**head命令：**

默认显示头10行。

显示头k行：

head -k /etc/passwd

head -n k /etc/passwd

head -n +k /etc/passwd

显示除最后k行外所有内容：

head -n -k /etc/passwd

**tail命令：**

默认显示最后10行。

显示最后k行：

tail -k /etc/passwd

tail -n k /etc/passwd

tail -n -k /etc/passwd

从开头第k行开始输出：

tail -n +k /etc/passwd

**xargs命令：**

管道是将输出内容重定向到下个命令stdin。

xargs是将输出作为下个命令的参数。

echo "--help" | cat

输出—help。因为—help被放进cat的stdin。

echo "--help" | cat

输出cat的帮助文档。因为—help被放进cat的参数列表。

-d指定分隔符：

echo "11@22" | xargs -d "@" echo -n

输出

11 22。

-n指定每次传几个参数：

echo "11@22@33@44" | xargs -d "@" -n 2 echo

输出：

11 22

33 44

**sort命令：**

默认对第一列按文本升序排序。

-r 按文本方式降序排序

-k 指定排序的列，如: -k 2，按第二列排序

-n 按数字方式排序

-t 指定列分隔符，如: -t:, 指定分隔符为’:’; -t ‘\0’, 指定分割符为NUL

支持按多列排序，如：sort -t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n，给ip地址排序

**uniq命令：**

用于检查及删除文本文件中重复出现的行，当重复的行并不相邻时，uniq 命令是不起作用的，一般与 sort 命令结合使用。

-c或--count 在每列旁边显示该行重复出现的次数

-d或--repeated 仅显示重复出现的行列

-u或--unique 仅显示出一次的行列

-i, --ignore-case 忽略大小写

-s, --skip-chars =N 比较时跳过前N 个字符

-f, --skip-fields=N 比较时跳过前N 列

[**Linux程序前台后台切换**](https://www.cnblogs.com/huanghuanghui/p/9708350.html)**：**

**&**  
加在一个命令的最后，可以把这个命令放到后台执行，如:  
watch -n 10 sh test.sh & #每10s在后台执行一次test.sh脚本

**ctrl + z**  
可以将一个正在前台执行的命令放到后台，并且处于暂停状态。

**jobs**  
查看当前有多少在后台运行的命令  
jobs -l选项可显示所有任务的PID，jobs的状态可以是running, stopped, Terminated。但是如果任务被终止了（kill），shell 从当前的shell环境已知的列表中删除任务的进程标识。

**fg**  
将后台中的命令调至前台继续运行。如果后台中有多个命令，可以用fg %jobnumber（是命令编号，不是进程号）将选中的命令调出。

**bg**  
将一个在后台暂停的命令，变成在后台继续执行。如果后台中有多个命令，可以用bg %jobnumber将选中的命令调出。

**kill**  
法子1：通过jobs命令查看job号（假设为num），然后执行kill %num  
法子2：通过ps命令查看job的进程号（PID，假设为pid），然后执行kill pid  
前台进程的终止：Ctrl+c

**nohup**  
如果让程序始终在后台执行，即使关闭当前的终端也执行（之前的&做不到），这时候需要nohup。该命令可以在你退出帐户/关闭终端之后继续运行相应的进程。关闭中断后，在另一个终端jobs已经无法看到后台跑得程序了，此时利用ps（进程查看命令）

[**rsync命令**](https://www.cnblogs.com/huanghuanghui/p/9708350.html)**：**

rsync 命令是一个远程数据同步工具，可通过 LAN/WAN 快速同步多台主机间的文件。rsync 使用 rsync 算法来使本地和远程两个主机之间的文件达到同步，这个算法只传送两个文件的不同部分，而不是每次都整份传送，因此速度相当快。可以用于本地拷贝，也可用于远端拷贝，用法与scp相似。

<https://www.jianshu.com/p/258ceb7b2223>

|  |  |
| --- | --- |
| -v，--verbose | 详细模式输出。 |
| -q，--quiet | 精简输出模式。 |
| -c，--checksum | 打开校验开关，强制对文件传输进行校验。 |
| -a，--archive | 归档模式，表示以递归方式传输文件，并保持所有文件属性，等于 -rlptgoD。 |

**Free命令：**

$ free -h

total used free shared buffers cached

Mem: 252G 139G 112G 19G 154M 105G

-/+ buffers/cache: 33G 218G

Swap: 0B 0B 0B

第一部分Mem行：

total：内存总数

used：已经使用的内存数

free：空闲的内存数

shared：可用的共享内存

buffers：内存缓冲数

cached：内存缓存数

第二部分:(-/+ buffers/cache)

used: 除去被用作buffers和cache内存后已用的内存

free: 用作buffers和cache的内存加上Mem部分空闲的内存数

第三部分:（Swap）

用一部分磁盘当做内存用的“内存”

**free 与 available 的区别**

free 是真正尚未被使用的物理内存数量。**通常看起来很低，因为linux会尽量使用空闲内存做缓存**；  
available 是应用程序认为可用内存数量，available = free + buffer + cache (注：只是大概的计算方法)

**Buffer与cache的区别：**

cache：文件系统层级的缓存，从磁盘里读取的内容是存储到这里，这样程序读取磁盘内容就会非常快，比如使用grep和find等命令查找内容和文件时，第一次会慢很多，再次执行就快好多倍，几乎是瞬间。但如上所说，如果对文件的更新不关心，就没必要清cache，否则如果要实施同步，必须要把内存空间中的cache clean下

buffer：磁盘等块设备的缓冲，内存的这一部分是要写入到磁盘里的。这种情况需要注意，位于内存buffer中的数据不是即时写入磁盘，而是系统空闲或者buffer达到一定大小统一写到磁盘中，所以断电易失，为了防止数据丢失所以我们最好正常关机或者多执行几次sync命令，让位于buffer上的数据立刻写到磁盘里。

**按我的理解，就是buffer存储文件的inode内容，cache存储文件的node内容。**

**Explained by RedHat:**

**Cache Pages:**

A cache is the part of the memory which transparently stores data so that future requests for that data can be served faster. This memory is utilized by the kernel to cache disk data and improve i/o performance.

The Linux kernel is built in such a way that it will use as much RAM as it can to cache information from your local and remote filesystems and disks. As the time passes over various reads and writes are performed on the system, kernel tries to keep data stored in the memory for the various processes which are running on the system or the data that of relevant processes which would be used in the near future. The cache is not reclaimed at the time when process get stop/exit, however when the other processes requires more memory then the free available memory, kernel will run heuristics to reclaim the memory by storing the cache data and allocating that memory to new process.

When any kind of file/data is requested then the kernel will look for a copy of the part of the file the user is acting on, and, if no such copy exists, it will allocate one new page of cache memory and fill it with the appropriate contents read out from the disk.

The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere in the disk. When some data is requested, the cache is first checked to see whether it contains that data. The data can be retrieved more quickly from the cache than from its source origin.

SysV shared memory segments are also accounted as a cache, though they do not represent any data on the disks. One can check the size of the shared memory segments using ipcs -m command and checking the bytes column.

**Buffers :**

Buffers are the disk block representation of the data that is stored under the page caches. Buffers contains the metadata of the files/data which resides under the page cache. Example: When there is a request of any data which is present in the page cache, first the kernel checks the data in the buffers which contain the metadata which points to the actual files/data contained in the page caches. Once from the metadata the actual block address of the file is known, it is picked up by the kernel for processing.

如何释放缓存：

<https://www.cnblogs.com/M18-BlankBox/p/5326484.html>

**查看进程的内存使用：**

top -p 2913

ps aux 第3、4、6列分别是cpu使用率、内存使用率和内存使用量(kB)

cat /proc/{pid}/status的VmRSS列

**cut命令：**

显示文本行中的指定部分。

-d：指定字段的分隔符，默认的字段分隔符为“TAB”；

-f：显示指定字段的内容；

cut –f1 // 显示第一个字段

cut –f2,3 // 显示第2、3个字段

cut –f2-5 // 显示2-5字段

cut –f5- // 显示从5字段开始的所有字段

cut –f-6 // 显示从6往前的所有字段

**Linux计算MD5和Sha1的命令**

MD5

MD5即Message-Digest Algorithm 5（信息-摘要算法 5），用于确保信息传输完整一致。

Sha1

安全散列算法（英语：Secure Hash Algorithm）是一种能计算出一个数位讯息所对应到的，长度固定的字串（又称讯息摘要）的算法。且若输入的讯息不同，它们对应到不同字串的机率很高；而SHA是FIPS所认证的五种安全散列算法。

MD5 与 SHA1 是当前最常用的两种哈希算法。

计算文件的 MD5 – md5sum

md5sum OurUnix.tar

计算文件的 SHA1 – sha1sum

sha1sum OurUnix.tar

**命令直接读取标准输入**

cut -c3 /dev/stdin

shell会提示你输入内容，直到你按下ctrl+d结束输入。然后cut命令将输入的每一行内容并截取第3个字符以输出。

**Shell 输入/输出重定向**

|  |  |
| --- | --- |
| **命令** | **说明** |
| command > file | 将输出重定向到 file。 |
| command < file | 将输入重定向到 file。 |
| command >> file | 将输出以追加的方式重定向到 file。 |
| n > file | 将文件描述符为 n 的文件重定向到 file。 |
| &>file | 把标准输出和标准错误输出都重定向到文件file中 |
| n >> file | 将文件描述符为 n 的文件以追加的方式重定向到 file。 |
| n >&m | 将输出文件 m 和 n 合并。 |
| n <&m | 将输入文件 m 和 n 合并。 |
| << delimiter | 将开始标记 delimiter和结束标记 delimiter之间的内容作为输入，  就是所谓的**Here Document** |

Here Document 是 Shell 中的一种特殊的重定向方式，用来将输入重定向到一个交互式 Shell 脚本或程序。基本形式如下：

command << delimiter

document

delimiter

它的作用是将两个 delimiter 之间的内容(document) 作为输入传递给 command。

**注意：**

**结尾的delimiter 一定要顶格写，前面不能有任何字符，后面也不能有任何字符，包括空格和 tab 缩进。**

**开始的delimiter前后的空格会被忽略掉。**

例如：

$ cat << EOF

欢迎来到

菜鸟教程

www.runoob.com

EOF

输出结果：

欢迎来到

菜鸟教程

www.runoob.com

详见：

<https://www.runoob.com/linux/linux-shell-io-redirections.html>

**ssh命令的坑**

while read -r gpu\_host

do

ssh $gpu\_host -C "sh sync\_image\_process\_model.sh"

done < gpu\_host\_list.txt

运行时发现执行完第一条ssh就退出了，没有遍历文件gpu\_host\_list.txt中的所有行。**原因在于ssh会读取标准输入中的所有内容，这样while就读不到了**。

我们只需要将ssh的标准输入重定向即可解决问题:

ssh $gpu\_host -C "sh sync\_image\_process\_model.sh" < /dev/null

**tree命令**

以树状图列出目录的内容，包括子目录里的文件。

-a 显示所有文件和目录。

-A 使用ASNI绘图字符显示树状图而非以ASCII字符组合。

-C 在文件和目录清单加上色彩，便于区分各种类型。

-d 显示目录名称而非内容。

-D 列出文件或目录的更改时间。

-f 在每个文件或目录之前，显示完整的相对路径名称。

-F 在执行文件，目录，Socket，符号连接，管道名称名称，各自加上"\*","/","=","@","|"号。

-g 列出文件或目录的所属群组名称，没有对应的名称时，则显示群组识别码。

-i 不以阶梯状列出文件或目录名称。

-L level 限制目录显示层级。

-l 如遇到性质为符号连接的目录，直接列出该连接所指向的原始目录。

-n 不在文件和目录清单加上色彩。

-N 直接列出文件和目录名称，包括控制字符。

-p 列出权限标示。

-P<范本样式> 只显示符合范本样式的文件或目录名称。

-q 用"?"号取代控制字符，列出文件和目录名称。

-s 列出文件或目录大小。

-t 用文件和目录的更改时间排序。

-u 列出文件或目录的拥有者名称，没有对应的名称时，则显示用户识别码。

-x 将范围局限在现行的文件系统中，若指定目录下的某些子目录，其存放于另一个文件系统上，则将该子目录予以排除在寻找范围外。

**修改软链接**

ln -snf [新的源文件或目录]  [目标文件或目录]

-f选项是强制删除已经存在目标。

**basename和dirname命令**

basename获取路径中的最后一级，如：

basename ~/ks\_svn/ks/gen\_makefile.sh

输出：

gen\_makefile.sh

basename ~/ks\_svn/ks/

输出：

ks

dirname获取父目录，如：

dirname ~/ks\_svn/ks/gen\_makefile.sh

输出：

/Users/yangwen/ks\_svn/ks

dirname ~/ks\_svn/ks/

输出：

/Users/yangwen/ks\_svn

**Shell中的变量：­­­­­**

IFS(Internal Field Separator):内置分隔符，类似awk中的FS。

**vimdiff命令：**

比较多个文件的不同。

]c: 跳转到下一个差异点;

[c: 跳转到上一个差异点;

命令前加上数字可以跳过多个差异点；

:qa （quit all）同时退出；

:wa （write all）保存全部文件；

:wqa （write, then quit all）保存全部文件，然后退出；

:qa! （force to quit all）退出的时候不希望保存任何操作的结果；

dp （diff "put"）当前文件的内容复制到另一个文件里；

do (diff "get") 另一个文件内容复制到当前文件；

zo （folding open，之所以用z这个字母，是因为它看上去比较像折叠着的纸）展开被折叠的文本行；

zc （folding close）重新折叠；

**rsync命令：**

rsync（remote synchronize）是一个远程数据同步工具，可通过网络快速同步多台主机之间的文件，也可以使用 rsync 同步本地硬盘中的不同目录。  
rsync 使用所谓的 rsync算法 进行数据同步，这种算法只传送两个文件的不同部分，而不是每次都整份传送，因此速度相当快。 可以参考 How Rsync Works A Practical Overview 进一步了解 rsync 的运作机制。  
rsync 的初始作者是 Andrew Tridgell 和 Paul Mackerras，目前由 [http://rsync.samba.org](http://rsync.samba.org/) 维护。

rsync 具有如下的基本特性：

可以镜像保存整个目录树和文件系统

可以很容易做到保持原来文件的权限、时间、软硬链接等

可以使用 rsh、ssh 方式来传输文件，也可以通过直接的 socket 连接

无论本地同步目录还是远程同步数据，首次运行时将会把全部文件拷贝一次，以后再运行时将只拷贝有变化的文件（对于新文件）或文件的变化部分（对于原有文件）。

rsync 的命令格式为：

1）本地使用：

rsync [OPTION...] SRC... [DEST]

2）通过远程 Shell 使用：

拉: rsync [OPTION...] [USER@]HOST:SRC... [DEST]

推: rsync [OPTION...] SRC... [USER@]HOST:DEST

3）访问 rsync 服务器:

拉: rsync [OPTION...] [USER@]HOST::SRC... [DEST]

推: rsync [OPTION...] SRC... [USER@]HOST::DEST

拉: rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]

推: rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST

常用选项：

|  |  |
| --- | --- |
| **选项** | **说明** |
| -a, ––archive | 归档模式，表示以递归方式传输文件，并保持所有文件属性，等价于 -rlptgoD (注意不包括 -H) |
| -r, ––recursive | 对子目录以递归模式处理 |
| -l, ––links | 保持符号链接文件 |
| -H, ––hard-links | 保持硬链接文件 |
| -p, ––perms | 保持文件权限 |
| -t, ––times | 保持文件时间信息 |
| -g, ––group | 保持文件属组信息 |
| -o, ––owner | 保持文件属主信息 (super-user only) |
| -D | 保持设备文件和特殊文件 (super-user only) |
| -z, ––compress | 在传输文件时进行压缩处理 |
| ––exclude=PATTERN | 指定排除一个不需要传输的文件匹配模式 |
| ––exclude-from=FILE | 从 FILE 中读取排除规则 |
| ––include=PATTERN | 指定需要传输的文件匹配模式 |
| ––include-from=FILE | 从 FILE 中读取包含规则 |
| ––copy-unsafe-links | 拷贝指向SRC路径目录树以外的链接文件 |
| ––safe-links | 忽略指向SRC路径目录树以外的链接文件（默认） |
| ––existing | 仅仅更新那些已经存在于接收端的文件，而不备份那些新创建的文件 |
| ––ignore-existing | 忽略那些已经存在于接收端的文件，仅备份那些新创建的文件 |
| -b, ––backup | 当有变化时，对目标目录中的旧版文件进行备份 |
| ––backup-dir=DIR | 与 -b 结合使用，将备份的文件存到 DIR 目录中 |
| ––link-dest=DIR | 当文件未改变时基于 DIR 创建硬链接文件 |
| ––delete | 删除那些接收端还有而发送端已经不存在的文件 |
| ––delete-before | 接收者在传输之前进行删除操作 (默认) |
| ––delete-during | 接收者在传输过程中进行删除操作 |
| ––delete-after | 接收者在传输之后进行删除操作 |
| ––delete-excluded | 在接收方同时删除被排除的文件 |
| -e, ––rsh=COMMAND | 指定替代 rsh 的 shell 程序 |
| ––ignore-errors | 即使出现 I/O 错误也进行删除 |
| ––partial | 保留那些因故没有完全传输的文件，以是加快随后的再次传输 |
| ––progress | 在传输时显示传输过程 |
| -P | 等价于 ––partial ––progress |
| ––delay-updates | 将正在更新的文件先保存到一个临时目录（默认为 “.~tmp~”），待传输完毕再更新目标文件 |
| -v, ––verbose | 详细输出模式 |
| -q, ––quiet | 精简输出模式 |
| -h, ––human-readable | 输出文件大小使用易读的单位（如，K，M等） |
| -n, ––dry-run | 显示哪些文件将被传输 |
| ––list-only | 仅仅列出文件而不进行复制 |
| ––rsyncpath=PROGRAM | 指定远程服务器上的 rsync 命令所在路径 |
| ––password-file=FILE | 从 FILE 中读取口令，以避免在终端上输入口令，通常在 cron 中连接 rsync 服务器时使用 |
| -4, ––ipv4 | 使用 IPv4 |
| -6, ––ipv6 | 使用 IPv6 |
| ––version | 打印版本信息 |
| ––help | 显示帮助信息 |

若使用普通用户身份运行 rsync 命令，默认同步后的文件属主将变为这个普通用户。

若使用超级用户身份运行 rsync 命令，默认同步后的文件属主将保持原来的用户。

**使用 --exclude/--include 选项**

可以使用 ––exclude 选项排除源目录中要传输的文件；同样地，也可以使用 ––include 选项指定要传输的文件。  
例如：下面的 rsync 命令将 192.168.0.101 主机上的 /www 目录（不包含 /www/logs 和 /www/conf子目录）复制到本地的 /backup/www/ 。

# rsync -vzrtopg --delete --exclude "logs/" --exclude "conf/" --progress \

backup@192.168.0.101:/www/ /backup/www/

又如：下面的 rsync 命令仅复制目录结构而忽略掉目录中的文件。

# rsync -av --include '\*/' --exclude '\*' \

backup@192.168.0.101:/www/ /backup/www-tree/

选项 ––include 和 ––exclude 都不能使用间隔符。例如：

--exclude "logs/" --exclude "conf/"

不能写成

--exclude "logs/ conf/"

**使用 --exclude-from/--include-from 选项**

当 include/exclude 的规则较复杂时，可以将规则写入规则文件。使用规则文件可以灵活地选择传输哪些文件（include）以及忽略哪些文件（exclude）。

若文件/目录在剔除列表中，则忽略传输

若文件/目录在包含列表中，则传输之

若文件/目录未被提及，也传输之

在 rsync 的命令行中使用 ––exclude-from=FILE 或 ––include-from=FILE 读取规则文件。

规则文件 FILE 的书写约定：

每行书写一条规则 RULE

以 # 或 ; 开始的行为注释行

包含（include）和排除（exclude）规则的语法如下：

include PATTERN 或简写为 + PATTERN

exclude PATTERN 或简写为 - PATTERN

PATTERN 的书写规则如下：

以 / 开头：匹配被传输的跟路径上的文件或目录

以 / 结尾：匹配目录而非普通文件、链接文件或设备文件

使用通配符

\*：匹配非空目录或文件（遇到 / 截止）

\*\*：匹配任何路径（包含 / ）

?：匹配除了 / 的任意单个字符

[：匹配字符集中的任意一个字符，如 [a-z] 或 [[:alpha:]]

可以使用转义字符 \ 将上述通配符还原为字符本身含义

下面给出几个使用规则的例子：  
例1：

# 不传输所有后缀为 .o 的文件

- \*.o

# 不传输传输根目录下名为 foo 的文件或目录

- /foo

# 不传输名为 foo 的目录

- foo/

# 不传输 /foo 目录下的名为 bar 的文件或目录

- /foo/bar

例2：

# 传输所有目录和C语言源文件并禁止传输其他文件

+ \*/

+ \*.c

- \*

例3：

# 仅传输 foo 目录和其下的 bar.c 文件

+ foo/

+ foo/bar.c

- \*

将规则写入规则文件之后，如何在命令行上使用它呢？下面给出一个例子：  
首先将下面的规则存入名为 www-rsync-rules 的文件

# 不传输 logs 目录

- logs/

# 不传输后缀为 .tmp 的文件

- \*.tmp

# 传输 Apache 虚拟主机文档目录（/\*/ 匹配域名）

+ /srv/www/

+ /srv/www/\*/

+ /srv/www/\*/htdocs/

+ /srv/www/\*/htdocs/\*\*

# 传输每个用户的 public\_html 目录（/\*/ 匹配用户名）

+ /home/

+ /home/\*/

+ /home/\*/public\_html/

+ /home/\*/public\_html/\*\*

# 禁止传输其他

- \*

然后即可使用类似如下的 rsync 命令：

rsync -av --delete --exclude-from=www-rsync-rules / remotehost:/dest/dir

更多详情见(详解RSYNC)

<https://blog.csdn.net/lianzg/article/details/24817087>

**pssh命令：**

pssh是python写的可以并发在多台机器上批量执行命令的工具。

参数列表：

--version show program's version number and exit

--help show this help message and exit

-h HOST\_FILE, --hosts=HOST\_FILE

hosts file (each line "[user@]host[:port]")

-H HOST\_STRING, --host=HOST\_STRING

additional host entries ("[user@]host[:port]")

-l USER, --user=USER username (OPTIONAL)

-p PAR, --par=PAR max number of parallel threads (OPTIONAL)

-o OUTDIR, --outdir=OUTDIR

output directory for stdout files (OPTIONAL)

-e ERRDIR, --errdir=ERRDIR

output directory for stderr files (OPTIONAL)

-t TIMEOUT, --timeout=TIMEOUT

timeout (secs) (0 = no timeout) per host (OPTIONAL)

-O OPTION, --option=OPTION

SSH option (OPTIONAL)

-v, --verbose turn on warning and diagnostic messages (OPTIONAL)

-A, --askpass Ask for a password (OPTIONAL)

-x ARGS, --extra-args=ARGS

Extra command-line arguments, with processing for

spaces, quotes, and backslashes

-X ARG, --extra-arg=ARG

Extra command-line argument

-i, --inline inline aggregated output and error for each server

--inline-stdout inline standard output for each server

-I, --send-input read from standard input and send as input to ssh

-P, --print print output as we get it

如：

cci\_serving\_deploy/op\_root/pssh/pssh -p 2 -t 0 --hosts=restart\_host\_list.txt /media/disk1/fordata/web\_server/project//m2u\_image\_process\_server/bin/load.sh reload

**ldd命令：**

列出binary依赖的链接库：

ldd {binary}

(1) ldd是一个shell脚本;

(2) ldd显示可执行模块的dependency的工作原理，其实质是通过ld-linux.so（elf动态库的装载器）来实现的。实际上可以直接执行ld-linux.so模块，如：

/lib64/ld-linux-x86-64.so.2 --list {binary}

在 Linux 下面，共享库的寻找和加载是由 /lib/ld.so 实现的。 ld.so 在标准路经(/lib, /usr/lib, /lib64, /usr/lib64) 中寻找应用程序用到的共享库。

如果需要用到的共享库在非标准路经，Linux 通用的做法是将非标准路经加入 /etc/ld.so.conf，然后运行 ldconfig 生成 /etc/ld.so.cache。 ld.so 加载共享库的时候，会从 ld.so.cache 查找。

Linux 的先辈 Unix 还有一个环境变量：LD\_LIBRARY\_PATH 来处理非标准路经的共享库。ld.so 加载共享库的时候，也会查找这个变量所设置的路经。

**Linux下动态链接库版本管理及查找加载方式**

**Linux下so的版本机制介绍**

在linux系统上执行 ls -l /usr/lib64，会看到很多具有下列特征的软链接,其中x、y、z为数字, 那么这些软连接和他们后面的数字有什么用途呢？

libfoo.so -> libfoo.so.x  
libfoo.so.x -> libfoo.so.x.y.z  
libbar.so.x -> libbar.so.x.y

x,y,z分别代表的是这个so的主版本号(MAJOR)，次版本号(MINOR)，以及发行版本号(RELEASE),对于这三个数字各自的含义，以及什么时候会进行增长，不同的文献上有不同的解释，不同的组织遵循的规定可能也有细微的差别，但有一个可以肯定的事情是：主版本号(MAJOR)不同的两个so库，所暴露出的API接口是不兼容的。而对于次版本号，和发行版本号，则有着不同定义，其中一种定义是：次要版本号表示API接口的定义发生了改变（比如参数的含义发生了变化），但是保持向前兼容；而发行版本号则是函数内部的一些功能调整、优化、BUG修复，不涉及API接口定义的修改。

**vmstat命令：**

vmstat是Virtual Meomory Statistics（虚拟内存统计）的缩写，可对操作系统的虚拟内存、进程、CPU活动进行监控。他是对系统的整体情况进行统计，不足之处是无法对某个进程进行深入分析。vmstat 工具提供了一种低开销的系统性能观察方式。因为 vmstat 本身就是低开销工具，在非常高负荷的服务器上，你需要查看并监控系统的健康情况,在控制窗口还是能够使用vmstat 输出结果。

-a：显示活跃和非活跃内存

-f：显示从系统启动至今的fork数量 。

-m：显示slabinfo

-n：只在开始时显示一次各字段名称。

-s：显示内存相关统计信息及多种系统活动数量。

delay：刷新时间间隔。如果不指定，只显示一条结果。

count：刷新次数。如果不指定刷新次数，但指定了刷新时间间隔，这时刷新次数为无穷。

-d：显示磁盘相关统计信息。

-p：显示指定磁盘分区统计信息

-S：使用指定单位显示。参数有 k 、K 、m 、M ，分别代表1000、1024、1000000、1048576字节（byte）。默认单位为K（1024 bytes）

-V：显示vmstat版本信息。

如：

vmstat 5 5

表示在5秒时间内进行5次采样。将得到一个数据汇总他能够反映真正的系统情况。

**iostat命令：**

iostat是I/O statistics（输入/输出统计）的缩写，iostat工具将对系统的磁盘操作活动进行监视。它的特点是汇报磁盘活动统计情况，同时也会汇报出CPU使用情况。同vmstat一样，iostat也有一个弱点，就是它不能对某个进程进行深入分析，仅对系统的整体情况进行分析。iostat属于sysstat软件包。可以用yum install sysstat 直接安装。

-c 显示CPU使用情况

-d 显示磁盘使用情况

-k 以 KB 为单位显示

-m 以 M 为单位显示

-N 显示磁盘阵列(LVM) 信息

-n 显示NFS 使用情况

-p[磁盘] 显示磁盘和分区的情况

-t 显示终端和CPU的信息

-x 显示详细信息

-V 显示版本信息

**pidstat命令：**

pidstat是sysstat工具的一个命令，用于监控全部或指定进程的cpu、内存、线程、设备IO等系统资源的占用情况。pidstat首次运行时显示自系统启动开始的各项统计信息，之后运行pidstat将显示自上次运行该命令以后的统计信息。用户可以通过指定统计的次数和时间来获得所需的统计信息。

pidstat [ 选项 ] [ <时间间隔> ] [ <次数> ]

常用的参数：

-u：默认的参数，显示各个进程的cpu使用统计

-r：显示各个进程的内存使用统计

-d：显示各个进程的IO使用情况

-p：指定进程号

-w：显示每个进程的上下文切换情况

-t：显示选择任务的线程的统计信息外的额外信息

-T { TASK | CHILD | ALL }

这个选项指定了pidstat监控的。TASK表示报告独立的task，CHILD关键字表示报告进程下所有线程统计信息。ALL表示报告独立的task和task下面的所有线程。

注意：task和子线程的全局的统计信息和pidstat选项无关。这些统计信息不会对应到当前的统计间隔，这些统计信息只有在子线程kill或者完成的时候才会被收集。

-V：版本号

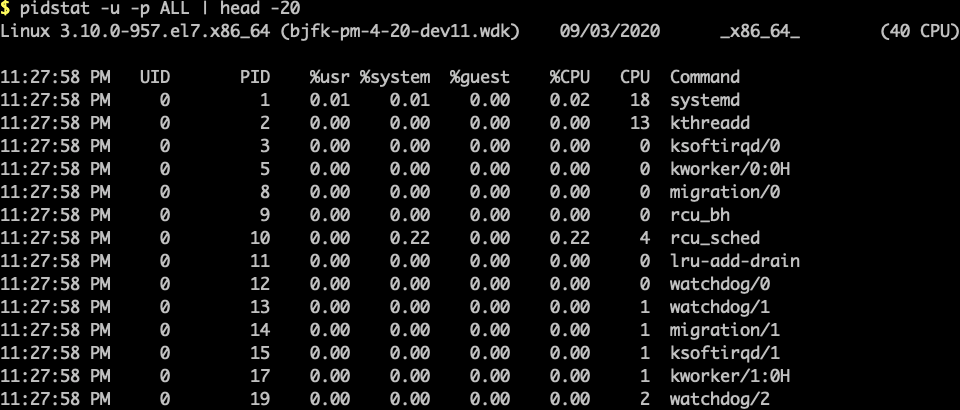
-h：在一行上显示了所有活动，这样其他程序可以容易解析。

-I：在SMP环境，表示任务的CPU使用率/内核数量

-l：显示命令名和所有参数

**查看所有进程的CPU使用情况：**

pidstat -u -p ALL



PID：进程ID

%usr：进程在用户空间占用cpu的百分比

%system：进程在内核空间占用cpu的百分比

%guest：进程在虚拟机占用cpu的百分比

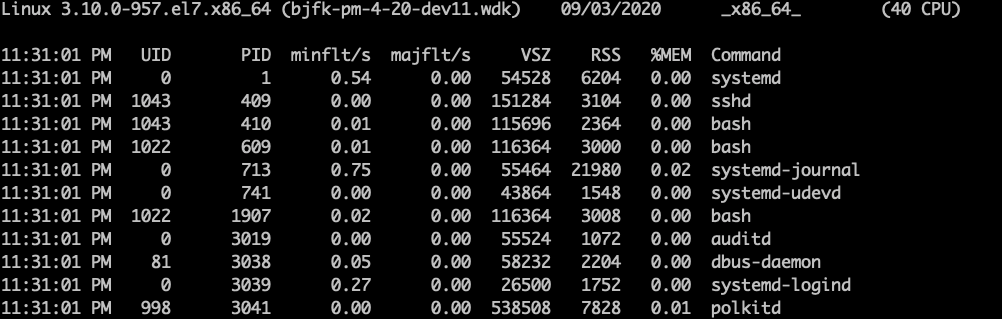
%CPU：进程占用cpu的百分比

CPU：处理进程的cpu编号

Command：当前进程对应的命令

**查看进程的内存使用情况：**

pidstat -r



Minflt/s: 每秒发生的cache缺页中断，不需要从磁盘中加载页

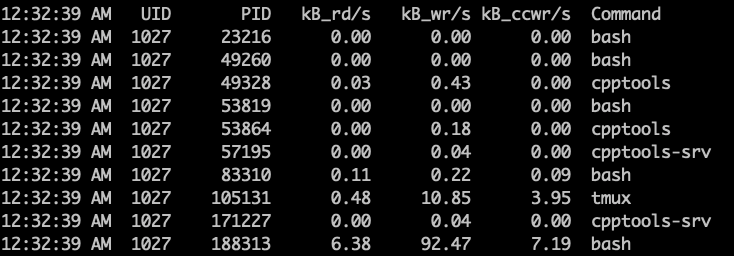
Majflt/s: 每秒发生的cache缺页中断，需要从磁盘中加载页

VSZ：虚拟地址大小，虚拟内存的使用KB

RSS：常驻集合大小，非交换区五里内存使用KB

**查看进程的IO使用情况：**

pidstat -d



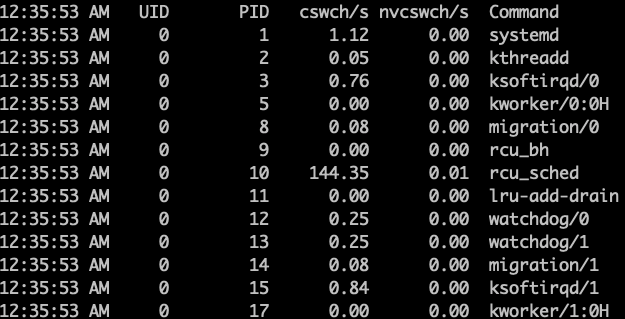
kB\_rd/s：每秒从磁盘读取的KB

kB\_wr/s：每秒写入磁盘KB

kB\_ccwr/s：任务取消的写入磁盘的KB。当任务截断脏的pagecache的时候会发生。

**查看进程上下文切换情况：**

pidstat -w

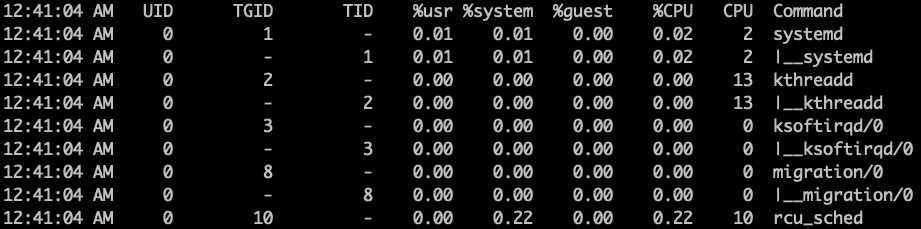


Cswch/s: 每秒主动任务上下文切换数量，如：进程需要等待资源就绪

Nvcswch/s: 每秒被动任务上下文切换数量，如：时间片耗尽

**查看进程的线程的额外信息：**

pidstat -t



TGID:线程group leader线程id

TID:线程id

%usr：进程在用户空间占用cpu的百分比

%system：进程在内核空间占用cpu的百分比

%guest：进程在虚拟机占用cpu的百分比

%CPU：进程占用cpu的百分比

CPU：处理进程的cpu编号

-T { TASK | CHILD | ALL }

This option specifies what has to be monitored by the pidstat command. The TASK keyword indicates that statistics are to be reported for individual tasks (this is the default option) whereas the CHILD keyword indicates that statistics are to be globally reported for the selected tasks and all their children. The ALL keyword indicates that statistics are to be reported for individual tasks and globally for the selected tasks and their children.

Note: Global statistics for tasks and all their children are not available for all options of pidstat. Also these statistics are not necessarily relevant to current time interval: The statistics of a child process are collected only when it finishes or it is killed.

**watch命令：**

watch是一个非常实用的命令，基本所有的Linux发行版都带有这个小工具，如同名字一样，watch可以帮你监测一个命令的运行结果，省得你一遍遍的手动运行。在Linux下，watch是周期性的执行下个程序，并全屏显示执行结果。你可以拿他来监测你想要的一切命令的结果变化，比如 tail 一个 log 文件，ls 监测某个文件的大小变化等。

-n或--interval  缺省每2秒运行一下程序，可以用-n或-interval来指定间隔的时间。

-d或--differences  用-d或--differences 选项watch 会高亮显示变化的区域。 而-d=cumulative选项会把变动过的地方(不管最近的那次有没有变动)都高亮显示出来。

-t 或-no-title  会关闭watch命令在顶部的时间间隔,命令，当前时间的输出。

实例1：

命令：每隔一秒高亮显示网络链接数的变化情况

watch -n 1 -d netstat -ant  
说明：

其它操作：  
切换终端： Ctrl+x  
退出watch：Ctrl+g

实例2：每隔一秒高亮显示http链接数的变化情况

命令：

watch -n 1 -d 'pstree|grep http'

说明：

每隔一秒高亮显示http链接数的变化情况。 后面接的命令若带有管道符，需要加''将命令区域归整。

实例3：实时查看模拟攻击客户机建立起来的连接数

命令：

watch 'netstat -an | grep:21 | \ grep<模拟攻击客户机的IP>| wc -l'

说明：

实例4：监测当前目录中 scf' 的文件的变化

命令：

watch -d 'ls -l|grep scf'

实例5：10秒一次输出系统的平均负载

命令：

watch -n 10 'cat /proc/loadavg'

[**at命令**](https://www.cnblogs.com/peida/archive/2013/01/05/2846152.html)

在指定的时间执行指定任务，只能执行一次。

-m 当指定的任务被完成之后，将给用户发送邮件，即使没有标准输出

-I atq的别名

-d atrm的别名

-v 显示任务将被执行的时间

-c 打印任务的内容到标准输出

-V 显示版本信息

-q<列队> 使用指定的列队

-f<文件> 从指定文件读入任务而不是从标准输入读入

-t<时间参数> 以时间参数的形式提交要运行的任务

at允许使用一套相当复杂的指定时间的方法。他能够接受在当天的hh:mm（小时:分钟）式的时间指定。假如该时间已过去，那么就放在第二天执行。当然也能够使用midnight（深夜），noon（中午），teatime（饮茶时间，一般是下午4点）等比较模糊的 词语来指定时间。用户还能够采用12小时计时制，即在时间后面加上AM（上午）或PM（下午）来说明是上午还是下午。 也能够指定命令执行的具体日期，指定格式为month day（月 日）或mm/dd/yy（月/日/年）或dd.mm.yy（日.月.年）。指定的日期必须跟在指定时间的后面。 上面介绍的都是绝对计时法，其实还能够使用相对计时法，这对于安排不久就要执行的命令是很有好处的。指定格式为：now + count time-units ，now就是当前时间，time-units是时间单位，这里能够是minutes（分钟）、hours（小时）、days（天）、weeks（星期）。count是时间的数量，究竟是几天，还是几小时，等等。 更有一种计时方法就是直接使用today（今天）、tomorrow（明天）来指定完成命令的时间。

TIME：时间格式，这里可以定义出什么时候要进行 at 这项任务的时间，格式有：

HH:MM

ex> 04:00

在今日的 HH:MM 时刻进行，若该时刻已超过，则明天的 HH:MM 进行此任务。

HH:MM YYYY-MM-DD

ex> 04:00 2009-03-17

强制规定在某年某月的某一天的特殊时刻进行该项任务

HH:MM[am|pm] [Month] [Date]

ex> 04pm March 17

也是一样，强制在某年某月某日的某时刻进行该项任务

HH:MM[am|pm] + number [minutes|hours|days|weeks]

ex> now + 5 minutes

ex> 04pm + 3 days

就是说，在某个时间点再加几个时间后才进行该项任务。

实例1：三天后的下午 5 点执行 /bin/ls

[root@localhost ~]# at 5pm+3 days

at> /bin/ls

at> <EOT> # ctrl+d

实例2：明天17点钟，输出时间到指定文件内

[root@localhost ~]# at 17:20 tomorrow

at> date >/root/2020.log

at> <EOT> # ctrl+d

计划任务设定后，在没有执行之前我们可以查看系统没有执行工作任务：

命令：atq

删除已经设置的任务

命令：atrm 7

[**crontab命令**](https://www.cnblogs.com/peida/archive/2013/01/05/2846152.html)

周期性执行指定的命令。

-u user：用来设定某个用户的crontab服务，例如，“-u ixdba”表示设定ixdba用户的crontab服务，此参数一般有root用户来运行。

file：file是命令文件的名字,表示将file做为crontab的任务列表文件并载入crontab。如果在命令行中没有指定这个文件，crontab命令将接受标准输入（键盘）上键入的命令，并将它们载入crontab。

-e：编辑某个用户的crontab文件内容。如果不指定用户，则表示编辑当前用户的crontab文件。

-l：显示某个用户的crontab文件内容，如果不指定用户，则表示显示当前用户的crontab文件内容。

-r：从/var/spool/cron目录中删除某个用户的crontab文件，如果不指定用户，则默认删除当前用户的crontab文件。

-i：在删除用户的crontab文件时给确认提示。

crontab文件的含义：

用户所建立的crontab文件中，每一行都代表一项任务，每行的每个字段代表一项设置，它的格式共分为六个字段，前五段是时间设定段，第六段是要执行的命令段，格式如下：

minute   hour   day   month   week   command

其中：

minute： 表示分钟，可以是从0到59之间的任何整数。

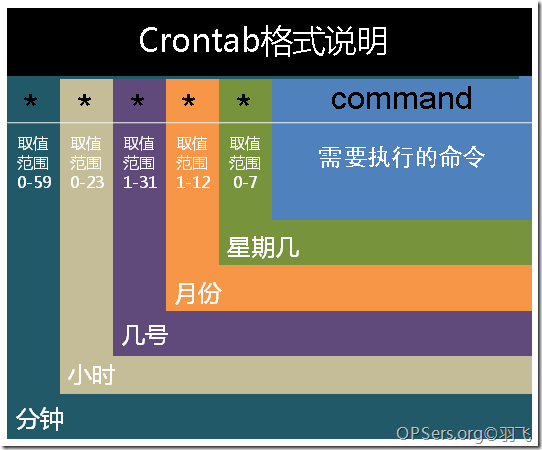
hour：表示小时，可以是从0到23之间的任何整数。

day：表示日期，可以是从1到31之间的任何整数。

month：表示月份，可以是从1到12之间的任何整数。

week：表示星期几，可以是从0到7之间的任何整数，这里的0或7代表星期日。

command：要执行的命令，可以是系统命令，也可以是自己编写的脚本文件。



在以上各个字段中，还可以使用以下特殊字符：

星号（\*）：代表所有可能的值，例如month字段如果是星号，则表示在满足其它字段的制约条件后每月都执行该命令操作。

逗号（,）：可以用逗号隔开的值指定一个列表范围，例如，“1,2,5,7,8,9”

中杠（-）：可以用整数之间的中杠表示一个整数范围，例如“2-6”表示“2,3,4,5,6”

正斜线（/）：可以用正斜线指定时间的间隔频率，例如“0-23/2”表示每两小时执行一次。同时正斜线可以和星号一起使用，例如\*/10，如果用在minute字段，表示每十分钟执行一次。

实例1：每1分钟执行一次command

命令：

\* \* \* \* \* command

实例2：每小时的第3和第15分钟执行

命令：

3,15 \* \* \* \* command

实例3：在上午8点到11点的第3和第15分钟执行

命令：

3,15 8-11 \* \* \* command

实例4：每隔两天的上午8点到11点的第3和第15分钟执行

命令：

3,15 8-11 \*/2 \* \* command

实例5：每个星期一的上午8点到11点的第3和第15分钟执行

命令：

3,15 8-11 \* \* 1 command

实例6：每晚的21:30重启smb

命令：

30 21 \* \* \* /etc/init.d/smb restart

实例7：每月1、10、22日的4 : 45重启smb

命令：

45 4 1,10,22 \* \* /etc/init.d/smb restart

**解析crontab参数：**

<https://crontab.guru/>

**netstat命令：**

netstat用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况。

-a或–all 显示所有连线中的Socket。

-A<网络类型>或–<网络类型> 列出该网络类型连线中的相关地址。

-c或–continuous 持续列出网络状态。

-C或–cache 显示路由器配置的快取信息。

-e或–extend 显示网络其他相关信息。

-F或–fib 显示FIB。

-g或–groups 显示多重广播功能群组组员名单。

-h或–help 在线帮助。

-i或–interfaces 显示网络界面信息表单。

-l或–listening 显示监控中的服务器的Socket。

-M或–masquerade 显示伪装的网络连线。

-n或–numeric 直接使用IP地址，而不通过域名服务器。

-N或–netlink或–symbolic 显示网络硬件外围设备的符号连接名称。

-o或–timers 显示计时器。

-p或–programs 显示正在使用Socket的程序识别码和程序名称。

-r或–route 显示Routing Table。

-s或–statistice 显示网络工作信息统计表。

-t或–tcp 显示TCP传输协议的连线状况。

-u或–udp 显示UDP传输协议的连线状况。

-v或–verbose 显示指令执行过程。

-V或–version 显示版本信息。

-w或–raw 显示RAW传输协议的连线状况。

-x或–unix 此参数的效果和指定”-A unix”参数相同。

–ip或–inet 此参数的效果和指定”-A inet”参数相同。

列出所有端口：netstat -a

显示当前UDP连接状况：netstat -nu

显示UDP端口号的使用情况：netstat -apu

显示网卡列表：netstat -i

显示网络统计信息：netstat -s

显示监听的套接口：netstat -l

显示所有已建立的有效连接：netstat -n

显示关于路由表的信息：netstat -r

列出所有 tcp 端口：netstat -at

找出程序运行的端口：netstat -ap | grep ssh

在 netstat 输出中显示 PID 和进程名称：netstat -pt

找出运行在指定端口的进程：netstat -anpt | grep ':16064'

**traceout命令：**

通过traceroute可以知道信息从本地到网络另一端的主机是走的什么路径。每次数据包由某一同样的出发点（source）到达某一同样的目的地(destination)走的路径可能会不一样，但基本上来说大部分时候所走的路由是相同的。traceroute通过发送小的数据包到目的设备直到其返回，来测量其需要多长时间。一条路径上的每个设备traceroute要测3次。输出结果中包括每次测试的时间(ms)和设备的名称（如有的话）及其IP地址。命令格式：

traceroute hostname

-d 使用Socket层级的排错功能。

-f 设置第一个检测数据包的存活数值TTL的大小。

-F 设置勿离断位。

-g 设置来源路由网关，最多可设置8个。

-i 使用指定的网络界面送出数据包。

-I 使用ICMP回应取代UDP资料信息。

-m 设置检测数据包的最大存活数值TTL的大小。

-n 直接使用IP地址而非主机名称。

-p 设置UDP传输协议的通信端口。

-r 忽略普通的Routing Table，直接将数据包送到远端主机上。

-s 设置本地主机送出数据包的IP地址。

-t 设置检测数据包的TOS数值。

-v 详细显示指令的执行过程。

-w 设置等待远端主机回报的时间。

-x 开启或关闭数据包的正确性检验。

**ss命令：**

ss是Socket Statistics的缩写，用来获取socket统计信息。可以显示和netstat类似的内容，优势在于能够显示更多更详细的有关TCP和连接状态的信息，而且比netstat更快速更高效。

当服务器的socket连接数量变得非常大时，无论是使用netstat命令还是直接cat /proc/net/tcp，执行速度都会很慢。服务器维持的连接达到上万个的时候，使用netstat等于浪费 生命，而用ss才是节省时间。

ss快的秘诀在于，它利用到了TCP协议栈中tcp\_diag。tcp\_diag是一个用于分析统计的模块，可以获得Linux 内核中第一手的信息，这就确保了ss的快捷高效。当然，如果你的系统中没有tcp\_diag，ss也可以正常运行，只是效率会变得稍慢，但仍然比 netstat要快。

**命令格式:**

ss [参数]

ss [参数] [过滤]

**命令参数：**

-h, --help 帮助信息

-V, --version 程序版本信息

-n, --numeric 不解析服务名称

-r, --resolve        解析主机名

-a, --all 显示所有套接字（sockets）

-l, --listening 显示监听状态的套接字（sockets）

-o, --options        显示计时器信息

-e, --extended       显示详细的套接字（sockets）信息

-m, --memory         显示套接字（socket）的内存使用情况

-p, --processes 显示使用套接字（socket）的进程

-i, --info 显示 TCP内部信息

-s, --summary 显示套接字（socket）使用概况

-4, --ipv4           仅显示IPv4的套接字（sockets）

-6, --ipv6           仅显示IPv6的套接字（sockets）

-0, --packet         显示 PACKET 套接字（socket）

-t, --tcp 仅显示 TCP套接字（sockets）

-u, --udp 仅显示 UCP套接字（sockets）

-d, --dccp 仅显示 DCCP套接字（sockets）

-w, --raw 仅显示 RAW套接字（sockets）

-x, --unix 仅显示 Unix套接字（sockets）

-f, --family=FAMILY  显示 FAMILY类型的套接字（sockets），FAMILY可选，支持  unix, inet, inet6, link, netlink

-A, --query=QUERY, --socket=QUERY

      QUERY := {all|inet|tcp|udp|raw|unix|packet|netlink}[,QUERY]

-D, --diag=FILE     将原始TCP套接字（sockets）信息转储到文件

 -F, --filter=FILE   从文件中都去过滤器信息

FILTER := [ state TCP-STATE ] [ EXPRESSION ]

显示TCP连接：ss -t -a

显示 Sockets 摘要：ss -s

列出所有打开的网络连接端口：ss -l

查看进程使用的socket：ss -pl

找出打开套接字/端口应用程序：ss -lp | grep 3306

显示所有UDP Sockets：ss -u -a

显示所有状态为established的SMTP连接：

ss -o state established '( dport = :smtp or sport = :smtp )'

显示所有状态为Established的HTTP连接：

ss -o state established '( dport = :http or sport = :http )'

列举出处于 FIN-WAIT-1状态的源端口为 80或者 443，目标网络为 193.233.7/24所有 tcp套接字：

ss -o state fin-wait-1 '( sport = :http or sport = :https )' dst 193.233.7/24

用TCP 状态过滤Sockets:

ss -4 state FILTER-NAME-HERE

ss -6 state FILTER-NAME-HERE

匹配远程地址和端口号：

ss dst ADDRESS\_PATTERN

ss dst 192.168.1.5

ss dst 192.168.119.113:http

ss dst 192.168.119.113:smtp

ss dst 192.168.119.113:443

匹配本地地址和端口号：

ss src ADDRESS\_PATTERN

ss src 192.168.119.103

ss src 192.168.119.103:http

ss src 192.168.119.103:80

ss src 192.168.119.103:smtp

ss src 192.168.119.103:25

将本地或者远程端口和一个数比较：

ss dport OP PORT

ss sport OP PORT

ss 和 netstat 效率对比：

time netstat -at

time ss

**curl命令：**

curl 是常用的命令行工具，用来请求 Web 服务器。它的名字就是客户端（client）的 URL 工具的意思。支持文件的上传和下载，支持HTTP、HTTPS、ftp等众多协议，还支持POST、cookies、认证、从指定偏移处下载部分文件、用户代理字符串、限速、文件大小、进度条等特征。

不带有任何参数时，curl 就是发出 GET 请求。

curl <https://www.baidu.com>

-d参数用于发送 POST 请求的数据体。

使用-d参数以后，HTTP 请求会自动加上标头Content-Type : application/x-www-form-urlencoded。并且会自动将请求转为 POST 方法，因此可以省略-X POST。

-d参数可以读取本地文本文件的数据，向服务器发送。

curl -d '@data.txt' <https://google.com/login>

-H参数添加 HTTP 请求的标头。

curl -H 'Accept-Language: en-US' https://google.com 添加 HTTP 标头Accept-Language: en-US。

curl -H 'Accept-Language: en-US' -H 'Secret-Message: xyzzy' https://google.com 添加两个 HTTP 标头。

curl -d '{"login": "emma", "pass": "123"}' -H 'Content-Type: application/json' https://google.com/login 添加 HTTP 请求的标头是Content-Type: application/json，然后用-d参数发送 JSON 数据。

-x参数指定 HTTP 请求的代理。

curl -x socks5://james:cats@myproxy.com:8080 https://www.example.com

上面命令指定 HTTP 请求通过myproxy.com:8080的 socks5 代理发出。

如果没有指定代理协议，默认为 HTTP。

curl -x james:cats@myproxy.com:8080 https://www.example.com

请求的代理使用 HTTP 协议。

-X参数指定 HTTP 请求的方法。curl -X POST https://www.example.com

对https://www.example.com发出 POST 请求。

-F参数用来向服务器上传二进制文件。

curl -F 'file=@photo.png' https://google.com/profile

给 HTTP 请求加上标头Content-Type: multipart/form-data，然后将文件photo.png作为file字段上传。

-F参数可以指定 MIME 类型。

curl -F 'file=@photo.png;type=image/png' https://google.com/profile

指定 MIME 类型为image/png，否则 curl 会把 MIME 类型设为application/octet-stream。

-F参数也可以指定文件名。

curl -F 'file=@photo.png;filename=me.png' https://google.com/profile

上面命令中，原始文件名为photo.png，但是服务器接收到的文件名为me.png

更多参考：

<https://catonmat.net/cookbooks/curl>

**find命令：**

Linux下find命令在目录结构中搜索文件，并执行指定的操作。在运行一个非常消耗资源的find命令时，很多人都倾向于把它放在后台执行，因为遍历一个大的文件系统可能会花费很长的时间。

find pathname -options [-print -exec -ok ...]

pathname: find命令所查找的目录路径。例如用.来表示当前目录，用/来表示系统根目录。

-exec：find命令对匹配的文件执行该参数所给出的shell命令。相应命令的形式为'command' {} \;，**注意{}和\；之间有空格**。

-ok：和-exec的作用相同，只不过以一种更为安全的模式来执行该参数所给出的shell命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

-name 按照文件名查找文件。

-perm 按照文件权限来查找文件。

-prune 使用这一选项可以使find命令不在当前指定的目录中查找，如果同时使用-depth选项，那么-prune将被find命令忽略。

-user 按照文件属主来查找文件。

-group 按照文件所属的组来查找文件。

-mtime -n +n 按照文件的更改时间来查找文件， -n表示文件更改时间距现在n天以内，+ n表示文件更改时间距现在n天以前。find命令还有-atime和-ctime 选项，但它们都和-mtime选项。

-nogroup 查找无有效所属组的文件，即该文件所属的组在/etc/groups中不存在。

-nouser 查找无有效属主的文件，即该文件的属主在/etc/passwd中不存在。

-newer file1 ! file2 查找更改时间比文件file1新但比文件file2旧的文件。

-type 查找某一类型的文件，诸如：

b - 块设备文件。

d - 目录。

c - 字符设备文件。

p - 管道文件。

l - 符号链接文件。

f - 普通文件。s

-size n：[c] 查找文件长度为n块的文件，带有c时表示文件长度以字节计。-depth：在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。

-fstype：查找位于某一类型文件系统中的文件，这些文件系统类型通常可以在配置文件/etc/fstab中找到，该配置文件中包含了本系统中有关文件系统的信息。

-mount：在查找文件时不跨越文件系统mount点。

-follow：如果find命令遇到符号链接文件，就跟踪至链接所指向的文件。

-cpio：对匹配的文件使用cpio命令，将这些文件备份到磁带设备中。

另外,下面三个的区别:

-amin n 查找系统中最后N分钟访问的文件

-atime n 查找系统中最后n\*24小时访问的文件

-cmin n 查找系统中最后N分钟被改变文件状态的文件

-ctime n 查找系统中最后n\*24小时被改变文件状态的文件

-mmin n 查找系统中最后N分钟被改变文件数据的文件

-mtime n 查找系统中最后n\*24小时被改变文件数据的文件

举例：

find –atime -2超找48小时内修改过的文件

find /opt/soft/test/ -perm 777查找/opt/soft/test/目录下 权限为 777的文件

find . -type d 查找目录文件

find . -size +1000c 查找大于1k字符的文件

-exec  参数后面跟的是command命令，它的终止是以;为结束标志的，所以这句命令后面的分号是不可缺少的，考虑到各个系统中分号会有不同的意义，所以前面加反斜杠。{}   花括号代表前面find查找出来的文件名。

find . -type f -exec ls -l {} \;

排除指定文件夹：

find ./ -type d -path "./replay" -prune -o -type d -print

输出当前目录下除replay之外的所有子目录

相当于

find ./ -type d -path "./replay" –a -prune -o -type d -print

-a和-o相当于&&和||，执行短路原则：

if (是replay目录) {

- prune // 跳过

} else {

-type d –print // 查找文件夹并输出到std out

}

find ./ -type d -path "./replay" -prune -o –print

if (是replay目录) {

- prune // 跳过

} else {

-print // 查找所有文件和文件夹并打到std out

}

跳过多个目录：

find ./ -type d \( -path "./replay" -o -path "./base" \) -a -prune -o -type d –print

查找更改时间比文件m2u\_leaf\_server.INFO新的文件：

find ./ -newer m2u\_leaf\_server.INFO

查找更改时间比文件m2u\_leaf\_server.INFO新，但比文件server\_run.out旧的文件：

find ./ -newer m2u\_leaf\_server.INFO ! -newer server\_run.out

查找后缀为.cc和.h的文件：

find默认采用emacs正则，会比较罗嗦，如果采用posix-extended正则会比较简单：

find ./ -regextype posix-extended -regex '.+\.(h|cc)'

find ./ -name "\*.h" -o -name "\*.cc"

**top命令：**

实时显示系统中各个进程的资源占用状况，类似于Windows的任务管理器。

-b 批处理

-c 显示完整的治命令

-I 忽略失效过程

-s 保密模式

-S 累积模式

-i<时间> 设置间隔时间

-u<用户名> 指定用户名

-p<进程号> 指定进程

-n<次数> 循环显示的次数

-H 显示线程信息

多核 CPU监控

在top基本视图中，按键盘数字1，可监控每个逻辑CPU的状况，再按数字1就返回基本视图。

进程字段排序

默认进入top时，各进程是按照CPU的占用量来排序的, 按x可打开/关闭排序字段的高亮显示。通过”shift + >”或”shift + <”可以向右或左改变排序列。

**top交互命令**

在top 命令执行过程中可以使用的一些交互命令。这些命令都是单字母的，如果在命令行中使用了s 选项， 其中一些命令可能会被屏蔽。

h 显示帮助画面，给出一些简短的命令总结说明

k 终止一个进程。

i 忽略闲置和僵死进程。这是一个开关式命令。

q 退出程序

r 重新安排一个进程的优先级别

S 切换到累计模式

s 改变两次刷新之间的延迟时间（单位为s），如果有小数，就换算成m s。输入0值则系统将不断刷新，默认值是5 s

f或者F 从当前显示中添加或者删除项目

o或者O 改变显示项目的顺序

l 切换显示平均负载和启动时间信息

m 切换显示内存信息

t 切换显示进程和CPU状态信息

c 切换显示命令名称和完整命令行

M 根据驻留内存大小进行排序

P 根据CPU使用百分比大小进行排序

T 根据时间/累计时间进行排序

W 将当前设置写入~/.toprc文件中

查看某个进程的线程信息：

top -Hp {pid} #可用于定位死循环，看进程里哪个线程卡了

pstack {pid}

**linux免密登录：**

两台机器：server1, server2

需求：在server1上免密登录server2

步骤：

1. server1上执行ssh-keygen -t rsa，生成文件~/.ssh/id\_rsa和~/.ssh/id\_rsa.pub
2. id\_rsa.pub拷贝到server2
3. server2上执行cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

server1上再用ssh试下，如果不生效，检查/etc/ssh/ssh\_config里选项IdentityFile ~/.ssh/id\_rsa是否打开。

**set命令：**

Linux set命令用于设置shell。

set指令能设置所使用shell的执行方式，可依照不同的需求来做设置。

如果命令行下不带任何参数，直接运行set，会显示所有的环境变量和 Shell 函数。

-a 　标示已修改的变量，以供输出至环境变量。

-b 　使被中止的后台程序立刻回报执行状态。

-C 　转向所产生的文件无法覆盖已存在的文件。

-d 　Shell预设会用杂凑表记忆使用过的指令，以加速指令的执行。使用-d参数可取消。

-e 　若指令传回值不等于0，则立即退出shell。

-f　 　取消使用通配符。

-h 　自动记录函数的所在位置。

-H Shell 　可利用"!"加<指令编号>的方式来执行history中记录的指令。

-k 　指令所给的参数都会被视为此指令的环境变量。

-l 　记录for循环的变量名称。

-m 　使用监视模式。

-n 　只读取指令，而不实际执行。

-p 　启动优先顺序模式。

-P 　启动-P参数后，执行指令时，会以实际的文件或目录来取代符号连接。

-t 　执行完随后的指令，即退出shell。

-u 　当执行时使用到未定义过的变量，则显示错误信息。

-v 　显示shell所读取的输入值。

-x 　执行指令后，会先显示该指令及所下的参数。

+<参数> 　取消某个set曾启动的参数。

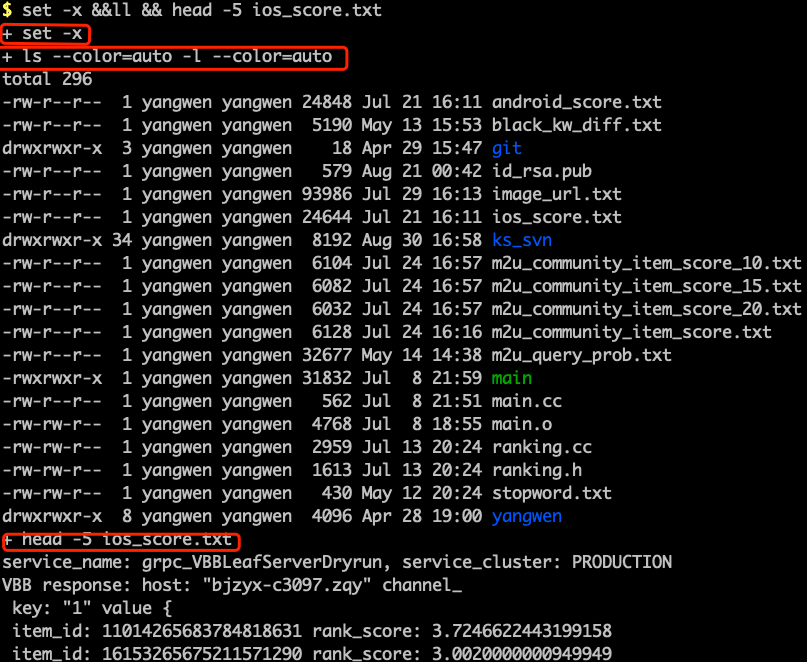
有4个参数最常用：

**set -u**

执行脚本的时候，如果遇到不存在的变量，Bash 默认忽略它。set -u就用来改变这种行为。脚本在头部加上它，遇到不存在的变量就会报错，并停止执行。-u还有另一种写法-o nounset，两者是等价的。

**set -x**

默认情况下，脚本执行后，屏幕只显示运行结果，没有其他内容。如果多个命令连续执行，它们的运行结果就会连续输出。有时会分不清，某一段内容是什么命令产生的。set -x用来在运行结果之前，先输出执行的那一行命令。-x还有另一种写法-o xtrace。



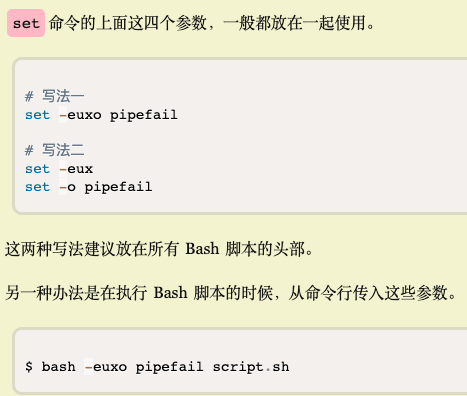
**set -e**

脚本只要发生错误，就终止执行。set -e根据返回值来判断，一个命令是否运行失败。但是，某些命令的非零返回值可能不表示失败，或者开发者希望在命令失败的情况下，脚本继续执行下去。这时可以暂时关闭: set +e，该命令执行结束后，再重新打开set -e。-e还有另一种写法-o errexit。

**set -o pipefail**

set -e有一个例外情况，就是不适用于管道命令。

所谓管道命令，就是多个子命令通过管道运算符（|）组合成为一个大的命令。Bash 会把最后一个子命令的返回值，作为整个命令的返回值。也就是说，只要最后一个子命令不失败，管道命令总是会执行成功，因此它后面命令依然会执行，set -e就失效了。set -o pipefail用来解决这种情况，只要一个子命令失败，整个管道命令就失败，脚本就会终止执行。



**date命令：**

Linux date命令可以用来显示或设定系统的日期与时间。

-d, --date=datestr显示datestr指定的时间，**注意：非系统时间**

-s, --set=datestr设置系统时间为datestr指定的时间

-r, --reference=FILE 显示系统最后一次修改时间

输出格式(以+开头)：

% : 印出 %

%n : 下一行

%t : 跳格

%H : 小时(00..23)

%I : 小时(01..12)

%k : 小时(0..23)

%l : 小时(1..12)

%M : 分钟(00..59)

%p : 显示本地 AM 或 PM

%r : 直接显示时间 (12 小时制，格式为 hh:mm:ss [AP]M)

%s : 从 1970 年 1 月 1 日 00:00:00 UTC 到目前为止的秒数

%S : 秒(00..61)

%T : 直接显示时间 (24 小时制)

%X : 相当于 %H:%M:%S

%Z : 显示时区

%a : 星期几 (Sun..Sat)

%A : 星期几 (Sunday..Saturday)

%b : 月份 (Jan..Dec)

%B : 月份 (January..December)

%c : 直接显示日期与时间

%d : 日 (01..31)

%D : 直接显示日期 (mm/dd/yy)

%h : 同 %b

%j : 一年中的第几天 (001..366)

%m : 月份 (01..12)

%U : 一年中的第几周 (00..53) (以 Sunday 为一周的第一天的情形)

%w : 一周中的第几天 (0..6)

%W : 一年中的第几周 (00..53) (以 Monday 为一周的第一天的情形)

%x : 直接显示日期 (mm/dd/yy)

%y : 年份的最后两位数字 (00.99)

%Y : 完整年份 (0000..9999)

例子：

date +%Y%m%d 以yyyymmdd格式输出系统当前时间

date --date="1 day ago" +%Y%m%d 一天前的日期

date -d "-2 day 20200911" +%Y%m%d 输出20200909

date +%s 系统时间按unix时间输出

date -d @1287331200 +%Y-%m-%d unix时间转为指定格式

计算时间差

time1=$(($(date +%s -d '2010-01-01 11:11:13') -$(date +%s -d '2010-01-01 11:11:11')))

echo $time1

date -d `date +%y%m01` 显示本月第一天

date -d `date +%y%m01`"-1 day" 显示上个月最后一天

date -d `date -d "-3 month" +%y%m01`"-1 day" 显示4个月前最后一天

date -d `date -d "+3 month" +%y%m01`"-1 day" 显示两个月后最后一天

**tr命令：**

tr 命令用于转换或删除字符。

tr [OPTION]... SET1 [SET2]

-c, --complement：反选设定字符。也就是符合 SET1 的部份不做处理，不符合的剩余部份才进行转换

-d, --delete：删除指令字符

-s, --squeeze-repeats：缩减连续重复的字符成指定的单个字符

-t, --truncate-set1：削减 SET1 指定范围，使之与 SET2 设定长度相等

SET1：指定要转换或删除的原字符集。当执行转换操作时，必须使用SET2指定转换的目标字符集。但执行删除操作时，不需要SET2；

SET2：指定要转换成的目标字符集。

例子：

字符由小写转换为大写：

echo "aef" | tr -t a-z A-Z

AEF

SET1范围小于SET2:

echo "aef" | tr 'a-e' 'A-F'

AEf

SET1范围大于SET2:

echo "adcbef" | tr 'a-f' 'A-D'

ADCBDD 可以看出，SET1超过SET2长度的部分， SET2会用最后一个字符自动扩展到跟SET1长度一致

echo "adcbef" | tr -t 'a-f' 'A-D'

ADCBef 相当于tr 'a-d' 'A-D'

使用tr删除字符：

echo "hello 123 world 456" | tr -d ' 0-9'

helloworld 数字和空格都被删除了

使用补集，删除非数字和非换行的字符：

echo "hello 123 world 456" | tr -d -c '0-9\n'

压缩输入中重复的字符：

echo "thissss is a text linnnnnnne." | tr -s ' sn'

this is a text line.

巧妙使用tr做数字相加操作：

echo 1 2 3 4 5 6 7 8 9 | [xargs](http://man.linuxde.net/xargs) -n1 | echo $[ $(tr '\n' '+') 0 ]

45

删除Windows文件“造成”的'^M'字符：

cat [file](http://man.linuxde.net/file) | tr -s "\r" "\n" > new\_file

或

cat file | tr -d "\r" > new\_file

大小写转换：

echo "adcbef78" | tr [:lower:] [:upper:]

ADCBEF78

tr的字符集合：

\NNN 八进制值的字符 NNN (1 to 3 为八进制值的字符)

\\ 反斜杠

\a Ctrl-G 铃声

\b Ctrl-H 退格符

\f Ctrl-L 走行换页

\n Ctrl-J 新行

\r Ctrl-M 回车

\t Ctrl-I tab键

\v Ctrl-X 水平制表符

CHAR1-CHAR2 ：字符范围从 CHAR1 到 CHAR2 的指定，范围的指定以 ASCII 码的次序为基础，只能由小到大，不能由大到小。

[CHAR\*] ：这是 SET2 专用的设定，功能是重复指定的字符到与 SET1 相同长度为止

[CHAR\*REPEAT] ：这也是 SET2 专用的设定，功能是重复指定的字符到设定的 REPEAT 次数为止(REPEAT 的数字采 8 进位制计算，以 0 为开始)

[:alnum:] ：所有字母字符与数字

[:alpha:] ：所有字母字符

[:blank:] ：所有水平空格

[:cntrl:] ：所有控制字符

[:digit:] ：所有数字

[:graph:] ：所有可打印的字符(不包含空格符)

[:lower:] ：所有小写字母

[:print:] ：所有可打印的字符(包含空格符)

[:punct:] ：所有标点字符

[:space:] ：所有水平与垂直空格符

[:upper:] ：所有大写字母

[:xdigit:] ：所有 16 进位制的数字

[=CHAR=] ：所有符合指定的字符(等号里的 CHAR，代表你可自订的字符)

**rename命令：**

用字符串替换的方式批量改变文件名。

参数：

原字符串：将文件名需要替换的字符串；

目标字符串：将文件名中含有的原字符替换成目标字符串；

文件：指定要改变文件名的文件列表。

rename main1.c main.c main1.c // 将main1.c重命名为main.c

**rename支持通配符:**

? 可替代单个字符

\* 可替代多个字符

[charset] 可替代charset集中的任意单个字符

举例：

文件夹中有这些文件foo1, ..., foo9, foo10, ..., foo278

如果使用rename foo foo0 foo?，会把foo1到foo9的文件重命名为foo01到foo09，重命名的文件只是有4个字符长度名称的文件，文件名中的foo被替换为foo0。

如果使用rename foo foo0 foo??，foo01到foo99的所有文件都被重命名为foo001到foo099，只重命名5个字符长度名称的文件，文件名中的foo被替换为foo0。

如果使用rename foo foo0 foo\*，foo001到foo278的所有文件都被重命名为foo0001到foo0278，所有以foo开头的文件都被重命名。

如果使用rename foo0 foo foo0[2]\*，从foo0200到foo0278的所有文件都被重命名为foo200到foo278，文件名中的foo0被替换为foo。

**rename支持正则表达式：**

rename "s/AA/aa/" \* //把文件名中的AA替换成aa

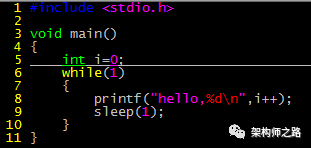
rename "s//.html//.[php](http://man.linuxde.net/php)/" \* //把.html 后缀的改成 .php后缀

rename "s/$//.txt/" \* //把所有的文件名都机上txt后缀

rename "s//.txt//" \* //把所有以.txt结尾的文件名的.txt删掉

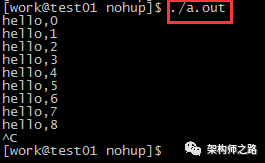
**nohup和&的功效**

测试代码如下：



是一个输出hello与循环轮数的死循环程序，每输出一行就休眠1秒。

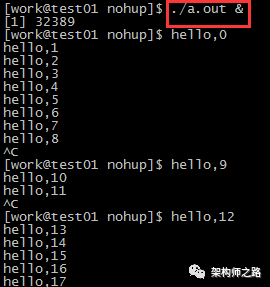
**使用 ./a.out 前台运行程序，会是什么效果呢？**



程序每隔一秒会在终端输出一个字符串。

此时如果键入Ctrl+C ，程序会收到一个SIGINT信号，如果不做特殊处理，程序的默认行为是终止（如上图）。

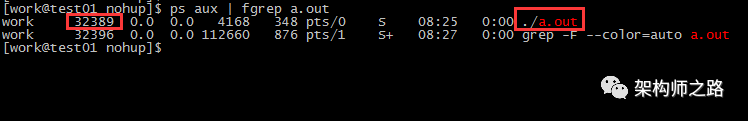
**使用 ./a.out& 后台运行程序，会是什么效果呢？**



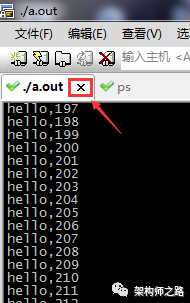
如上图：

首先会在终端显示进程号是32389

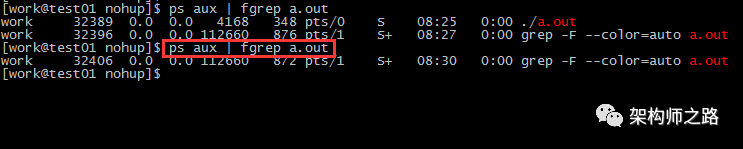
键入Ctrl + C，发出SIGINT信号，程序会继续运行



ps确认一下，确认进程依然在运行，进程号是32389。

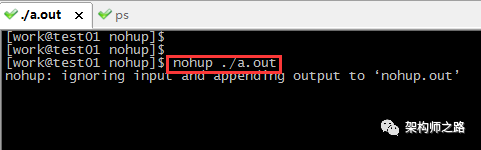


此时如果关掉session，程序会收到一个SIGHUP信号，此时会怎么样呢？



ps再次确认，可以看到关闭session之后，进程号是32389的a.out进程也关闭了。

**使用nohup ./a.out 又会是什么效果呢？**

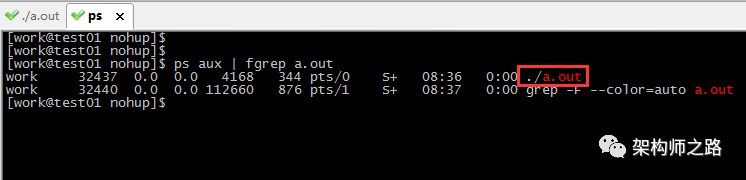


使用nohup 运行程序a.out，会发现：

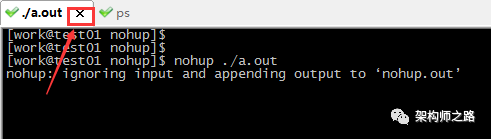
前台没有出现进程号

有一个“忽略输入，输出至nohup.out”的提示

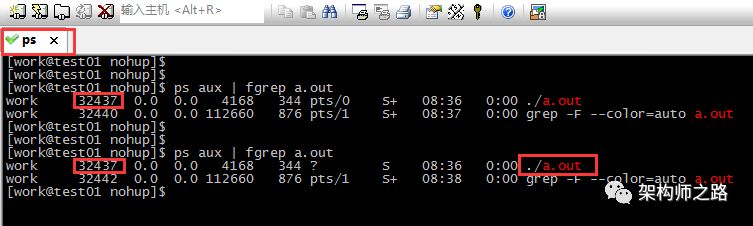
hello的输出也没有出现在前台



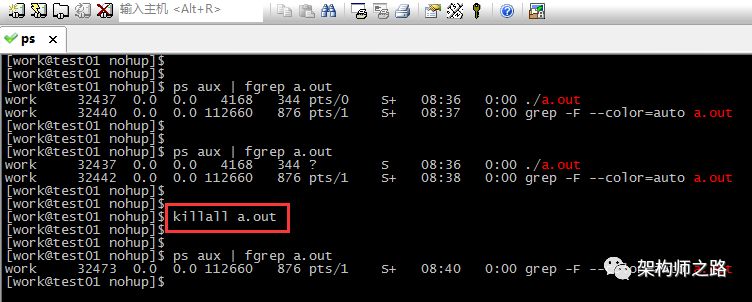
手动ps看进程号，这次a.out的进程号是32437。



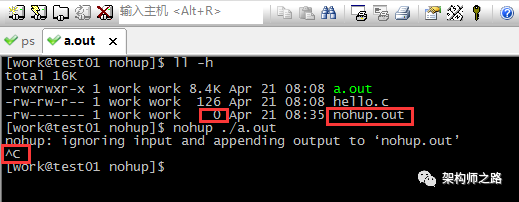
此时如果关掉session，程序会收到一个SIGHUP信号，程序会不会关闭呢？



关掉session后，再次ps看一下，ID为32437的a.out进程还在。



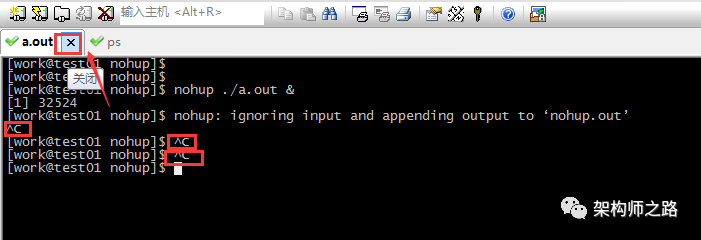
这些只能通过kill把程序干掉了，killall之后，ps查看进程已经关闭。



killall之后，查看发现多了一个nohup.out文件，不过这个文件的大小是0，有点奇怪，启动程序的时候，明明提示了“appending output to nohup.out”呀，先把问题遗留在这，测试一下Ctrl +C。

仍如上图，使用nohup启动a.out，如果键入Ctrl+C ，程序收到SIGINT信号后，直接关闭了。

**最后测试一下nohup和&同时使用，即用nohup./a.out &运行程序，又会是什么效果呢？**



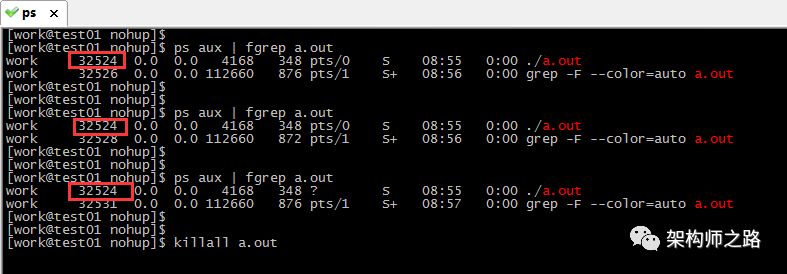
使用nohup ./a.out &运行程序后，可以看到：

会在终端显示进程号是32524

也会有一个“忽略输入，输出至nohup.out”的提示

键入Ctrl + C，发送SIGINT信号，似乎没反应。

关闭session，发送SIGHUP信号，再来看看。



**ID为32524的进程依然存在，后续也只能用kill来关闭它。**

**结论**

**使用&后台运行程序：**

**结果会输出到终端**

**使用Ctrl + C发送SIGINT信号，程序免疫**

**关闭session发送SIGHUP信号，程序关闭**

**使用nohup运行程序：**

**结果默认会输出到nohup.out**

**使用Ctrl + C发送SIGINT信号，程序关闭**

**关闭session发送SIGHUP信号，程序免疫**

**平日线上经常使用nohup和&配合来启动程序：**

**同时免疫SIGINT和SIGHUP信号**

**ps命令：**

ps命令支持三种使用的语法格式:

UNIX 风格，选项可以组合在一起，并且选项前必须有“-”连字符

BSD 风格，选项可以组合在一起，但是选项前不能有“-”连字符

GNU 风格的长选项，选项前有两个“-”连字符

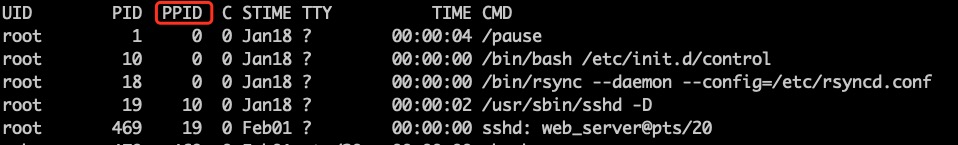
显示所有进程：

ps -A

ps -e

显示所有进程(带执行命令，PPID是父进程id)：

ps -ef



根据 CPU 使用降序排序

ps -aux --sort -pcpu

根据内存使用升序排序

ps -aux --sort -pmem

根据CPU降序&内存升序排序

ps -aux --sort -pcpu,+pmem

根据用户过滤

ps -u root

用进程名过滤

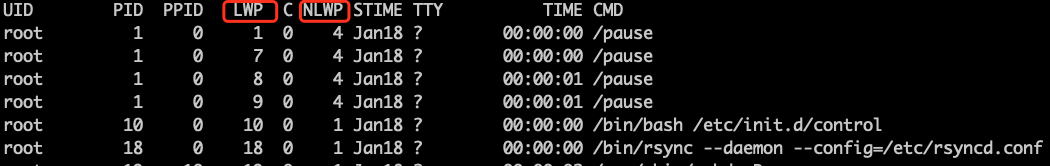
ps -C bash

树状显示

ps -axjf

显示线程(LWP或NLWP)

ps -eLf



**shell中的路径：**

shell文件中调用pwd，输出的执行路径，$0为文件所在位置相对于执行路径的相对路径。

~/ks\_svn/reco/test.sh

#!/bin/bash

echo "pwd: "`pwd`

echo '$0: ' $0

echo "dirname: "`dirname $0`

如：

在~/ks\_svn/reco下执行sh test.sh，输出：

pwd: /home/yangwen/ks\_svn/reco

$0: tesh.sh

dirname: .

在~/ks\_svn下执行sh reco/tesh.sh，输出：

pwd: /home/yangwen/ks\_svn

$0: reco/tesh.sh

dirname: reco

在~/ks\_svn/reco/fx/ks\_flash下执行sh ../../tesh.sh，输出：

pwd: /home/yangwen/ks\_svn/reco/fx/ks\_flash

$0: ../../tesh.sh

dirname: ../..

想要在shell脚本中获得脚本所在的绝对路径：

$(cd "$(dirname "$0")";pwd)

或

$(dirname $(readlink -f "$0"))

**readlink命令**

输出软链的源文件，或将文件路径转为全路径。

ln -s tesh.sh tesh\_1.sh

ln -s tesh\_1.sh tesh\_2.sh

readlink reco/tesh\_2.sh # 输出tesh\_1.sh

readlink -f reco/tesh\_2.sh # 输出/home/yangwen/ks\_svn/reco/tesh.sh

cd reco/fx/ && readlink -f ../tesh.sh # 输出/home/yangwen/ks\_svn/reco/tesh.sh

**file命令**

file是通过查看文件的头部内容，来获取文件的类型使用file命令可以知道某个文件究竟是二进制（ELF格式）的可执行文件, 还是Shell Script文件，或者是其它的什么格式。

file能识别的文件类型：目录、Shell脚本、英文文本、二进制可执行文件、C语言源文件、文本文件、DOS的可执行文件。



**stat命令**

用于显示文件的状态信息。

-L：支持符号连接；

-f：显示文件系统状态而非文件状态；

-t：以简洁方式输出信息；

--[help](http://man.linuxde.net/help)：显示指令的帮助信息；

--version：显示指令的版本信息。

**Linux日志处理**

看到[LINUX系统](http://www.qqread.com/z/linux/learning/index.html" \t "_blank)信息日志的途径基本有以下2种：  
(1) dmesg

(2) /var/log/下的文件

dmesg是从kernel 的ring buffer（环缓冲区）中读取信息的。

什么是ring buffer呢？

在LINUX中，所有的系统信息（包内核信息）都会传送到ring buffer中。而内核产生的信息由printk（）打印出来。系统启动时所看到的信息都是由该函数打印到屏幕中。 printk（）打出的信息往往以 ＜0＞＜2＞……这的数字表明消息的重要级别。高于一定的优先级别会打印到屏幕上， 否则只会保留在系统的缓冲区中（ring buffer）。

至于dmesg具体是如何从ring buffer中读取的，大家可以看dmesg.c源代码。很短，比较容易读懂。

/var/log/下为什么有这么多文件呢？

一句话解释： 是syslogd这个守护[进程](http://www.qqread.com/z/windows/process/index.html" \t "_blank)根据/etc/syslog.conf，将不同的服务产生的Log记录到不同的文件中。

LINUX系统启动后，由/etc/init.d/sysklogd先后启动klogd，syslogd两个守护进程。其中klogd会通过syslog（）系统调用或者读取proc文件系统来从系统缓冲区（ring buffer）中得到由内核printk（）发出的信息。而syslogd是通过klogd来读取系统内核信息。

总结：

（1）所有系统信息是输出到ring buffer中去的。dmesg所显示的内容也是从ring buffer中读取的。

（2）LINUX系统中/etc/init.d/sysklogd会启动2个守护进程：Klogd&&Syslogd（3）klogd是负责读取内核信息的，有2种方式：

syslog（）系统调用

直接的对/proc/kmsg进行读取（/proc/kmsg是专门输出内核信息的地方）

（4）Klogd的输出结果会传送给syslogd进行处理，syslogd会根据/etc/syslog.conf的[配置](http://www.qqread.com/z/sys/safe-seting/index.html" \t "_blank)把log信息输出到/var/log/下的不同文件中。

原博：

https://blog.51cto.com/bowen/107113

**dmesg命令**

dmesg命令是用来在[Unix-like](https://baike.baidu.com/item/%E7%B1%BBUNIX%E7%B3%BB%E7%BB%9F/4336219?fr=aladdin" \t "_blank)系统中显示内核的相关信息的。dmesg全称是display message (or display driver)，从kernal ring buffer中获取数据的。dmesg命令能很好地帮我们鉴别硬件相关的error和warning。除此之外，dmesg命令还能打印出守护进程相关的信息，帮助debug。

-c 显示信息后，清除ring buffer中的内容。

-s<缓冲区大小> 预设置为8196，刚好等于ring buffer的大小。

-n 设置记录信息的层级。

显示和内存、硬盘、USB、TTY、网卡相关的信息

dmesg | grep -i memory

dmesg | grep -i dma

dmesg | grep -i usb

dmesg | grep -i tty

dmesg | grep -i eth

只输出特定类型的信息

dmesg --facility=daemon //输出守护进程的信息

显示时间戳

dmesg命令默认的时间显示是从开机到log记录的时间，对于人类来说真的不方便，需要可读时间：

dmesg -T

更进一步，把级别也显示出来：

dmesg -Tx

显示原始数据

dmesg默认输出的信息是处理过的，想要显示原始数据（raw data）：

dmesg -r

**wc命令**

-L 最长一行的长度

-l 文件行数

-w 单词数量

-c 字节数

-m 字符数

**shell字符串操作**

字符串长度：

${#string\_name}

str="fat cat"

echo ${#str}

字符串拼接：

name="Shell"

url="http://www.baidu.com"

str1=$name$url #中间不能有空格

str2="$name $url" #如果被双引号包围，那么中间可以有空格

str3=$name": "$url #中间可以出现别的字符串

str4="$name: $url" #这样写也可以

str5="${name}Script: ${url}index.html" #这个时候需要给变量名加上大括号

字符串截取：

|  |  |
| --- | --- |
| 格式 | 说明 |
| ${string: start :length} | 从 string 字符串的左边第 start 个字符开始，向右截取 length 个字符。 |
| ${string: start} | 从 string 字符串的左边第 start 个字符开始截取，直到最后。 |
| ${string: 0-start :length} | 从 string 字符串的右边第 start 个字符开始，向右截取 length 个字符。 |
| ${string: 0-start} | 从 string 字符串的右边第start个字符开始截取，直到最后。 |
| ${string#\*chars} | 从 string 字符串第一次出现 \*chars的位置开始，截取\*chars右边的所有字符。 |
| ${string##\*chars} | 从 string 字符串最后一次出现 \*chars的位置开始，截取\*chars 右边的所有字符。 |
| ${string%\*chars} | 从右找，string字符串第一次出现 \*chars的位置开始，截取\*chars左边的所有字符。 |
| ${string%%\*chars} | 从右找，string字符串最后一次出现\*char的位置开始，截取\*chars左边的所有字符。 |

url="c.biancheng.net"

echo ${url: 2: 9} 输出biancheng

echo ${url: 2} 输出biancheng.net

echo ${url: 0-13: 9} 输出biancheng

echo ${url: 0-13} 输出biancheng.net

**chars是指定的字符（或者子字符串），\*是通配符的一种，表示任意长度的字符串。**

url="http://c.biancheng.net/index.html"

echo ${url#\*:} 输出c.biancheng.net/index.html

echo ${url#http://} 输出c.biancheng.net/index.html

echo ${url##\*/} 输出 index.html

echo ${url%/\*} 输出<http://c.biancheng.net>

echo ${url%%/\*}输出http:

字符串替换

${string/substring/replacement} 使用$replacement, 来代替第一个匹配的$substring

${string//substring/replacement} 使用$replacement, 代替所有匹配的$substring

**pstack命令**

pstack {pid}

pstack是一个shell脚本，用于打印正在运行的进程的栈跟踪信息，它实际上是gstack的一个链接，而gstack本身是基于gdb封装的shell脚本。此命令可显示每个进程的栈跟踪。pstack 命令必须由相应进程的属主或 root 运行。可以使用 pstack 来确定进程挂起的位置。

pstack是gdb的一部分，如果系统没有pstack命令，使用yum搜索安装gdb即可。

这个命令在排查进程问题时非常有用，比如我们发现一个服务一直处于work状态（如假死状态，好似死循环），使用这个命令就能轻松定位问题所在；可以在一段时间内，多执行几次pstack，若发现代码栈总是停在同一个位置，那个位置就需要重点关注，很可能就是出问题的地方。

**strace命令**

strace是一个可用于诊断、调试和教学的Linux用户空间跟踪器。可以用它来监控用户空间进程和内核的交互，比如系统调用、信号传递、进程状态变更等。

strace底层使用内核的ptrace特性来实现其功能。

strace有两种运行模式。

一种是通过它启动要跟踪的进程：在原本的命令前加上strace即可。比如我们要跟踪 "ls -lh /var/log/messages" 这个命令的执行，可以这样：

strace ls -lh /var/log/messages

另外一种运行模式，是跟踪已经在运行的进程，给strace传递个-p pid选项:

strace -p {pid}

完成跟踪时，按ctrl + C 结束strace即可。

**常用选项：**

strace -tt -T -v -f -e trace=file -o strace.log -s 1024 ls -lh log/dryrun\_nearby\_realtime\_reco\_leaf.INFO

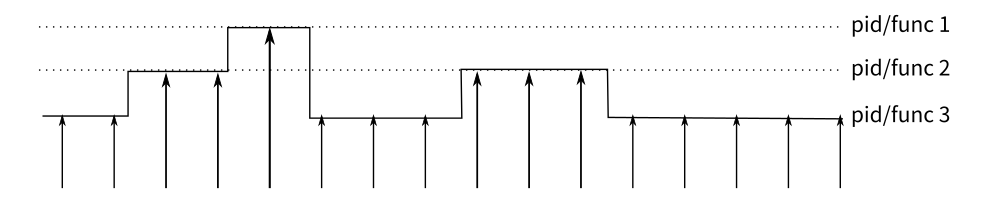
-tt 在每行输出的前面，显示毫秒级别的时间  
-T 显示每次系统调用所花费的时间  
-v 对于某些相关调用，把完整的环境变量，文件stat结构等打出来。  
-f 跟踪目标进程，以及目标进程创建的所有子进程  
-e 控制要跟踪的事件和跟踪行为,比如指定要跟踪的系统调用名称  
-o 把strace的输出单独写到指定的文件  
-s 当系统调用的某个参数是字符串时，最多输出指定长度的内容，默认是32个字节  
-p 指定要跟踪的进程pid, 要同时跟踪多个pid, 重复多次-p选项即可。

**perf命令**

perf是Linux下的一款性能分析工具，能够进行函数级与指令级的热点查找。

perf的原理：

每隔一个固定的时间，就在CPU上（每个核上都有）产生一个中断，在中断上看看，当前是哪个pid，哪个函数，然后给对应的pid和函数加一个统计值，这样，我们就知道CPU有百分几的时间在某个pid，或者某个函数上了。这个原理图示如下：

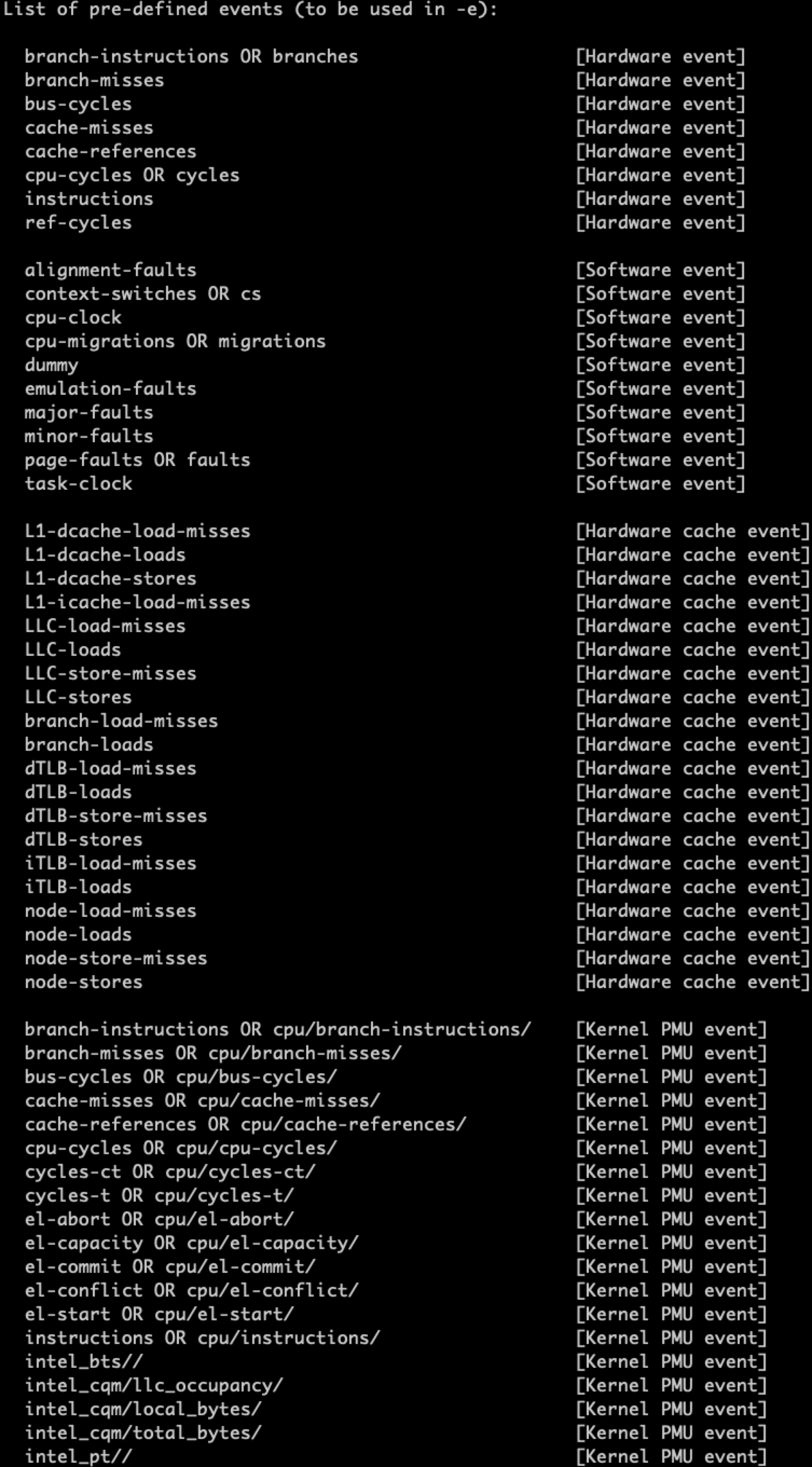


这是一种采样的模式，我们预期，运行时间越多的函数，被时钟中断击中的机会越大，从而推测，那个函数（或者pid等）的CPU占用率就越高。

当然，如果某个进程运气特别好，它每次都刚好躲过你发起探测的位置，你的统计结果可能就完全是错的了。这是所有采样统计都有可能遇到的问题了。

类似git，docker等多功能工具，perf也是使用perf <子命令>这种模式。所有人首先需要学习的是两个最简单的命令：perf list和perf top。

perf list列出perf可以支持的所有事件。例如这样：



perf top可以动态收集和更新统计列表，它支持很多参数，关键要记住两个参数：

-e 指定跟踪的事件

-e可以指定前面perf list提供的所有事件（包括没有列出的tracepoint），可以用多个-e指定多个事件同时跟踪（但显示的时候会分开显示）

一个-e也可以直接指定多个事件，中间用逗号隔开即可：

sudo perf top -e branch-misses,cycles

事件可以指定后缀，比如我想只跟踪发生在用户态时产生的分支预测失败，我可以这样：

sudo perf top -e branch-misses:u,cycles

全部事件都有这个要求，我还可以：

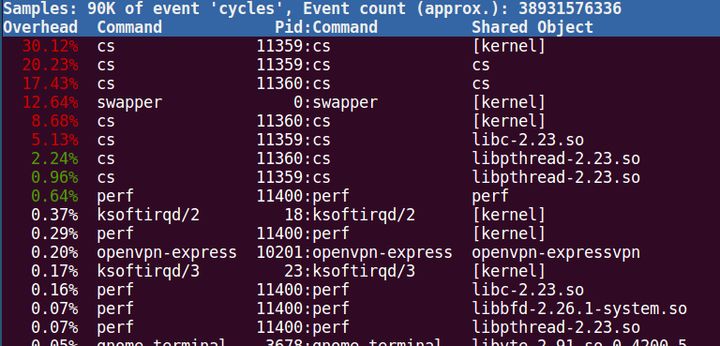
sudo perf top -e ‘{branch-misses,cycles}:u'

看看perf-list的手册，会找到更多的后缀。

-s指定按什么参数来进行分类

-s参数可以不使用，默认会按函数进行分类，但如果你想按pid来分，就需要靠-s来进行分类了。前面我们已经看过这样的例子了。-s也可以指定多个域（用逗号隔开），例如这样：

sudo perf top -e 'cycles' -s comm,pid,dso



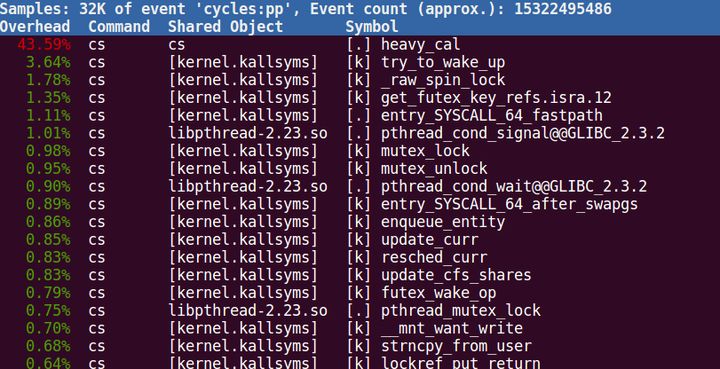
perf-top用来理解，体会perf的功能是比较好的，但实践中用得不多，用得比较多的是perf-record和perf-report命令。perf-record用来启动一次跟踪，而perf-report用来输出跟踪结果。

一般的过程是：

sudo perf record -e 'cycles' -- myapplication arg1 arg2

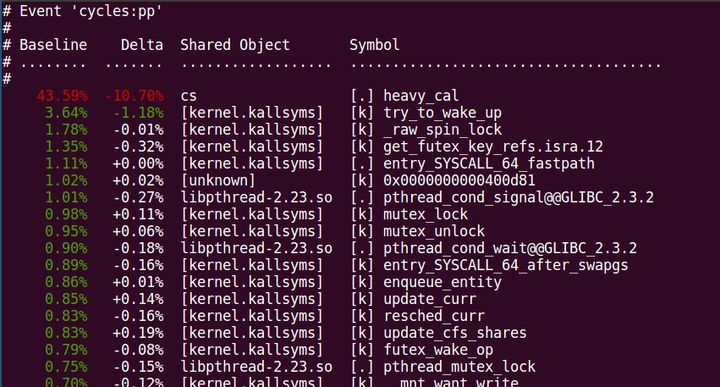
sudo perf report

下面是一个报告的例子：



perf record在当前目录产生一个perf.data文件（如果这个文件已经存在，旧的文件会被改名为perf.data.old），用来记录过程数据。之后运行的perf report命令会输出统计的结果。perf.data只包含原始数据，perf report需要访问本地的符号表，pid和进程的对应关系等信息来生成报告。所以perf.data不能直接拷贝到其他机器上用的。但你可以通过perf-archive命令把所有这些数据打包，这样移到另一个机器上就可以用了。注意，perf-archive是指perf-archive这个命令，不是指perf archive这个子命令。这个命令在编译perf源代码的时候会产生的，如果你的发行版不支持，可以自己编译一个。比较可惜的是，perf-archive备份的代码不能跨平台使用（比如你从arm平台上备份的数据，在x86上是分析不了的）。

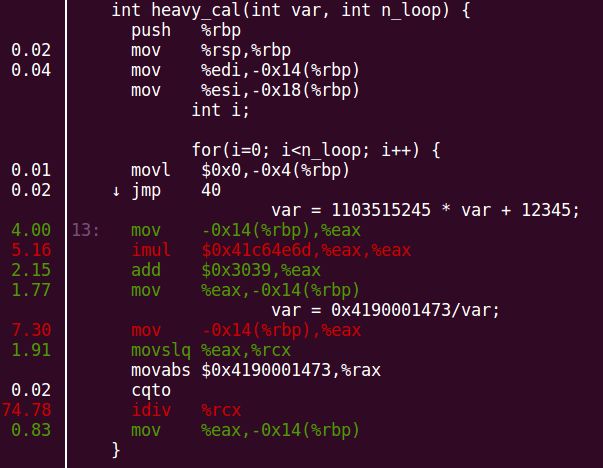
perf.data保留前一个版本，可以支持perf diff这个命令，这个命令比较两次两次运行的区别。这样你可以用不同参数运行你的程序，看看运行结果有什么不同，用前面这个cs程序为例，我用4线程对比2线程，就有如下结果：



这里看到，增加线程后，heavy\_cal的占比大幅下降了10.70%，其他的变化不大。

perf record不一定用于跟踪自己启动的进程，通过指定pid，可以直接跟踪固定的一组进程。

perf report是一个菜单接口，可以一直展开到每个函数的代码的，例如我们要展开上面这个heavy\_cal()函数的具体计数，我们在上面回车，选择代码分析，我们可以得到：



Perf top

实时显示系统/进程的性能统计信息，常用参数

-e：指定性能事件

-a：显示在所有CPU上的性能统计信息

-C：显示在指定CPU上的性能统计信息

-p：指定进程PID

-t：指定线程TID

-K：隐藏内核统计信息

-U：隐藏用户空间的统计信息

-s：指定待解析的符号信息

‘‐G’ or‘‐‐call‐graph’ <output\_type,min\_percent,call\_order>

graph: 使用调用树，将每条调用路径进一步折叠。这种显示方式更加直观。

每条调用路径的采样率为绝对值。也就是该条路径占整个采样域的比率。

fractal

默认选项。类似与 graph，但是每条路径前的采样率为相对值。

flat

不折叠各条调用

选项 call\_order 用以设定调用图谱的显示顺序，该选项有 2个取值，分别是callee 与caller。将该选项设为callee 时，perf按照被调用的顺序显示调用图谱，上层函数被下层函数所调用。

该选项被设为caller 时，按照调用顺序显示调用图谱，即上层函数调用了下层函数路径，也不显示每条调用路径的采样率

注： Perf top需要root权限

**查看进程有多少线程**

cat/proc/{pid}/status | grep -i threads

ls/proc/<pid>/task | wc -l

ps hH p {pid} | wc -l

ps -xhH #查看所有线程

**pstree命令**

以树形结构显示程序和进程之间的关系:

-a 显示启动每个进程对应的完整指令，包括启动进程的路径、参数等。

-n 根据进程 PID 号来排序输出，默认是以程序名排序输出的。

-p 显示进程的 PID。

-u 显示进程对应的用户名称。

**sar命令**

sar命令是Linux下系统运行状态统计工具，对系统当前的状态进行取样，然后计算数据和比例来表达系统的当前运行状态。它的特点是可以连续对系统取样，获得大量的取样数据。取样数据和分析的结果都可以存入文件，使用它消耗的系统资源很小。

-A：显示所有的报告信息；

-b：显示I/O速率；

-B：显示换页状态；

-c：显示进程创建活动；

-d：显示每个块设备的状态；

-e：设置显示报告的结束时间；

-f：从指定文件提取报告；

-i：设状态信息刷新的间隔时间；

-P：报告每个CPU的状态；

-R：显示内存状态；

-u：显示CPU利用率；

-v：显示索引节点，文件和其他内核表的状态；

-w：显示交换分区状态；

-x：显示给定进程的状态。

**CPU资源监控：**

每5s采样一次，连续采样3次，观察cpu使用情况，并将采样结果以二进制形式存入test\_sar中（查看二进制文件test中的内容，sar命令：sar -u -f test\_sar）

sar -u -o test\_sar 5 3

Linux 3.10.0-1.0.1.el7.x86\_64 (bjrz-c2064.lf) 10/19/2021 \_x86\_64\_ (56 CPU)

05:25:23 PM CPU %user %nice %system %iowait %steal %idle

05:25:28 PM all 69.91 0.00 8.33 0.01 0.00 21.75

05:25:33 PM all 73.42 0.00 8.28 0.00 0.00 18.30

05:25:38 PM all 43.53 0.00 3.70 0.00 0.00 52.77

Average: all 62.28 0.00 6.77 0.00 0.00 30.95

输出项说明：

CPU：all表示统计信息为所有 CPU的平均值。

%user：显示在用户级别(application)运行使用 CPU总时间的百分比。

%nice：显示在用户级别，用于nice操作，所占用 CPU总时间的百分比。

%system：在核心级别(kernel)运行所使用 CPU总时间的百分比。

%iowait：显示用于等待I/O操作占用 CPU总时间的百分比。

%steal：管理程序(hypervisor)为另一个虚拟进程提供服务而等待虚拟CPU 的百分比。

%idle：显示 CPU空闲时间占用 CPU总时间的百分比。

P.S：

1.若%iowait的值过高，表示硬盘存在I/O瓶颈

2.若%idle的值高但系统响应慢时，有可能是CPU等待分配内存，此时应加大内存容量

3.若%idle的值持续低于1，则系统的CPU处理能力相对较低，表明系统中最需要解决的资源是CPU。

**inode、文件和其他内核表监控：**

每5秒采样一次，连续采样10次，观察核心表的状态

sar -v 3 4

Linux 3.10.0-1.0.1.el7.x86\_64 (bjrz-c2064.lf) 10/19/2021 \_x86\_64\_ (56 CPU)

05:28:12 PM dentunusd file-nr inode-nr pty-nr

05:28:15 PM 1205980 70336 318152 1

05:28:18 PM 1205993 70224 318158 1

05:28:21 PM 1206012 70112 318181 1

05:28:24 PM 1206103 70112 318294 1

Average: 1206022 70196 318196 1

输出项说明：

dentunusd：目录高速缓存中未被使用的条目数量

file-nr：文件句柄（filehandle）的使用数量

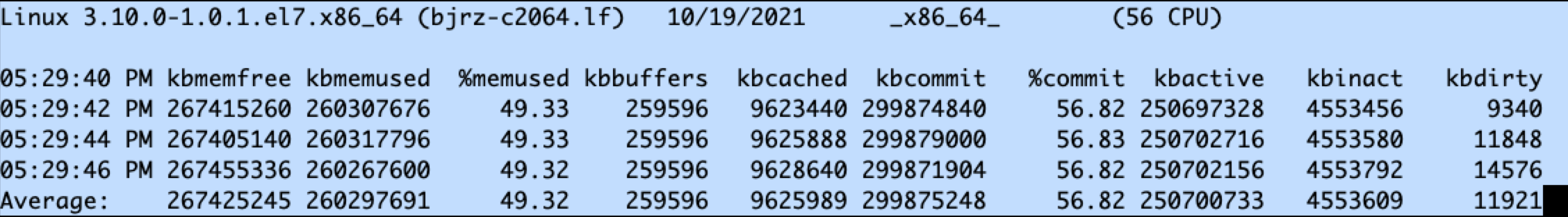
inode-nr：索引节点句柄（inodehandle）的使用数量

pty-nr：使用的pty数量

**内存和交换空间监控：**

每5s采样一次，连续采样10次，监控内存分页

sar -r 2 3



输出项说明：

kbmemfree：这个值和free命令中的free值基本一致,所以它不包括buffer和cache的空间.

kbmemused：这个值和free命令中的used值基本一致,所以它包括buffer和cache的空间.

%memused：这个值是kbmemused和内存总量(不包括swap)的一个百分比.

kbbuffers和kbcached：这两个值就是free命令中的buffer和cache.

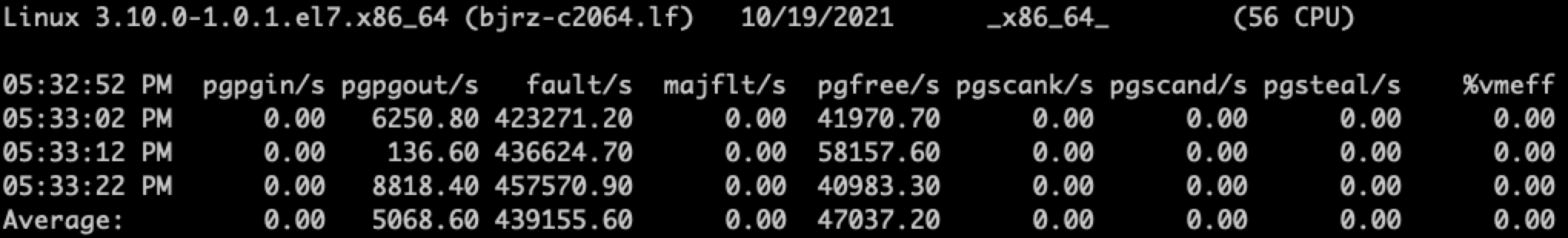
kbcommit：保证当前系统所需要的内存,即为了确保不溢出而需要的内存(RAM+swap).

%commit：这个值是kbcommit与内存总量(包括swap)的一个百分比.

**内存分页监控：**

每10s采样一次，连续采样3次，监控内存分页

sar -B 10 3



输出项说明：

pgpgin/s：表示每秒从磁盘或SWAP置换到内存的字节数(KB)

pgpgout/s：表示每秒从内存置换到磁盘或SWAP的字节数(KB)

fault/s：每秒钟系统产生的缺页数,即主缺页与次缺页之和(major +minor)

majflt/s：每秒钟产生的主缺页数.

pgfree/s：每秒被放入空闲队列中的页个数

pgscank/s：每秒被kswapd扫描的页个数

pgscand/s：每秒直接被扫描的页个数

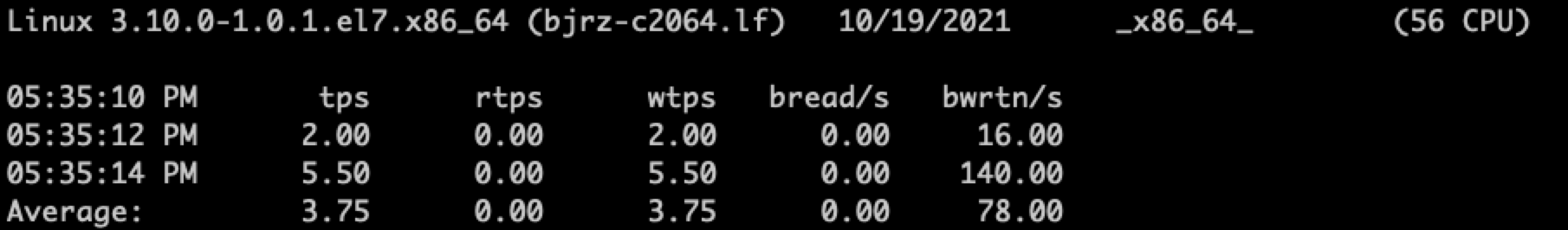
pgsteal/s：每秒钟从cache中被清除来满足内存需要的页个数

%vmeff：每秒清除的页(pgsteal)占总扫描页(pgscank+pgscand)的百分比

**I/O和传送速率监控：**

每5s采样一次，连续采样10次，报告缓冲区的使用情况

sar -b 2 2



输出项说明：

tps：每秒钟物理设备的 I/O传输总量

rtps：每秒钟从物理设备读入的数据总量

wtps：每秒钟向物理设备写入的数据总量

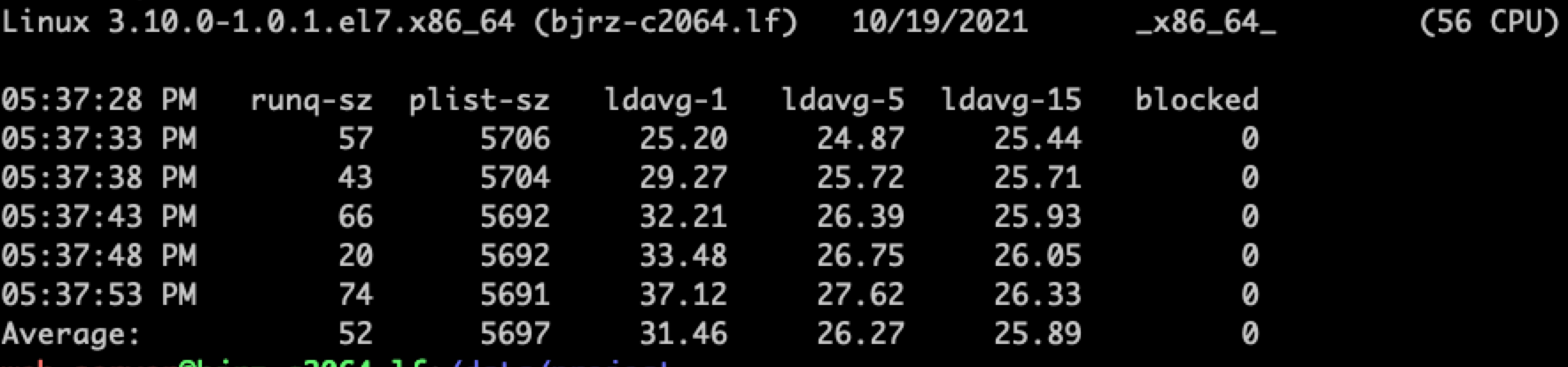
bread/s：每秒钟从物理设备读入的数据量，单位为块/s

bwrtn/s：每秒钟向物理设备写入的数据量，单位为块/s

**进程队列长度和平均负载状态监控：**

每5s采样一次，连续采样10次，监控进程队列长度和平均负载状态

sar -q 5 5



输出项说明：

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（Systemload average）

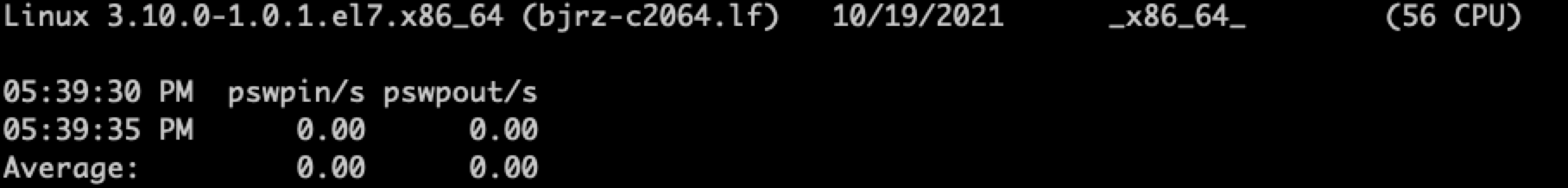
ldavg-5：过去5分钟的系统平均负载

ldavg-15：过去15分钟的系统平均负载

**系统交换活动信息监控：**

每5s采样一次，连续采样10次，监控系统交换活动信息

sar -W 5 1



输出项说明：

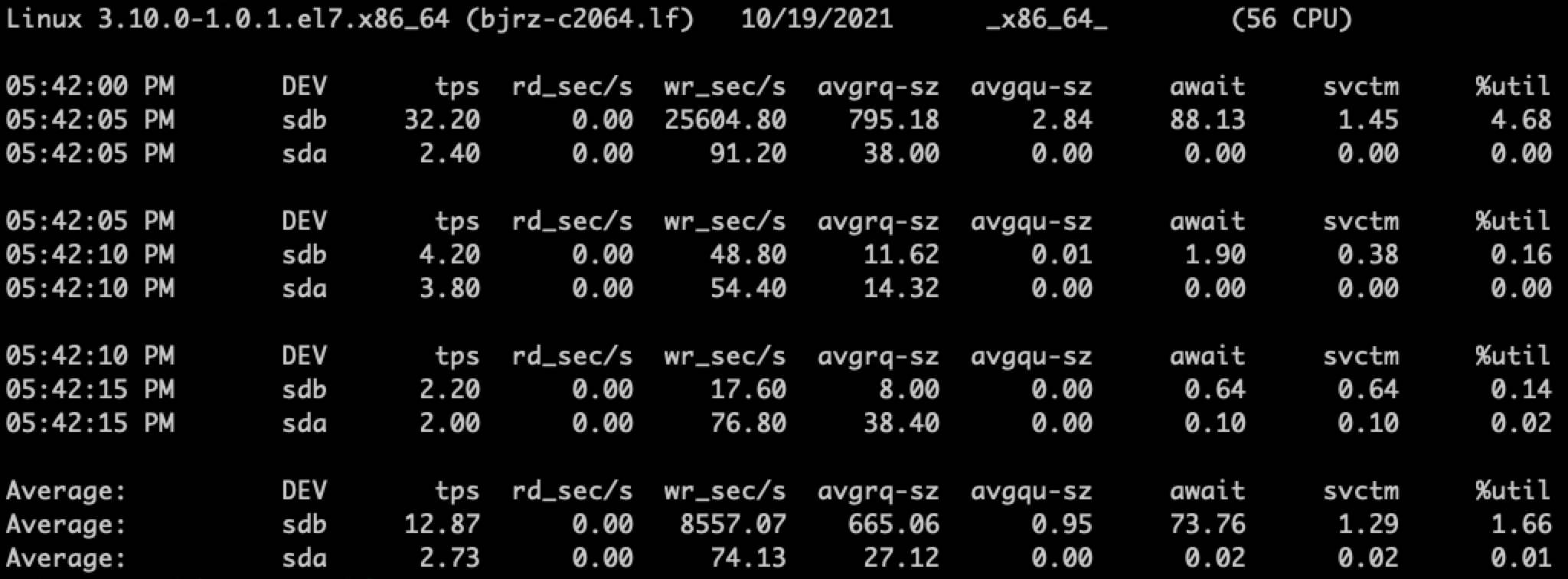
pswpin/s：每秒系统换入的交换页面（swap page）数量

pswpout/s：每秒系统换出的交换页面（swap page）数量

**设备使用情况监控：**

每5s采样一次，连续采样10次，报告设备使用情况

sar -d 5 3 -p



-p可以打印出sda,hdc等磁盘设备名称,如果不用参数-p,设备节点则有可能是dev8-0,dev22-0

输出项说明：

tps:每秒从物理磁盘I/O的次数.多个逻辑请求会被合并为一个I/O磁盘请求,一次传输的大小是不确定的.

rd\_sec/s:每秒读扇区的次数.

wr\_sec/s:每秒写扇区的次数.

avgrq-sz:平均每次设备I/O操作的数据大小(扇区).

avgqu-sz:磁盘请求队列的平均长度.

await:从请求磁盘操作到系统完成处理,每次请求的平均消耗时间,包括请求队列等待时间,单位是毫秒(1秒=1000毫秒).

svctm:系统处理每次请求的平均时间,不包括在请求队列中消耗的时间.

%util:I/O请求占CPU的百分比,比率越大,说明越饱和.

avgqu-sz的值较低时，设备的利用率较高。

当%util的值接近1%时，表示设备带宽已经占满

**总结**

要判断系统瓶颈问题，有时需几个 sar 命令选项结合起来

怀疑CPU存在瓶颈，可用sar -u和sar -q等来查看

怀疑内存存在瓶颈，可用sar -B、sar -r和sar -W等来查看

怀疑I/O存在瓶颈，可用sar -b、sar -u和sar -d等来查看

**addr2line命令**

可以将指令地址转换成文件名、函数名和行号。

常用的是以下参数

-e 指定可执行bin

-a 显示函数地址

-f 显示函数名称

addr2line -e dryrun\_nearby\_realtime\_reco\_leaf -f -a 0x3666fab

以上命令将会显示dryrun\_nearby\_realtime\_reco\_leaf，在地址为0x3666fab,对应的函数名称、地址及源代码信息。执行结果如下：

0x0000000003666fab

\_ZN22TCMallocImplementation16GetAllocatedSizeEPKv

/home/wangtian/dev/gperftools/src/tcmalloc.cc:1509

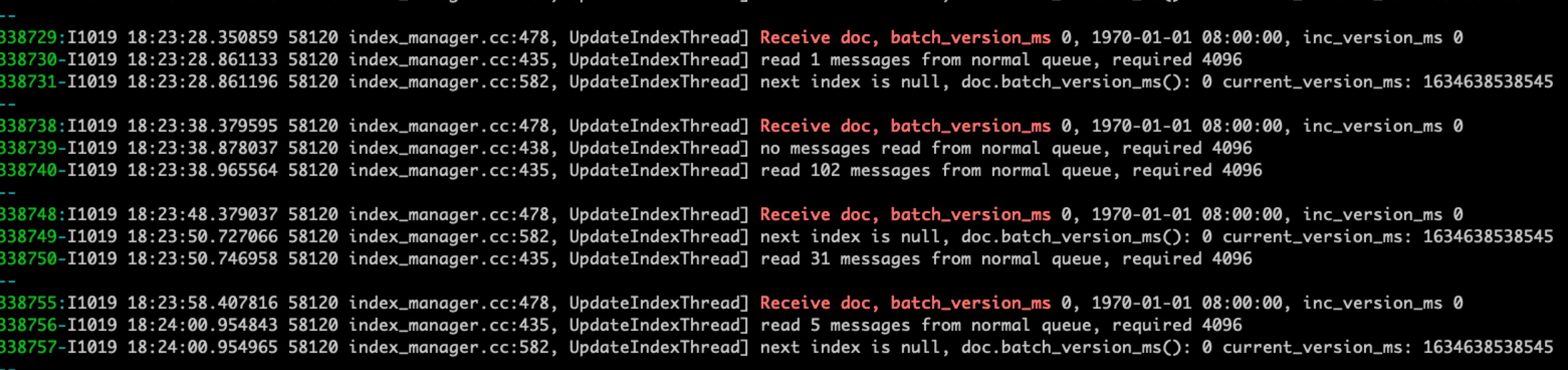
**grep命令**

可以将指令地址转换成文件名、函数名和行号。

-A<显示行数> 或 --after-context=<显示行数>: 除了显示符合范本样式的那一行之外，并显示该行之后的内容。

grep -rn -A2 "Receive doc, batch\_version\_ms" leaf.INFO

输出



**nm命令**

显示bin的符号表。

-a或--debug-syms：显示调试符号。

-B：等同于--format=bsd，用来兼容MIPS的nm。

-C或--demangle：将低级符号名解码(demangle)成用户级名字。这样可以使得C++函数名具有可读性。  
-D或--dynamic：显示动态符号。该任选项仅对于动态目标(例如特定类型的共享库)有意义。  
-f format：使用format格式输出。format可以选取bsd、sysv或posix，该选项在GNU的nm中有用。默认为bsd。  
-g或--extern-only：仅显示外部符号。  
-n、-v或--numeric-sort：按符号对应地址的顺序排序，而非按符号名的字符顺序。  
-p或--no-sort：按目标文件中遇到的符号顺序显示，不排序。  
-P或--portability：使用POSIX.2标准输出格式代替默认的输出格式。等同于使用任选项-f posix。

-s或--print-armap：当列出库中成员的符号时，包含索引。索引的内容包含：哪些模块包含哪些名字的映射。

-r或--reverse-sort：反转排序的顺序(例如，升序变为降序)。  
--size-sort：按大小排列符号顺序。该大小是按照一个符号的值与它下一个符号的值进行计算的。  
-t radix或--radix=radix：使用radix进制显示符号值。radix只能为“d”表示十进制、“o”表示八进制或“x”表示十六进制。  
--target=bfdname：指定一个目标代码的格式，而非使用系统的默认格式。  
-u或--undefined-only：仅显示没有定义的符号(那些外部符号)。  
-l或--line-numbers：对每个符号，使用调试信息来试图找到文件名和行号。对于已定义的符号，查找符号地址的行号。对于未定义符号，查找指向符号重定位入口的行号。如果可以找到行号信息，显示在符号信息之后。  
-V或--version：显示nm的版本号。

**输出符号类型说明**

A 该符号的值是绝对的，在以后的链接过程中，不允许进行改变。这样的符号值，常常出现在中断向量表中，例如用符号来表示各个中断向量函数在中断向量表中的位置。

B 该符号的值出现在非初始化数据段(bss)中。例如，在一个文件中定义全局static int test。则该符号test的类型为b，位于bss section中。其值表示该符号在bss段中的偏移。一般而言，bss段分配于RAM中  
C 该符号为common。common symbol是未初始话数据段。该符号没有包含于一个普通section中。只有在链接过程中才进行分配。符号的值表示该符号需要的字节数。例如在一个c文件中，定义int test，并且该符号在别的地方会被引用，则该符号类型即为C。否则其类型为B。  
D 该符号位于初始话数据段中。一般来说，分配到data section中。例如定义全局int baud\_table[5] = {9600, 19200, 38400, 57600, 115200}，则会分配于初始化数据段中。

G 该符号也位于初始化数据段中。主要用于small object提高访问small data object的一种方式。

I 该符号是对另一个符号的间接引用。

N 该符号是一个debugging符号。

R 该符号位于只读数据区。例如定义全局const int test[] = {123, 123};则test就是一个只读数据区的符号。注意在cygwin下如果使用gcc直接编译成MZ格式时，源文件中的test对应\_test，并且其符号类型为D，即初始化数据段中。但是如果使用m6812-elf-gcc这样的交叉编译工具，源文件中的test对应目标文件的test,即没有添加下划线，并且其符号类型为R。一般而言，位于rodata section。值得注意的是，如果在一个函数中定义const char \*test = “abc”, const char test\_int = 3。使用nm都不会得到符号信息，但是字符串“abc”分配于只读存储器中，test在rodata section中，大小为4。  
S 符号位于非初始化数据区，用于small object。  
T 该符号位于代码区text section。  
U 该符号在当前文件中是未定义的，即该符号的定义在别的文件中。例如，当前文件调用另一个文件中定义的函数，在这个被调用的函数在当前就是未定义的；但是在定义它的文件中类型是T。但是对于全局变量来说，在定义它的文件中，其符号类型为C，在使用它的文件中，其类型为U。  
V 该符号是一个weak object。  
W The symbol is a weak symbol that has not been specifically tagged as a weak object symbol.  
- 该符号是a.out格式文件中的stabs symbol。  
? 该符号类型没有定义

**readelf命令**

显示elf信息。

-a --all 相当于指定--file-header --program-headers --sections --symbols --relocs --dynamic --notes和--version-info。

-h --file-header 显示文件开头的ELF头中包含的信息

-l --program-headers --segments 显示文件的segment头中包含的信息（如果有）。

-S --sections --section-headers 显示文件的section头中包含的信息（如果有）。

-g --section-groups 显示文件的section group中包含的信息（如果有）。

-t --section-details 显示详细的section信息。意味着-S。

-s --symbols --syms 显示文件的symbol table section中的条目（如果有）。

-e --headers 显示文件中的所有headers。相当于-h -l -S。

-n --notes 显示NOTE segment(和/或)section的内容（如果有）。

-r --relocs 显示文件relocation section的内容（如果有）。

-u --unwind 显示文件的unwind section的内容（如果有）。目前仅支持IA64 ELF文件的unwind section

-d --dynamic 显示文件dynamic section的内容（如果有）。

-V --version-info 显示文件中version section的内容，如果存在。

-A --arch-specific 显示文件中特定于体系结构的信息（如果有）。

-D --use-dynamic 显示符号时，此选项使readelf使用文件dynamic section中的符号表，而不是symbol section中的符号表。

-x <number or name> --hex-dump=<number or name> 以十六进制字节显示indecated section的内容。一个数字通过索引section表标识一个特定的section;任何其他string标识目标文件中具有该名称的所有section。

-R <number or name> --relocated-dump=<number or name> 以十六进制字节显示indecated section的内容。一个数字通过索引section表标识一个特定的section;任何其他string标识目标文件中具有该名称的所有section。该section的内容将在显示之前重新定位。

-p <number or name> --string-dump=<number or name> 将indicated section的内容显示为可打印字符串。一个数字通过索引section表标识一个特定的section;任何其他string标识目标文件中具有该名称的所有section。

-c --archive-index 显示二进制归档的header部分中包含的文件符号索引信息。对ar执行与t命令相同的功能，但不使用BFD库。

-w[lLiaprmfFsoR] --debug-dump[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges] 显示文件中debug section的内容（如果存在）。 如果后面有一个可选的字母或单词，则只会转储在这些特定部分中找到的数据。  
注意：=codedline选项将显示.debug\_line section解释后的内容，而=rawline选项将以原始格式转储内容。

-I --histogram 显示符号表的内容时，显示桶列表长度的直方图。

-v --version 显示readelf的版本号。

-W --wide 不破坏输出线以适应80列。 默认情况下，readelf会断开64位ELF文件的section header和segment 列表行，以便它们适合80列。 此选项使readelf打印每个节标题resp。 每个segment只有一行，在80列以上的终端上可读性更高。

**objdump命令**

objdump命令是用查看目标文件或者可执行的目标文件的构成的gcc工具。

-a --archive-headers

# 显示档案库的成员信息,类似ls -l将lib\*.a的信息列出。

-b bfdname --target=bfdname

# 指定目标码格式。这不是必须的，objdump能自动识别许多格式，比如：

objdump -b oasys -m vax -h fu.o

# 显示fu.o的头部摘要信息，明确指出该文件是Vax系统下用Oasys编译器生成的目标文件。objdump -i将给出这里可以指定的目标码格式列表。

-C --demangle

# 将底层的符号名解码成用户级名字，除了去掉所开头的下划线之外，还使得C++函数名以可理解的方式显示出来。

--debugging

-g

# 显示调试信息。企图解析保存在文件中的调试信息并以C语言的语法显示出来。仅仅支持某些类型的调试信息。有些其他的格式被readelf -w支持。

-e --debugging-tags

# 类似-g选项，但是生成的信息是和ctags工具相兼容的格式。

--disassemble

-d

# 从objfile中反汇编那些特定指令机器码的section。

-D --disassemble-all

# 与 -d 类似，但反汇编所有section.

--prefix-addresses

# 反汇编的时候，显示每一行的完整地址。这是一种比较老的反汇编格式。

-EB

-EL

--endian={big|little}

# 指定目标文件的小端。这个项将影响反汇编出来的指令。在反汇编的文件没描述小端信息的时候用。例如S-records.

-f

--file-headers

显示objfile中每个文件的整体头部摘要信息。

-h

--section-headers

--headers

显示目标文件各个section的头部摘要信息。

-H

--help

简短的帮助信息。

-i

--info

显示对于 -b 或者 -m 选项可用的架构和目标格式列表。

-j name

--section=name

仅仅显示指定名称为name的section的信息

-l

--line-numbers

用文件名和行号标注相应的目标代码，仅仅和-d、-D或者-r一起使用使用-ld和使用-d的区别不是很大，在源码级调试的时候有用，要求编译时使用了-g之类的调试编译选项。

-m machine

--architecture=machine

指定反汇编目标文件时使用的架构，当待反汇编文件本身没描述架构信息的时候(比如S-records)，这个选项很有用。可以用-i选项列出这里能够指定的架构.

--reloc

-r

显示文件的重定位入口。如果和-d或者-D一起使用，重定位部分以反汇编后的格式显示出来。

--dynamic-reloc

-R

显示文件的动态重定位入口，仅仅对于动态目标文件意义，比如某些共享库。

-s

--full-contents

显示指定section的完整内容。默认所有的非空section都会被显示。

-S

--source

尽可能反汇编出源代码，尤其当编译的时候指定了-g这种调试参数时，效果比较明显。隐含了-d参数。

--show-raw-insn

反汇编的时候，显示每条汇编指令对应的机器码，如不指定--prefix-addresses，这将是缺省选项。

--no-show-raw-insn

反汇编时，不显示汇编指令的机器码，如不指定--prefix-addresses，这将是缺省选项。

--start-address=address

从指定地址开始显示数据，该选项影响-d、-r和-s选项的输出。

--stop-address=address

显示数据直到指定地址为止，该项影响-d、-r和-s选项的输出。

-t

--syms

显示文件的符号表入口。类似于nm -s提供的信息

-T

--dynamic-syms

显示文件的动态符号表入口，仅仅对动态目标文件意义，比如某些共享库。它显示的信息类似于 nm -D|--dynamic 显示的信息。

-V

--version

版本信息

--all-headers

-x

显示所可用的头信息，包括符号表、重定位入口。-x 等价于-a -f -h -r -t 同时指定。

-z

--disassemble-zeroes

一般反汇编输出将省略大块的零，该选项使得这些零块也被反汇编。

@file 可以将选项集中到一个文件中，然后使用这个@file选项载入。

**tcpdump简明教程**

**选项**

**-i any** 监听所有的网卡接口，用来查看是否有网络流量

**-i eth0** 只监听eth0网卡接口

**-D** 显示可用的接口列表

**-n** 不要解析主机名

**-nn** 不要解析主机名或者端口名

**-q** 显示更少的输出(更加quiet)

**-t** 输出可读的时间戳

**-tttt** 输出最大程度可读的时间戳

**-X** 以hex和ASCII两种形式显示包的内容

**-XX** 与\*\*-X\*\*类似，增加以太网header的显示

**-v, -vv, -vvv** 显示更加多的包信息

**-c** 只读取x个包，然后停止

**-s** 指定每一个包捕获的长度，单位是byte，使用-s0可以捕获整个包的内容

**-S** 输出绝对的序列号

**-e** 获取以太网header

**-E** 使用提供的秘钥解密IPSEC流量

**表达式**

在**tcpdump**中，可以使用表达式过滤指定类型的流量。有三种主要的表达式类型：**type**，**dir**，**proto**。

类型（type）选项包含：**host**，**net**，**port**

方向（dir）选项包含：**src**，**dst**

协议（proto）选项包含：**tcp**，**udp**，**icmp**，**ah**等

**示例[tcpdump简明教程](https://github.com/mylxsw/growing-up/blob/master/doc/tcpdump%E7%AE%80%E6%98%8E%E6%95%99%E7%A8%8B.md)**

捕获所有流量

查看所有网卡接口上发生了什么

tcpdump -i any

指定网卡接口

查看指定网卡上发生了什么

tcpdump -i eth0

原生输出

查看更多的信息，不解析主机名和端口号，显示绝对序列号，可读的时间戳

tcpdump -ttttnnvvS

查看指定IP的流量

这是最常见的方式，这里只查看来自或者发送到IP地址1.2.3.4的流量。

tcpdump host 1.2.3.4

查看更多的包信息，输出HEX

当你需要查看包中的内容时，使用hex格式输出是非常有用的。

# tcpdump -nnvXSs 0 -c1 icmp

tcpdump: data link type PKTAP

tcpdump: listening on pktap, link-type PKTAP (Apple DLT\_PKTAP), capture size 262144 bytes

16:08:16.791604 IP (tos 0x0, ttl 64, id 34318, offset 0, flags [none], proto ICMP (1), length 56)

192.168.102.35 > 114.114.114.114: ICMP 192.168.102.35 udp port 50694 unreachable, length 36

IP (tos 0x0, ttl 152, id 0, offset 0, flags [none], proto UDP (17), length 112)

114.114.114.114.53 > 192.168.102.35.50694: [|domain]

0x0000: 5869 6c88 7f64 784f 4392 ed7e 0800 4500 Xil..dxOC..~..E.

0x0010: 0038 860e 0000 4001 e906 c0a8 6623 7272 .8....@.....f#rr

0x0020: 7272 0303 3665 0000 0000 4500 0070 0000 rr..6e....E..p..

0x0030: 0000 9811 16cd 7272 7272 c0a8 6623 0035 ......rrrr..f#.5

0x0040: c606 005c 0000 ...\..

1 packet captured

357 packets received by filter

0 packets dropped by kernel

使用源和目的地址过滤

tcpdump src 2.3.4.6

tcpdump dst 3.4.5.6

过滤某个子网的数据包

tcpdump net 1.2.3.0/24

过滤指定端口相关的流量

tcpdump port 3389

tcpdump src port 1025

过滤指定协议的流量

tcpdump icmp

只显示IPV6流量

tcpdump ip6

使用端口范围过滤

tcpdump portrange 21-23

基于包的大小过滤流量

tcpdump less 32

tcpdump greater 64

tcpdump <=128

将捕获的内容写入文件

使用-w选项可以将捕获的数据包信息写入文件以供以后分析，这些文件就是著名的PCAP(PEE-cap)文件，很多应用都可以处理它。

tcpdump port 80 -w capture\_file

使用tcpdump加载之前保存的文件进行分析

tcpdump -r capture\_file