**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。 |
| 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

**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

**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。

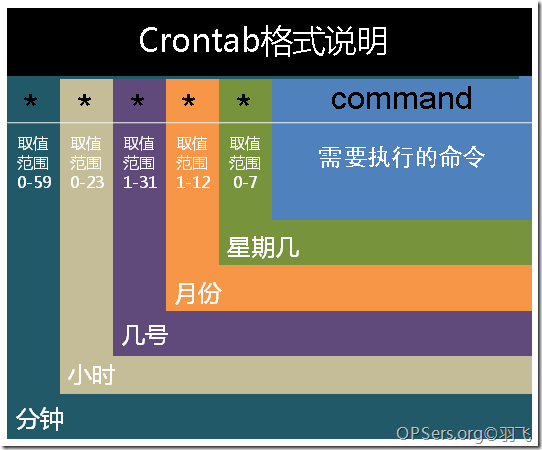
hour：表示小时，可以是从0到23之间的任何整数。

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

**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

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

**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