贝到要控制的服务器的 home/username/.ssh/authorized_keys 中,如果没有则新建(.ssh权
```

命令重命名

```
# 在各个用户的.bash_profile中添加重命名配置
alias ll='ls -alF'
```

同步服务器时间

```
sudo ntpdate -u ntp.api.bz
```

后台运行命令

```
# 后台运行,并且有nohup.out输出
nohup xxx &

```

```
# 后台运行，不输出任何日志
nohup xxx > /dev/null &

# 后台运行，并将错误信息做标准输出到日志中
nohup xxx >out.log 2>&1 &
```

强制活动用户退出

```
# 命令来完成强制活动用户退出.其中TTY表示终端名称
pkill -kill -t [TTY]
```

查看命令路径

```
which <命令>
```

查看进程所有打开最大fd数

```
ulimit -n
```

配置dns

```
vim /etc/resolv.conf
```

nslookup,查看域名路由表

```
nslookup google.com
```

last, 最近登录信息列表

```
# 最近登录的5个账号
last -n 5
```

设置固定ip

```
ifconfig em1  192.168.5.177 netmask 255.255.255.0
```

查看进程内加载的环境变量

```
# 也可以去 cd /proc 目录下，查看进程内存中加载的东西
ps eww -p  XXXXX(进程号)
```

查看进程树找到服务器进程

```
ps auwxf
```

查看进程启动路径

```
cd /proc/xxx(进程号)
ls -all
# cwd对应的是启动路径
```

添加用户, 配置sudo权限

```
# 新增用户
useradd 用户名
passwd 用户名

#增加sudo权限
vim /etc/sudoers
# 修改文件里面的
# root    ALL=(ALL)       ALL
# 用户名 ALL=(ALL)       ALL
```

强制关闭进程名包含xxx的所有进程

```
ps aux|grep xxx | grep -v grep | awk '{print $2}' | xargs kill -9
```

磁盘,文件,目录相关操作

vim操作

```
#normal模式下 g表示全局, x表示查找的内容, y表示替换后的内容
:%s/x/y/g

#normal模式下
0  # 光标移到行首(数字0)
$  # 光标移至行尾
shift + g # 跳到文件最后
gg # 跳到文件头

# 显示行号
:set nu

# 去除行号
:set nonu

```

```
# 检索
/xxx(检索内容)  # 从头检索，按n查找下一个
?xxx(检索内容)  # 从尾部检索
```

打开只读文件,修改后需要保存时(不用切换用户即可保存的方式)

```
# 在normal模式下
:w !sudo tee %
```

查看磁盘, 文件目录基本信息

```
# 查看磁盘挂载情况
mount

# 查看磁盘分区信息
df

# 查看目录及子目录大小
du -H -h

# 查看当前目录下各个文件，文件夹占了多少空间，不会递归
du -sh *
```

wc命令

```
# 查看文件里有多少行
wc -l filename

# 看文件里有多少个word
wc -w filename

# 文件里最长的那一行是多少个字
wc -L filename

# 统计字节数
wc -c
```

常用压缩, 解压缩命令

压缩命令

```
tar czvf xxx.tar 压缩目录

zip -r xxx.zip 压缩目录
```

解压缩命令

```
tar zxvf xxx.tar

# 解压到指定文件夹
tar zxvf xxx.tar -C /xxx/yyy/

unzip xxx.zip
```

变更文件所属用户, 用户组

```
chown eagleye.eagleye xxx.log
```

cp, scp, mkdir

```
#复制
cp xxx.log

# 复制并强制覆盖同名文件
cp -f xxx.log

# 复制文件夹
cp -r xxx(源文件夹) yyy(目标文件夹)

# 远程复制
scp -P ssh端口 username@10.10.10.101:/home/username/xxx /home/xxx

# 级联创建目录
mkdir -p /xxx/yyy/zzz

# 批量创建文件夹, 会在test,main下都创建java, resources文件夹
mkdir -p src/{test,main}/{java,resources}
```

比较两个文件

```
diff -u 1.txt 2.txt
```

日志输出的字节数,可以用作性能测试

```
# 如果做性能测试, 可以每执行一次, 往日志里面输出“.”, 这样日志中的字节数就是实际的性能测试运行的次数, 还可以
tail -f xxx.log | pv -bt
```

查看, 去除特殊字符

```
# 查看特殊字符
cat -v xxx.sh

```

```
# 去除特殊字符
sed -i 's/^M//g' env.sh  去除文件的特殊字符，比如^M:  需要这样输入：ctrl+v+enter
```

处理因系统原因引起的文件中特殊字符的问题

```
# 可以转换为该系统下的文件格式
cat file.sh > file.sh_bak

# 先将file.sh中文件内容复制下来然后运行，然后粘贴内容，最后ctrl + d 保存退出
cat > file1.sh

# 在vim中通过如下设置文件编码和文件格式
:set fileencodings=utf-8 ，然后 w （存盘）一下即可转化为 utf8 格式，
:set fileformat=unix

# 在mac下使用dos2unix进行文件格式化
find . -name "*.sh" | xargs dos2unix
```

tee, 重定向的同时输出到屏幕

```
awk '{print $0}' xxx.log | tee test.log
```

检索相关

grep

```
# 反向匹配，查找不包含xxx的内容
grep -v xxx

# 排除所有空行
grep -v '^$'

# 返回结果 2,则说明第二行是空行
grep -n "^$" 111.txt

# 查询以abc开头的行
grep -n "^abc" 111.txt

# 同时列出该词语出现在文章的第几行
grep 'xxx' -n xxx.log

# 计算一下该字串出现的次数
grep 'xxx' -c xxx.log

# 比对的时候，不计较大小写的不同
grep 'xxx' -i xxx.log
```

awk

```
# 以':' 为分隔符,如果第五域有user则输出该行
awk -F ':' '{if ($5 ~ /user/) print $0}' /etc/passwd

# 统计单个文件中某个字符（串）(中文无效)出现的次数
awk -v RS='character' 'END {print --NR}' xxx.txt
```

find检索命令

```
# 在目录下找后缀是.mysql的文件
find /home/eagleye -name '*.mysql' -print

# 会从 /usr 目录开始往下找，找最近3天之内存取过的文件。
find /usr -atime 3 –print

# 会从 /usr 目录开始往下找，找最近5天之内修改过的文件。
find /usr -ctime 5 –print

# 会从 /doc 目录开始往下找，找jacky 的、文件名开头是 j的文件。
find /doc -user jacky -name 'j*' –print

# 会从 /doc 目录开始往下找，找寻文件名是 ja 开头或者 ma开头的文件。
find /doc \( -name 'ja*' -o- -name 'ma*' \) –print

#  会从 /doc 目录开始往下找，找到凡是文件名结尾为 bak的文件，把它删除掉。-exec 选项是执行的意思，rm 是删除
find /doc -name '*bak' -exec rm {} \;
```

网络相关

查看什么进程使用了该端口

```
lsof -i:port
```

获取本机ip地址

```
/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|grep -v inet6|awk '{print $2}'|tr -d "addr:"
```

iptables

```
# 查看iptables状态
service iptables status

# 要封停一个ip
iptables -I INPUT -s ***.***.***.*** -j DROP
```

```
# 要解封一个IP，使用下面这条命令：
iptables -D INPUT -s ***.***.***.*** -j DROP

备注：参数-I是表示Insert（添加），-D表示Delete（删除）。后面跟的是规则，INPUT表示入站，***.***.***.***

#开启9090端口的访问
/sbin/iptables -I INPUT -p tcp --dport 9090 -j ACCEPT

# 防火墙开启、关闭、重启
/etc/init.d/iptables status
/etc/init.d/iptables start
/etc/init.d/iptables stop
/etc/init.d/iptables restart
```

nc命令，tcp调试利器

```
#给某一个endpoint发送TCP请求,就将data的内容发送到对端
nc 192.168.0.11 8000 < data.txt

#nc可以当做服务器，监听某个端口号,把某一次请求的内容存储到received_data里
nc -l 8000 > received_data

#上边只监听一次，如果多次可以加上-k参数
nc -lk 8000
```

tcpdump

```
# dump出本机12301端口的tcp包
tcpdump -i em1 tcp port 12301 -s 1500 -w abc.pcap
```

跟踪网络路由路径

```
# traceroute默认使用udp方式，如果是-I则改成icmp方式
traceroute -I www.163.com

# 从ttl第3跳跟踪
traceroute -M 3 www.163.com

# 加上端口跟踪
traceroute -p 8080 192.168.10.11
```

ss

```
# 显示本地打开的所有端口
ss -l

# 显示每个进程具体打开的socket
ss -pl
```

```
# 显示所有tcp socket
ss -t -a

# 显示所有的UDP Socekt
ss -u -a

# 显示所有已建立的SMTP连接
ss -o state established '( dport = :smtp or sport = :smtp )'

# 显示所有已建立的HTTP连接
ss -o state established '( dport = :http or sport = :http )'

找出所有连接X服务器的进程
ss -x src /tmp/.X11-unix/*

列出当前socket统计信息
ss -s

解释：netstat是遍历/proc下面每个PID目录，ss直接读/proc/net下面的统计信息。所以ss执行的时候消耗资源以及消
```

netstat

```
# 输出每个ip的连接数，以及总的各个状态的连接数
netstat -n | awk '/^tcp/ {n=split($(NF-1),array,":");if(n<=2)++S[array[(1)]];else++S[array[(

# 统计所有连接状态，
# CLOSED：无连接是活动的或正在进行
# LISTEN：服务器在等待进入呼叫
# SYN_RECV：一个连接请求已经到达，等待确认
# SYN_SENT：应用已经开始，打开一个连接
# ESTABLISHED：正常数据传输状态
# FIN_WAIT1：应用说它已经完成
# FIN_WAIT2：另一边已同意释放
# ITMED_WAIT：等待所有分组死掉
# CLOSING：两边同时尝试关闭
# TIME_WAIT：主动关闭连接一端还没有等到另一端反馈期间的状态
# LAST_ACK：等待所有分组死掉
netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'

# 查找较多time_wait连接
netstat -n|grep TIME_WAIT|awk '{print $5}'|sort|uniq -c|sort -rn|head -n20
```

监控linux性能命令

top

```
按大写的 F 或 O 键，然后按 a-z 可以将进程按照相应的列进行排序，然后回车。而大写的 R 键可以将当前的排序倒转
```

列名	含义
PID	进程id
PPID	父进程id
RUSER	Real user name
UID	进程所有者的用户id
USER	进程所有者的用户名
GROUP	进程所有者的组名
TTY	启动进程的终端名。不是从终端启动的进程则显示为 ?
PR	优先级
NI	nice值。负值表示高优先级，正值表示低优先级
P	最后使用的CPU，仅在多CPU环境下有意义
%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，单位秒
TIME+	进程使用的CPU时间总计，单位1/100秒
%MEM	进程使用的物理内存百分比
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
CODE	可执行代码占用的物理内存大小，单位kb
DATA	可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位kb
SHR	共享内存大小，单位kb
nFLT	页面错误次数
nDRT	最后一次写入到现在，被修改过的页面数。
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
WCHAN	若该进程在睡眠，则显示睡眠中的系统函数名
Flags	任务标志，参考 sched.h

dmesg,查看系统日志

```
1 dmesg
```

iostat,磁盘IO情况监控

```
iostat -xz 1

# r/s, w/s, rkB/s, wkB/s: 分别表示每秒读写次数和每秒读写数据量（千字节）。读写量过大，可能会引起性能问题。
# await: IO操作的平均等待时间，单位是毫秒。这是应用程序在和磁盘交互时，需要消耗的时间，包括IO等待和实际操作的
# avgqu-sz: 向设备发出的请求平均数量。如果这个数值大于1，可能是硬件设备已经饱和（部分前端硬件设备支持并行写入
# %util: 设备利用率。这个数值表示设备的繁忙程度，经验值是如果超过60，可能会影响IO性能（可以参照IO操作平均等待
# 如果显示的是逻辑设备的数据，那么设备利用率不代表后端实际的硬件设备已经饱和。值得注意的是，即使IO性能不理想，
```

free,内存使用情况

```
free -m

eg:

                                        total       used       free     shared    buffers
Mem:            1002        769        232          0         62        421
-/+ buffers/cache:                286        715
Swap:           1153          0       1153

第一部分Mem行:
total 内存总数: 1002M
used 已经使用的内存数: 769M
free 空闲的内存数: 232M
shared 当前已经废弃不用,总是0
buffers Buffer 缓存内存数: 62M
cached Page 缓存内存数:421M

关系: total(1002M) = used(769M) + free(232M)

第二部分(-/+ buffers/cache):
(-buffers/cache) used内存数: 286M (指的第一部分Mem行中的used - buffers - cached)
(+buffers/cache) free内存数: 715M (指的第一部分Mem行中的free + buffers + cached)

可见-buffers/cache反映的是被程序实实在在吃掉的内存,而+buffers/cache反映的是可以挪用的内存总数.

第三部分是指交换分区
```

sar,查看网络吞吐状态

```
# sar命令在这里可以查看网络设备的吞吐率。在排查性能问题时，可以通过网络设备的吞吐量，判断网络设备是否已经饱和
sar -n DEV 1

#
# sar命令在这里用于查看TCP连接状态，其中包括:
# active/s: 每秒本地发起的TCP连接数，既通过connect调用创建的TCP连接;
# passive/s: 每秒远程发起的TCP连接数，即通过accept调用创建的TCP连接;
# retrans/s: 每秒TCP重传数量;
# TCP连接数可以用来判断性能问题是否由于建立了过多的连接，进一步可以判断是主动发起的连接，还是被动接受的连接。
sar -n TCP,ETCP 1
```

vmstat, 给定时间监控CPU使用率, 内存使用, 虚拟内存交互, IO读写

```
# 2表示每2秒采集一次状态信息, 1表示只采集一次(忽略既是一直采集)
vmstat 2 1

eg:
r b swpd free buff cache si so bi bo in cs us sy id wa
1 0 0 3499840 315836 3819660 0 0 0 1 2 0 0 0 100 0
0 0 0 3499584 315836 3819660 0 0 0 0 88 158 0 0 100 0
0 0 0 3499708 315836 3819660 0 0 0 2 86 162 0 0 100 0
0 0 0 3499708 315836 3819660 0 0 0 10 81 151 0 0 100 0
1 0 0 3499732 315836 3819660 0 0 0 2 83 154 0 0 100 0
```

- **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。

操作系统 原创 Linux

上一篇：
基于Zipkin的Thrift服务RPC调用链跟踪
下一篇：
2015个人总结

Categories

服务治理 2

监控预警 2

算法 1

随笔 5

Tags

原创 22 分布式 8 Elasticsearch 6 随笔 4 Zookeeper 4 Java 3 异常检测 2 算法 2 SOA 2 Finagle 2 Zipkin 2 业务监控 2 运维 2 自动化 2 一致性 1 调度系统 1 监控监控 1 海淀妇幼 1 刨腹产 1 双胞胎 1

Links

并发编程网
科学松鼠会
Coolshell
36Kr
码农圈
InfoQ
阿里技术沙龙
美团技术博客

RSS

Hello ,I'm Panda.
This is my blog on Github

站长统计