一篇文章搞定【Linux】

陈同学 陈同学在搬砖 2021-12-17 12:00

往期汇总

一篇文章搞定【所有】Cpp八股文
一篇文章搞定【所有】数据库八股文
一篇文章搞定【所有】网络编程八股文
一篇文章搞定【所有】STL八股文
一篇文章搞定【所有】计算机网络八股文
一篇文章搞定【所有】操作系统八股文

文件相关命令

文件属性操作

文件属性组成

在这里插入图片描述

Linux上的一个文件属性组成如上,需要注意的是第二部分,格式如下

在这里插入图片描述

文件类型

```
当为[ d]则是目录
当为[ -]则是普通文件;
若是[ l]则表示为链接文档(link file);
若是[ b]则表示为装置文件里面的可供储存的接口设备(可随机存取装置);
若是[ c]则表示为装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置);
若是[s ]则表示为套接接口文件;
若是[ p]则表示为管道。
```

文件权限

```
包括
文件所有者权限
文件所属群组权限
其他用户权限
```

chgrp

功能

```
chgrp命令用来改变指定文件所属的用户组。
如果用户不是该文件的文件主或超级用户(root)，则不能改变该文件的组。
```

输入

```
chgrp [-cfhRv][所属群组][文件或目录]
```

```
-c或--changes：效果类似“-v”参数，但仅回报更改的部分；
-f或--quiet或--silent：不显示错误信息；
-h或--no-dereference：只对符号连接的文件作修改，而不是该其他任何相关文件；
-R或--recursive：递归处理，将指令目录下的所有文件及子目录一并处理；
-v或--verbose：显示指令执行过程；
```

其中，组名可以是用户组的id，也可以是用户组的组名。文件名可以 是由空格分开的要改变属组的文件列表，也可以是由通配符描述的文件集合。

实例

```
将文件a的用户组从root改为mcc
sudo chgrp mcc a
```

chown

功能

chown将指定文件的拥有者改为指定的用户或组， 普通用户不能将自己的文件改变成其他的拥有者。其操作权限一般为管理员root。

输入

```
chown [-cfhvR] user[:group] file.
```

```
文件是以空格分开的要改变权限的文件列表，支持通配符。
user ： 用户可以是用户名或者用户ID;
group ： 组可以是组名或者组ID;
-c ： 显示更改的部分的信息
-f ： 忽略错误信息
-h ：修复符号链接
-v ： 显示详细的处理信息
-R ： 处理指定目录以及其子目录下的所有文件
```

实例

```
改变拥有者和群组
    chown mail:mail log2012.log
改变文件拥有者
    chown root: log2012.log
改变文件群组
    chown :mail log2012.log
改变指定目录以及其子目录下的所有文件的拥有者和群组
    chown -R -v root:mail test6
```

chmod

功能

改变文件的权限

输入

```
chmod [-cfvR] [--help] [--version] mode file...
参数说明
mode : 权限设定字串，格式如下 :

[ugoa...][[+-=][rwxX]...][,...]
其中:

u 表示该文件的拥有者，g 表示与该文件的拥有者属于同一个群体(group)者，o 表示其他以外的人，a 表示这三者皆是。
+ 表示增加权限、- 表示取消权限、= 表示唯一设定权限。
r 表示可读取，w 表示可写入，x 表示可执行，X 表示只有当该文件是个子目录或者该文件已经被设定过为可执行。
其他参数说明:

-c : 若该文件权限确实已经更改，才显示其更改动作
-f : 若该文件权限无法被更改也不要显示错误讯息
-v : 显示权限变更的详细资料
-R : 对目前目录下的所有文件与子目录进行相同的权限变更(即以递回的方式逐个变更)
--help : 显示辅助说明
--version : 显示版本
```

实例

将文件 file1.txt 设为所有人皆可读取 ：

chmod ugo+r file1.txt
将文件 file1.txt 设为所有人皆可读取 ：

chmod a+r file1.txt
将文件 file1.txt 与 file2.txt 设为该文件拥有者，与其所属同一个群体者可写入，但其他以外的人则不可写入 ：

chmod ug+w,o-w file1.txt file2.txt
将 ex1.py 设定为只有该文件拥有者可以执行 ：

chmod u+x ex1.py
将目前目录下的所有文件与子目录皆设为任何人可读取 ：

chmod -R a+r *
此外chmod也可以用数字来表示权限如 ：

chmod 777 file
语法为：

chmod abc file
其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。

r=4，w=2，x=1
若要rwx属性则4+2+1=7；
若要rw-属性则4+2=6；
若要r-x属性则4+1=5。
chmod a=rwx file
和

chmod 777 file
效果相同

chmod ug=rwx,o=x file
和

chmod 771 file

umask

功能

Linux umask命令指定在建立文件时预设的权限掩码。

umask可用来设定[权限掩码]。[权限掩码]是由3个八进制的数字所组成，

将现有的存取权限减掉权限掩码后，即可产生建立文件时预设的权限。

输入

umask [-S][权限掩码]

实例

```
查看文件掩码
kylechen@kyle:~$ umask
0022
```

```
以非数字的形式查看文件掩码
kylechen@kyle:~$ umask -S
u=rwx,g=rx,o=rx
```

```
更改文件掩码
kylechen@kyle:~$ umask 002
kylechen@kyle:~$ umask
0002
```

test

功能

测试文件的某项属性

输入

```
test [参数][文件]
```

实例

实例一:测试文件 /dmtsai 是否存在

文件目录操作

cd

功能

变换目录

pwd

功能

显示当前目录

mkdir

功能

建立一个新的目录

rmdir

功能

删除一个空的目录

tree

功能

树状列出目录内容

文件整体操作

ls

功能

```
ls命令用于显示指定工作目录下之内容
```

输入

```
 ls [-alrtAFR] [name]

参数 :

-a 显示所有文件及目录 (ls内定将文件名或目录名称开头为"."的视为隐藏档，不会列出)
-l 除文件名称外，亦将文件型态、权限、拥有者、文件大小等资讯详细列出
-r 将文件以相反次序显示(原定依英文字母次序)
-t 将文件依建立时间之先后次序列出
-A 同 -a ，但不列出 "." (目前目录) 及 ".." (父目录)
-F 在列出的文件名称后加一符号；例如可执行档则加 "*", 目录则加 "/"
-R 若目录下有文件，则以下之文件亦皆依序列出
```

实例

列出根目录(\)下的所有目录：

```
ls /
bin               dev   lib         media  net   root     srv  upload  www
boot              etc   lib64       misc   opt   sbin     sys  usr
home  lost+found  mnt    proc  selinux  tmp  var
列出目前工作目录下所有名称是 s 开头的文件，越新的排越后面 ：

ls -ltr s*
将 /bin 目录以下所有目录及文件详细资料列出 ：

ls -lR /bin
列出目前工作目录下所有文件及目录；目录于名称后加 "/"，可执行档于名称后加 "*" ：

ls -AF
```

ll

功能

```
ll并不是linux下一个基本的命令，它实际上是ls -l的一个别名。
Ubuntu默认不支持命令ll，必须用 ls -l，这样使用起来不是很方便。
如果要使用此命令，可以作如下修改：
打开 ~/.bashrc
找到 #alias ll='ls -l'，去掉前面的#就可以了。
```

cp

功能

主要用于复制文件或目录

输入

```
cp [options] source dest


-a：此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于dpR参数组合。
-d：复制时保留链接。这里所说的链接相当于Windows系统中的快捷方式。
-f：覆盖已经存在的目标文件而不给出提示。
-i：与-f选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答"y"时目标文件将被覆盖。
-p：除复制文件的内容外，还把修改时间和访问权限也复制到新文件中。
-r：若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件。
-l：不复制文件，只是生成链接文件。
```

实例

使用指令"cp"将当前目录"test/"下的所有文件复制到新目录"newtest"下，输入如下命令：$ cp –r test/ newtest

rm

功能

rm命令用于删除一个文件或者目录

输入

```
rm [options] name...
参数：

-i 删除前逐一询问确认。
-f 即使原档案属性设为唯读，亦直接删除，无需逐一确认。
-r 将目录及以下之档案亦逐一删除。
```

实例

```
删除文件可以直接使用rm命令，若删除目录则必须配合选项"-r"，例如：

# rm  test.txt
rm：是否删除 一般文件 "test.txt"？ y
# rm  homework
rm：无法删除目录"homework"：是一个目录
# rm  -r  homework
rm：是否删除 目录 "homework"？ y
删除当前目录下的所有文件及目录，命令行为：

rm  -r  *
文件一旦通过rm命令删除，则无法恢复，所以必须格外小心地使用该命令。
```

mv

功能

用来为文件或目录改名、或将文件或目录移入其它位置

输入

实例

```
将文件 aaa 更名为 bbb ：

mv aaa bbb
将info目录放入logs目录中。注意，如果logs目录不存在，则该命令将info改名为logs。

mv info/ logs
再如将/usr/student下的所有文件和目录移到当前目录下，命令行为：

$ mv /usr/student/*  .
```

touch

功能

touch命令用于修改文件或者目录的时间属性，包括存取时间和更改时间。若文件不存在，系统会建立一个新的文件。

输入

```
touch [-acfm][-d<日期时间>][-r<参考文件或目录>] [-t<日期时间>][--help][--version][文件或目录…
]
参数说明：
a 改变档案的读取时间记录。
m 改变档案的修改时间记录。
c 假如目的档案不存在，不会建立新的档案。与 --no-create 的效果一样。
f 不使用，是为了与其他 unix 系统的相容性而保留。
r 使用参考档的时间记录，与 --file 的效果一样。
d 设定时间与日期，可以使用各种不同的格式。
t 设定档案的时间记录，格式与 date 指令相同。
--no-create 不会建立新档案。
--help 列出指令格式。
--version 列出版本讯息。
```

实例

```
使用指令"touch"修改文件"testfile"的时间属性为当前系统时间，输入如下命令：

$ touch testfile                #修改文件的时间属性
首先，使用ls命令查看testfile文件的属性，如下所示：

$ ls -l testfile                #查看文件的时间属性
#原来文件的修改时间为16:09
-rw-r--r-- 1 hdd hdd 55 2011-08-22 16:09 testfile
执行指令"touch"修改文件属性以后，并再次查看该文件的时间属性，如下所示：

$ touch testfile                #修改文件时间属性为当前系统时间
$ ls -l testfile                #查看文件的时间属性
#修改后文件的时间属性为当前系统时间
-rw-r--r-- 1 hdd hdd 55 2011-08-22 19:53 testfile
使用指令"touch"时，如果指定的文件不存在，则将创建一个新的空白文件。例如，在当前目录下，使用该指令创建一个空白文件"file"，输入如下命令：

$ touch file            #创建一个名为“file”的新的空白文件
```

file

功能

file命令用于辨识文件类型。

输入

```
file [-bcLvz][-f <名称文件>][-m <魔法数字文件>...][文件或目录...]
参数：

-b  列出辨识结果时，不显示文件名称。
-c  详细显示指令执行过程，便于排错或分析程序执行的情形。
-f<名称文件>  指定名称文件，其内容有一个或多个文件名称时，让file依序辨识这些文件，格式为每列一个文件名称。
-L  直接显示符号连接所指向的文件的类别。
-m<魔法数字文件>  指定魔法数字文件。
-v  显示版本信息。
-z  尝试去解读压缩文件的内容。
[文件或目录...] 要确定类型的文件列表，多个文件之间使用空格分开，可以使用shell通配符匹配多个文件。
```

实例

```
显示文件类型：

[root@localhost ~]# file install.log
install.log: UTF-8 Unicode text

[root@localhost ~]# file -b install.log      <== 不显示文件名称
UTF-8 Unicode text

[root@localhost ~]# file -i install.log      <== 显示MIME类别。
install.log: text/plain; charset=utf-8

[root@localhost ~]# file -b -i install.log
text/plain; charset=utf-8
```

which

功能

which指令会在环境变量$PATH设置的目录里查找符合条件的文件。并显示它所在的绝对路径

输入

which [文件...]

实例

```
使用指令"which"查看指令"bash"的绝对路径，输入如下命令：

$ which bash
上面的指令执行后，输出信息如下所示：

/bin/bash                    #bash可执行程序的绝对路径
```

type

功能

查找是否是系统所属的命令

实例

```
kylechen@kyle:~$ type ls
ls 是 `ls --color=auto' 的别名


kylechen@kyle:~$ type if
if 是 shell 关键字


kylechen@kyle:~$ type cd
cd 是 shell 内建
```

whereis

功能

该指令只能用于查找二进制文件、源代码文件和man手册页，一般文件的定位需使用locate命令。

输入

```
whereis [-bfmsu][-B <目录>...][-M <目录>...][-S <目录>...][文件...]
参数:

-b  只查找二进制文件。
-B<目录>  只在设置的目录下查找二进制文件。
-f  不显示文件名前的路径名称。
-m  只查找说明文件。
-M<目录>  只在设置的目录下查找说明文件。
-s  只查找原始代码文件。
-S<目录>  只在设置的目录下查找原始代码文件。
-u  查找不包含指定类型的文件。
```

实例

使用指令"whereis"查看指令"bash"的位置，输入如下命令：

```
$ whereis bash
```

上面的指令执行后，输出信息如下所示：

```
bash:/bin/bash/etc/bash.bashrc/usr/share/man/man1/bash.1.gz
```

注意：以上输出信息从左至右分别为查询的程序名、bash路径、bash的man 手册页路径。

如果用户需要单独查询二进制文件或帮助文件，可使用如下命令：

```
$ whereis -b bash
$ whereis -m bash
```

输出信息如下：

```
$ whereis -b bash                #显示bash 命令的二进制程序
bash: /bin/bash /etc/bash.bashrc /usr/share/bash     # bash命令的二进制程序的地址
$ whereis -m bash                #显示bash 命令的帮助文件
bash: /usr/share/man/man1/bash.1.gz   #bash命令的帮助文件地址
```

locate

功能

Linux locate命令用于查找符合条件的文档， 他会去保存文档和目录名称的数据库内，查找合乎范本样式条件的文档或目录。一般情况我们只需要输入 locate your_file_name 即可查找指定文件。

```
附加说明

locate与find 不同: find 是去硬盘找，locate 只在/var/lib/slocate资料库中找。
locate的速度比find快，它并不是真的查找，而是查数据库，一般文件数据库在/var/lib/slocate/slocate.db中，所以locate的查找并不是实时的
```

locate -u

输入

```
locate [-d ][--help][--version][范本样式...]
参数：

-d或--database= 配置locate指令使用的数据库。
locate指令预设的数据库位于/var/lib/slocate目录里，文档名为slocate.db，您可使用 这个参数另行指定。
--help  在线帮助。
--version  显示版本信息。
```

实例

```
查找passwd文件，输入以下命令：
locate passwd
```

find

功能

查找文件

实例

```
find   -name april*                    在当前目录下查找以april开始的文件
find   -name   april*   fprint file        在当前目录下查找以april开始的文件，并把结果输出到file中
find   -name ap* -o -name may*   查找以ap或may开头的文件
find   /mnt   -name tom.txt   -ftype vfat   在/mnt下查找名称为tom.txt且文件系统类型为vfat的文件
find   /mnt   -name t.txt ! -ftype vfat   在/mnt下查找名称为tom.txt且文件系统类型不为vfat的文件
find   /tmp   -name wa* -type l            在/tmp下查找名为wa开头且类型为符号链接的文件
find   /home   -mtime   -2                 在/home下查最近两天内改动过的文件
find /home    -atime -1                  查1天之内被存取过的文件
find /home -mmin    +60                  在/home下查60分钟前改动过的文件
find /home   -amin   +30                   查最近30分钟前被存取过的文件
find /home   -newer   tmp.txt              在/home下查更新时间比tmp.txt近的文件或目录
find /home   -anewer   tmp.txt             在/home下查存取时间比tmp.txt近的文件或目录
find   /home   -used   -2                   列出文件或目录被改动过之后，在2日内被存取过的文件或目录
find   /home   -user cnscn                 列出/home目录内属于用户cnscn的文件或目录
find   /home   -uid   +501                   列出/home目录内用户的识别码大于501的文件或目录
find   /home   -group   cnscn               列出/home内组为cnscn的文件或目录
find   /home   -gid 501                    列出/home内组id为501的文件或目录
find   /home   -nouser                     列出/home内不属于本地用户的文件或目录
find   /home   -nogroup                    列出/home内不属于本地组的文件或目录
find   /home    -name tmp.txt    -maxdepth   4   列出/home内的tmp.txt 查时深度最多为3层
find   /home   -name tmp.txt   -mindepth   3   从第2层开始查
find   /home   -empty                      查找大小为0的文件或空目录
find   /home   -size   +512k                 查大于512k的文件
find   /home   -size   -512k                 查小于512k的文件
find   /home   -links   +2                 查硬连接数大于2的文件或目录
find   /home   -perm   0700                  查权限为700的文件或目录
find   /tmp   -name tmp.txt   -exec cat {} \;
find   /tmp   -name   tmp.txt   -ok   rm {} \;

find   /   -amin   -10    # 查找在系统中最后10分钟访问的文件
find   /   -atime   -2      # 查找在系统中最后48小时访问的文件
find   /   -empty           # 查找在系统中为空的文件或者文件夹
find   /   -group   cat       # 查找在系统中属于 groupcat的文件
find   /   -mmin   -5       # 查找在系统中最后5分钟里修改过的文件
find   /   -mtime   -1     #查找在系统中最后24小时里修改过的文件
find   /   -nouser         #查找在系统中属于作废用户的文件
find   /   -user   fred    #查找在系统中属于FRED这个用户的文件
```

文件内容操作

cat

功能

由第一行开始显示文件内容 用于连接文件并打印到标准输出设备上

输入

```
cat [-AbeEnstTuv] [--help] [--version] fileName
参数说明：
-n 或 --number：由 1 开始对所有输出的行数编号。
-b 或 --number-nonblank：和 -n 相似，只不过对于空白行不编号。
-s 或 --squeeze-blank：当遇到有连续两行以上的空白行，就代换为一行的空白行。
-v 或 --show-nonprinting：使用 ^ 和 M- 符号，除了 LFD 和 TAB 之外。
-E 或 --show-ends ： 在每行结束处显示 $。
-T 或 --show-tabs: 将 TAB 字符显示为 ^I。
-A, --show-all：等价于 -vET。
-e：等价于"-vE"选项；
-t：等价于"-vT"选项；
```

实例

```
把 textfile1 的文档内容加上行号后输入 textfile2 这个文档里：

cat -n textfile1 > textfile2
把 textfile1 和 textfile2 的文档内容加上行号（空白行不加）之后将内容附加到 textfile3 文档里：

cat -b textfile1 textfile2 >> textfile3
清空 /etc/test.txt 文档内容：

cat /dev/null > /etc/test.txt
cat 也可以用来制作镜像文件。例如要制作软盘的镜像文件，将软盘放好后输入：

cat /dev/fd0 > OUTFILE
相反的，如果想把 image file 写到软盘，输入：

cat IMG_FILE > /dev/fd0
```

tac

功能

从最后一行开始显示， 可以看出 tac 是 cat 的倒着写！将文件全部内容从尾到头反向连续输出到标准输出(屏幕)上

nl

功能

显示的时候，顺道输出行号 其默认的结果与 cat -n 有点不太一样， nl 可以将行号做比较多的显示设计，包括位数与是否自动补齐 0 等等的功能。

输入

1. 命令格式：

```
nl [选项][文件]
```

2. 命令参数：

```
-b  ：指定行号指定的方式，主要有两种：
-b a ：表示不论是否为空行，也同样列出行号(类似 cat -n)；
-b t ：如果有空行，空的那一行不要列出行号(默认值)；
-n  ：列出行号表示的方法，主要有三种：
-n ln ：行号在萤幕的最左方显示；
-n rn ：行号在自己栏位的最右方显示，且不加 0 ；
-n rz ：行号在自己栏位的最右方显示，且加 0 ；
-w  ：行号栏位的占用的位数。
-p 在逻辑定界符处不重新开始计算。
```

实例

实例一：用 nl 列出 log2012.log 的内容

命令：

```
nl log2012.log
```

输出：

```
[root@localhost test]# nl log2012.log

     1  2012-01

     2  2012-02




     3  ======[root@localhost test]#
```

说明：

文件中的空白行，nl 不会加上行号

实例二：用 nl 列出 log2012.log 的内容，空本行也加上行号

命令：

```
nl -b a log2012.log
```

输出：

```
[root@localhost test]# nl -b a log2012.log

     1  2012-01

     2  2012-02

     3

     4

     5  ======[root@localhost test]#
```

实例3：让行号前面自动补上0,统一输出格式

命令：

输出：

```
[root@localhost test]# nl -b a -n rz log2014.log

000001  2014-01

000002  2014-02

000003  2014-03

000004  2014-04

000005  2014-05

000006  2014-06

000007  2014-07

000008  2014-08

000009  2014-09

000010  2014-10

000011  2014-11

000012  2014-12

000013  =======

[root@localhost test]# nl -b a -n rz -w 3 log2014.log

001     2014-01

002     2014-02

003     2014-03

004     2014-04

005     2014-05

006     2014-06

007     2014-07

008     2014-08

009     2014-09

010     2014-10

011     2014-11

012     2014-12

013     =======
```

more

功能

一页一页的显示文件内容

输入

```
more [filename]
```

常用操作命令 Enter 向下n行，需要定义。默认为1行 Ctrl+F 向下滚动一屏 空格键 向下滚动一屏 Ctrl+B 返回上一屏 = 输出当前行的行号： f 输出文件名和当前行的行号 V 调用vi编辑器 !命令 调用Shell，并执行命令 q 退出more

less

功能

less 与 more 类似， 但是比 more 更好的是，他可以往前翻页！

实例

```
1、查看文件

less log2013.log
2、ps查看进程信息并通过less分页显示

ps -ef |less
3、查看命令历史使用记录并通过less分页显示

[root@localhost test]# history | less
22  scp -r tomcat6.0.32 root@192.168.120.203:/opt/soft
23  cd ..
24  scp -r web root@192.168.120.203:/opt/
25  cd soft
26  ls
.....省略.....
4、浏览多个文件

less log2013.log log2014.log
说明：
输入 ： n后，切换到 log2014.log
输入 ： p 后，切换到log2013.log
```

1.全屏导航

ctrl + F - 向前移动一屏 ctrl + B - 向后移动一屏 ctrl + D - 向前移动半屏 ctrl + U - 向后移动半屏 2.单行导航

j - 向前移动一行 k - 向后移动一行 3.其它导航

G - 移动到最后一行 g - 移动到第一行 q / ZZ - 退出 less 命令

head

功能

只看头几行

输入

```
1、用法

head [选项] [文件..]

2、命令选项

-c, --bytes=[-]K    k,显示文档开始的前k个字节，-k,不显示文档结尾的最后 k 个字节
-n, --lines=[-]K     k,显示文档开始的前k行，-k,不显示文档结尾的最后 k 行
```

实例

```
1）显示  a.txt  前 5 行内容

复制代码
[root@mini ~]# cat a.txt
01
02
03
04
05
06
07
08
09
10
11
12
[root@mini ~]# head -5 a.txt
01
02
03
04
05
[root@mini ~]# head -n 5 a.txt
01
02
03
04
05
复制代码


2）显示除了  a.txt  最后 10 行的内容

[root@mini ~]# head -n -10 a.txt
01
02
[root@mini ~]#
```

tail

功能

只看尾巴几行

输入

```
tail [参数] [文件]
参数:

-f 循环读取
-q 不显示处理信息
-v 显示详细的处理信息
-c<数目> 显示的字节数
-n<行数> 显示文件的尾部 n 行内容
--pid=PID 与-f合用,表示在进程ID,PID死掉之后结束
-q, --quiet, --silent 从不输出给出文件名的首部
-s, --sleep-interval=S 与-f合用,表示在每次反复的间隔休眠S秒
```

```
有一个常用的参数 -f 常用于查阅正在改变的日志文件。
tail -f filename 会把 filename 文件里的最尾部的内容显示在屏幕上，
并且不断刷新，只要 filename 更新就可以看到最新的文件内容。
```

实例

```
要显示 notes.log 文件的最后 10 行，请输入以下命令:

tail notes.log
要跟踪名为 notes.log 的文件的增长情况，请输入以下命令:

tail -f notes.log
此命令显示 notes.log 文件的最后 10 行。当将某些行添加至 notes.log 文件时，tail 命令会继续显示这些行。 显示一直继续，直到您按下（Ct

显示文件 notes.log 的内容，从第 20 行至文件末尾:

tail +20 notes.log
显示文件 notes.log 的最后 10 个字符:

tail -c 10 notes.log
```

tar

功能

linux下的压缩文件类型如上 现在 compress zip指令几乎不用了 一般用的是gzip bzip2 xz的压缩指令 但是用这三个指定每次只能压缩/解压缩一个文件 故引入了tar指令 先将要压缩的多个文件打包 然后再压缩 解压缩同理 tar指令提供了很多的参数 可以同时提供打包和压缩功能 (解打包和解压缩)

输入

awk

功能

数据处理 相较于 sed 常常作用于一整个行的处理， awk 则比较倾向于一行当中分成数个『字段』来处理。因此，awk

相当的适合处理小型的数据数据处理

实例

https://www.cnblogs.com/virgosnail/p/11103651.html https://www.cnblogs.com/slqdba/p/10470270.html

sed

功能

数据处理

输入

在这里插入图片描述

实例

diff

功能

diff 就是用在比对两个文件之间的差异的， 并且是以行为单位来比对的！一般是用在 ASCII 纯文本 档的比对上。由于是以行为比对的单位， 因此 diff 通常是用在同一的文件(或软件)的新旧版本差异上！

输入

在这里插入图片描述

实例

实例一:比较两文件的不同
实例二:比较两目录的不同

vim

功能

vim编辑器的常用指令

实例

20分钟学会Vim

性能检测相关命令

image

mpstat

功能:

显示CPU的状态信息
这些信息存放在/proc/stat文件中。

在多CPUs系统里，

其不但能查看所有CPU的平均状况信息，

而且能够查看特定CPU的信息。

输入语法:

mpstat(选项)(参数)

选项

-A ： 此选项等效于# mpstat -I ALL -u -P ALL

-I {SUM | CPU | ALL} ： 报告中断统计信息。 使用SUM关键字，mpstat命令报告每个处理器的中断总数。使用CPU关键字，显示CPU或CPU每秒接收

-P {cpu [, ...] | ON | ALL} ： 指示要报告统计信息的处理器编号。cpu是处理器号。注意，处理器0是第一个处理器。ON关键字表示将为每个在线

参数

间隔时间：每次报告的间隔时间（秒）；
次数：显示报告的次数。

输出信息:

```
user 在internal时间段里，用户态的CPU时间（%），不包含 nice值为负 进程 (usr/total)*100
nice 在internal时间段里，nice值为负进程的CPU时间（%）    (nice/total)*100
system 在internal时间段里，内核态的CPU时间（%）    (system/total)*100
iowait 在internal时间段里，硬盘IO等待时间（%） (iowait/total)*100
irq 在internal时间段里，硬中断时间（%）       (irq/total)*100
soft 在internal时间段里，软中断时间（%）     (softirq/total)*100
idle 在internal时间段里，CPU除去等待磁盘IO操作外的因为任何原因而空闲的时间闲置时间（%）(idle/total)*100
```

实例:

mpstat

显示开机到现在以来cpu的平均状态信息

```
Linux 4.15.0-88-generic (kyle)  2020年02月25日  _x86_64_ (4 CPU)

20时12分09秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
20时12分09秒  all   27.16    0.55    9.85    0.34    0.00    0.43    0.00    0.00    0.00   61.66
```

mpstat -P ALL 2 3

每隔2秒显示一次 所有cpu的状态信息 一共产生3个间隔的信息 最后给出这3次间隔的平均信息

```
kylechen@kyle:~$ mpstat -P ALL 2 3
Linux 4.15.0-88-generic (kyle)  2020年02月25日  _x86_64_ (4 CPU)

20时12分43秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
20时12分45秒  all    9.96    0.00    3.57    0.00    0.00    0.12    0.00    0.00    0.00   86.35
20时12分45秒    0    8.96    0.00    2.99    0.00    0.00    0.00    0.00    0.00    0.00   88.06
20时12分45秒    1    8.87    0.00    3.45    0.49    0.00    0.49    0.00    0.00    0.00   86.70
20时12分45秒    2    9.76    0.00    2.44    0.00    0.00    0.00    0.00    0.00    0.00   87.80
20时12分45秒    3   12.32    0.00    4.43    0.00    0.00    0.00    0.00    0.00    0.00   83.25

20时12分45秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
20时12分47秒  all   13.43    0.00    4.68    0.00    0.00    0.12    0.00    0.00    0.00   81.77
20时12分47秒    0   16.59    0.00    5.69    0.00    0.00    0.47    0.00    0.00    0.00   77.25
20时12分47秒    1   12.08    0.00    3.86    0.00    0.00    0.00    0.00    0.00    0.00   84.06
20时12分47秒    2   12.80    0.00    4.74    0.00    0.00    0.00    0.00    0.00    0.00   82.46
20时12分47秒    3   12.44    0.00    5.26    0.00    0.00    0.00    0.00    0.00    0.00   82.30

20时12分47秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
20时12分49秒  all    8.44    0.00    2.02    0.00    0.00    0.13    0.00    0.00    0.00   89.42
20时12分49秒    0    7.69    0.00    1.54    0.00    0.00    0.51    0.00    0.00    0.00   90.26
20时12分49秒    1    6.47    0.00    2.99    0.00    0.00    0.50    0.00    0.00    0.00   90.05
20时12分49秒    2   10.66    0.00    1.02    0.00    0.00    0.00    0.00    0.00    0.00   88.32
20时12分49秒    3    8.46    0.00    2.49    0.00    0.00    0.00    0.00    0.00    0.00   89.05

平均时间:  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
平均时间:  all   10.65    0.00    3.44    0.00    0.00    0.12    0.00    0.00    0.00   85.78
平均时间:    0   11.20    0.00    3.46    0.00    0.00    0.33    0.00    0.00    0.00   85.01
平均时间:    1    9.17    0.00    3.44    0.16    0.00    0.33    0.00    0.00    0.00   86.91
平均时间:    2   11.09    0.00    2.77    0.00    0.00    0.00    0.00    0.00    0.00   86.13
平均时间:    3   11.09    0.00    4.08    0.00    0.00    0.00    0.00    0.00    0.00   84.83
```

mpstat -P ALL -I SUM

查看cpu中断的统计(各个cpu分开显示)

```
inux 4.15.0-88-generic (kyle)  2020年02月25日  _x86_64_ (4 CPU)

20时13分24秒  CPU    intr/s
20时13分24秒  all   2057.27
20时13分24秒    0    473.14
20时13分24秒    1    591.17
20时13分24秒    2    687.38
20时13分24秒    3    493.70
```

mpstat -I SUM

查看cpu中断的统计(各个cpu合并显示)

```
kylechen@kyle:~$  mpstat -I SUM
Linux 4.15.0-88-generic (kyle)  2020年02月25日  _x86_64_ (4 CPU)

20时13分47秒  CPU    intr/s
20时13分47秒  all   2036.32
```

free

功能

free指令会显示内存的使用情况， 包括实体内存， 虚拟的交换文件内存， 共享内存区段， 以及系统核心使用的缓冲区等。

输入语法

```
free [-bkmotV][-s <间隔秒数>]
参数说明：

-b  以Byte为单位显示内存使用情况。
-k  以KB为单位显示内存使用情况。
-m  以MB为单位显示内存使用情况。
-h  以合适的单位显示内存使用情况，最大为三位数，自动计算对应的单位值。单位如下：

B = bytes
K = kilos
M = megas
G = gigas
T = teras
-t  显示内存总和列。
-V  显示版本信息。
-o  不显示缓冲区调节列。
-s<间隔秒数>持续观察内存使用状况。
```

输出信息

```
total 列显示系统总的可用物理内存和交换空间大小。
used 列显示已经被使用的物理内存和交换空间。
free 列显示还有多少物理内存和交换空间可用使用。
shared 列显示被共享使用的物理内存大小。(进程间的共享内存)
buff/cache 列显示被    磁盘缓存 使用的物理内存大小。(buffer表示写缓冲    cache表示读缓冲)
available 列显示还可以被应用程序使用的物理内存大小
```

注意点
1.当应用程序需要内存时，如果没有足够的 free 内存可以用
，内核就会从 buffer 和 cache 中回收内存来满足应用程序的请
求。所以从应用程序的角度来说，available = free + buffer + cache。
请注意，这只是一个很理想的计算方式，实际中的数据往往有较大的误差。

2.swap space 是磁盘上的一块区域，可以是一个分区，也可以是一个文件
。所以具体的实现可以是 swap 分区也可以是 swap 文件
。当系统物理内存吃紧时，Linux 会将内存中不常访问的数据保存到 swap 上，
这样系统就有更多的物理内存为各个进程服务，而当系统需要访问 swap 上存储的内容时
，再将 swap 上的数据加载到内存中，这就是常说的换出和换入。
交换空间可以在一定程度上缓解内存不足的情况，
但是它需要读写磁盘数据，所以性能不是很高。

实例

free 不指定单位显示内存使用情况

```
              总计         已用         空闲       共享     缓冲/缓存     可用
内存:       3920116      2680416       179976       596300      1059724       408108
交换:        999420       519916       479504
```

free -h以合适的单位显示内存使用情况

```
kylechen@kyle:~$ free -h
              总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.6G         178M         573M         1.0G         401M
交换:          975M         507M         468M
```

free -th以合适的单位显示内存使用情 况,同时显示总量列

```
kylechen@kyle:~$ free -th
              总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         200M         570M         1.0G         424M
交换:          975M         507M         468M
总量:          4.7G         3.0G         668M
```

```
free -h -s 1  以1秒为间隔显示内存使用情况

kylechen@kyle:~$ free -h -s 1
               总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         198M         569M         1.0G         421M
交换:          975M         507M         468M

               总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         196M         571M         1.0G         419M
交换:          975M         507M         468M

               总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         195M         571M         1.0G         419M
交换:          975M         507M         468M

               总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         196M         571M         1.0G         420M
交换:          975M         507M         468M

               总计         已用         空闲       共享     缓冲/缓存     可用
内存:          3.7G         2.5G         199M         568M         1.0G         423M
交换:          975M         507M         468M
```

top

功能

```
top是系统管理员最重要的工具之一。被广泛用于监视服务器的负载。
它可以动态的查看系统当前正在运行的进程情况 ， 内存使用情况， cpu使用情况
也就是说上面mpstate 和free能实现的功能 它都内在的包含了
```

输入语法

很简单 直接输入top

输出信息

```
任务: 298 total,   1 running, 246 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.7 us,  1.6 sy,  0.0 ni, 92.6 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3920116 total,   301948 free,  2514912 used,  1103256 buff/cache
KiB Swap:   999420 total,   480272 free,   519148 used.   618380 avail Mem

进程id USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1590 kylechen  20   0 3974220 265424  72628 S   7.3  6.8   5:33.19 gnome-shell
 1446 kylechen  20   0  521960  38412  19292 S   6.3  1.0   3:28.27 Xorg
 3008 kylechen  20   0  780520  50972  32452 S   4.6  1.3   0:17.32 gnome-term+
 2779 kylechen  20   0 1201196 318344  79024 S   1.3  8.1  11:48.81 chrome
 1955 kylechen  20   0 1505112 399200 243564 S   1.0 10.2   5:32.38 chrome
    1 root      20   0  225908   4064   1584 S   0.7  0.1   0:08.15 systemd
 1036 gdm       20   0 3492528  89676  62056 S   0.7  2.3   0:11.70 gnome-shell
 1993 kylechen  20   0  556488  55780  24440 S   0.7  1.4   0:34.27 chrome
    8 root      20   0       0      0      0 I   0.3  0.0   0:08.76 rcu_sched
  983 mysql     20   0 1358648      0      0 S   0.3  0.0   0:03.85 mysqld
 1571 kylechen  20   0  220792    156      0 S   0.3  0.0   0:03.21 at-spi2-re+
 2192 kylechen  20   0  486420   5828      0 S   0.3  0.1   0:01.88 sogou-qimp+
 5330 kylechen  20   0 2621992 450080  80948 S   0.3 11.5   1:13.24 XMind
 5708 root      20   0       0      0      0 I   0.3  0.0   0:01.65 kworker/u8+
 6974 kylechen  20   0  880400 185648 107616 S   0.3  4.7   0:45.80 chrome
 7245 root      20   0       0      0      0 I   0.3  0.0   0:00.22 kworker/2:0
 7760 kylechen  20   0   51360   4004   3332 R   0.3  0.1   0:00.19 top
```

输出信息如上所示 一大坨 很繁琐 咱们一行一行来分析

第一行表示 当前的进程状态 总共有298个进程 1个处理runing 246个处于sleeping 0个处于stopped状态

```
任务: 298 total,   1 running, 246 sleeping,   0 stopped
```

第二行表示 当前的cpu状态 这里显示的状态参数其实和mpstat那里说的是一样的 具体可以看上面的mpstat命令讲解

```
%Cpu(s):  5.7 us,  1.6 sy,  0.0 ni, 92.6 id,  0.1 wa,  0.0 hi,
```

**第三四行表示 **当前的内存状态 这里显示的状态参数其实和free那里说的是一样的 具体可以看上面的free命令讲解

```
KiB Mem :  3920116 total,   301948 free,  2514912 used,  1103256 buff/cache
KiB Swap:   999420 total,   480272 free,   519148 used.   6183800.1 wa,  0.0 hi,
```

后面几行其实 表示的就是每个进程具体占用的系统资源 会随着时间动态变化

PID

进程ID，进程的唯一标识符

USER

进程所有者的实际用户名。

PR

进程的调度优先级。这个字段的一些值是'rt'。这意味这这些进程运行在实时态。

NI

进程的nice值（优先级）。越小的值意味着越高的优先级。

VIRT

进程使用的虚拟内存。

RES

驻留内存大小。驻留内存是任务使用的非交换物理内存大小。

SHR

SHR是进程使用的共享内存。

S

这个是进程的状态。它有以下不同的值:

D – 不可中断的睡眠态。

R – 运行态

S – 睡眠态

T – 被跟踪或已停止

Z – 僵尸态

I - 空闲状态（idle）

%CPU

自从上一次更新时到现在任务所使用的CPU时间百分比。

%MEM

进程使用的可用物理内存百分比。

TIME+

任务启动后到现在所使用的全部CPU时间，精确到百分之一秒。

COMMAND

运行进程所使用的命令。

还有许多在默认情况下不会显示的输出，它们可以显示进程的页错误、有效组和组ID和其他更多的信息。

```
进程id USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1590 kylechen  20   0 3974220 265424  72628 S   7.3  6.8   5:33.19 gnome-shell
 1446 kylechen  20   0  521960  38412  19292 S   6.3  1.0   3:28.27 Xorg
 3008 kylechen  20   0  780520  50972  32452 S   4.6  1.3   0:17.32 gnome-term+
 2779 kylechen  20   0 1201196 318344  79024 S   1.3  8.1  11:48.81 chrome
 .......
```

实例

```
实例1:top

top - 21:25:55 up  1:27,  1 user,  load average: 1.51, 1.87, 1.49
任务: 305 total,   2 running, 250 sleeping,   0 stopped,   0 zombie
%Cpu(s): 22.7 us,  1.8 sy,  0.0 ni, 75.1 id,  0.1 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem :  3920116 total,   121880 free,  2859016 used,   939220 buff/cache
KiB Swap:   999420 total,   333328 free,   666092 used.   182088 avail Mem

PID  USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2779 kylechen  20   0 1191.7m 314.5m  59.2m S 17.1  8.2  20:45.14 chrome
 1590 kylechen  20   0 3881.1m 242.1m  70.3m R  4.8  6.3   7:19.67 gnome-shell
 1446 kylechen  20   0  444.6m  31.1m  11.8m S  1.3  0.8   4:42.48 Xorg
 8711 kylechen  20   0  622.0m  17.4m   5.6m S  0.7  0.5   0:02.43 gnome-termi+
 1955 kylechen  20   0 1458.6m 348.2m 189.5m S  0.5  9.1   7:31.10 chrome
 1989 kylechen  20   0  653.0m 112.1m  47.2m S  0.5  2.9   7:06.33 chrome
 8736 kylechen  20   0   50.2m   1.5m   0.8m R  0.2  0.0   0:05.51 top
 1036 gdm       20   0 3410.7m  82.2m  55.3m S  0.1  2.1   0:15.10 gnome-shell
 1484 kylechen  20   0  442.8m  48.0m   8.9m S  0.1  1.3   0:40.19 fcitx
 1993 kylechen  20   0  551.4m  48.9m  17.8m S  0.1  1.3   0:45.63 chrome
 3299 kylechen  20   0  633.0m  20.9m   5.1m S  0.1  0.5   0:17.90 chrome
    1 root      20   0  220.6m   3.0m   0.6m S  0.0  0.1   0:10.02 systemd
    2 root      20   0    0.0m   0.0m   0.0m S  0.0  0.0   0:00.00 kthreadd
    4 root       0 -20    0.0m   0.0m   0.0m I  0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20    0.0m   0.0m   0.0m I  0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0    0.0m   0.0m   0.0m S  0.0  0.0   0:00.22 ksoftirqd/0
    8 root      20   0    0.0m   0.0m   0.0m I  0.0  0.0   0:11.86 rcu_sched
```

```
实例2：交互式命令

top是一个交互式的命令
所以用top调出动态显示的进程状态以后
在界面上继续用键盘输入指令
会继续在界面上执行对应的操作


交互命令1：  回车/空格

top命令默认在一个特定间隔(3秒)后刷新显示。
要手动刷新，用户可以输入回车或者空格。

交互命令2：   B
一些重要信息会以加粗字体显示。
这个命令可以切换粗体显示。


交互命令3：  d 或s
当按下’d’或’s’时，你
将被提示输入一个值（以秒为单位），
它会以设置的值作为刷新间隔。
如果你这里输入了1，top将会每秒刷新。

交互命令4 :‘R’
切换反向/常规排序。

交互命令5 :‘V’
切换树视图。

交互命令6：  ‘k’
top命令中最重要的一个命令之一。
用于发送信号给任务（通常是结束任务）。


交互命令7：  ‘e’
切换显示的单位
依次以k ->m->g->t->p单位选择
```

进程相关命令

在这里插入图片描述

ps

功能

ps 用来查看进程状态 不过这种查看是静态的 也就是只会显示 你输入命令那一刻的进程状态 不会像top那样是动态变化的 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 这几种风格可以混用，但是可能会发生冲突。

输入语法

ps[参数]

```
参数如下
ps 没有属性参数的时候显示的是同一终端terminal下所有的进程
ps T  显示同一个终端terminal下的所有进程 输出信息更丰富了一些
ps a 显示同一控制终端tty下的所有进程  结果按照进程id来排序 输出的关键属性有 进程状态  进程控制终端
ps c  显示进程的名称 不显示路径
ps -A 显示所有用户的所有进程  包括没有控制终端的进程 结果按照进程id排序 输出的关键属性有进程的控制终端 和ps -aux 相比 它输出的信息没
ps -e 等于“ps -A”
ps f  显示同一个控制终端tty下的进程  同时用树状结构的显示程序间的关系
ps -a  显示所有用户进程 不包括没有控制终端的进程 结果按照进程id排序
ps -u  显示本用户下所有进程 不包括没有控制终端的进程  结果按照进程id排序 而且显示的进程信息很全面
ps  -u [用户名] 显示指定用户名下的所有进程      不包括没有控制终端的进程  结果按照进程id排序 而且显示的进程信息很全面 比如 ps -u roo
ps -x  显示本用户所有进程 包括没有控制终端的进程 结果按照进程id排序

ps -au 显示当前用户下所有的进程  不包括没有控制终端的进程
ps -ax 显示所有用户的进程 包括没有控制终端的进程 输出信息里面没有USER用户列
ps -ux 显示当前用户下所有进程      包括没有控制终端的进程
ps -aux  显示所有用户的所有进程 包括没有控制终端的进程
ps -aux --sort -pcpu  显示所有的进程 并且按照cpu使用率排序
ps -aux --sort -pmem  显示所有的进程 并且按照cpu使用率排序
ps -auxf 显示所有用户的所有进程 包括没有控制终端的进程 同时以树形结构显示进程间的关系
ps -e  f  (注意e和f中间有空格)显示所有进程  包括没有控制终端的进程    同时会用树状结构的显示程序间的关系
```

输出信息

```
USER - 运行该过程的用户
PID 就是这个程序的 ID
PPID 则是其上级父程序的ID
%CPU- 进程 cpu 利用率。
%MEM - 进程驻留集大小占计算机物理内存的百分比。
VSZ  - 进程的虚拟内存大小 KiB。
RSS- 进程正在使用的物理内存的大小。
PRI 这个是 Priority (优先执行序) 的缩写
NI 这个是 Nice 值
ADDR 这个是 kernel function, 指出该程序在内存的那个部分。如果是个 running的程序, 一般就是 "-"
TTY 登入者的终端机位置
TIME 使用掉的 CPU 时间。
CMD 所下达的指令为何
STAT 代表这个程序的状态
   ps工具标识进程的5种状态码:
   D 不可中断
   R 运行
   S 中断
   T 停止
   Z 僵死
   在上面这些状态吗后面还会有下面这些后缀
   < 优先级高的进程
   N 优先级较低的进程
   L 有些页被锁进内存;
   s 进程的领导者 (在它之下有子进程) ;
   l 多进程的 (使用 CLONE_THREAD, 类似 NPTL pthreads) ;
   + 位于后台的进程组;
```

实例

```
实例1
kylechen@kyle:~$ ps
  PID TTY          TIME CMD
17796 pts/0    00:00:00 bash
24667 pts/0    00:00:00 ps
```

```
实例2
kylechen@kyle:~$ ps -T
  PID  SPID TTY          TIME CMD
17796 17796 pts/0    00:00:00 bash
24673 24673 pts/0    00:00:00 ps
```

实例3

```
kylechen@kyle:~$ ps a
  PID TTY      STAT   TIME COMMAND
 1444 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SH
 1446 tty2     Sl+   19:57 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1
 1457 tty2     Sl+    0:00 /usr/lib/gnome-session/gnome-session-binary --session
 1590 tty2     Sl+   31:32 /usr/bin/gnome-shell
 1628 tty2     Sl     0:00 ibus-daemon --xim --panel disable
 1632 tty2     Sl     0:00 /usr/lib/ibus/ibus-dconf
 1636 tty2     Sl     0:00 /usr/lib/ibus/ibus-x11 --kill-daemon
 1708 tty2     Sl+    0:01 /usr/lib/gnome-settings-daemon/gsd-power
 1710 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-print-notification
 1711 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-rfkill
 1714 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-screensaver-proxy
 1717 tty2     Sl+    0:03 /usr/lib/gnome-settings-daemon/gsd-sharing
 1720 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-smartcard
后面的篇幅太长 ...略掉
```

实例4

```
kylechen@kyle:~$ ps a
  PID TTY      STAT   TIME COMMAND
 1444 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SH
 1446 tty2     Sl+   19:57 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1
 1457 tty2     Sl+    0:00 /usr/lib/gnome-session/gnome-session-binary --session
 1590 tty2     Sl+   31:32 /usr/bin/gnome-shell
 1628 tty2     Sl     0:00 ibus-daemon --xim --panel disable
 1632 tty2     Sl     0:00 /usr/lib/ibus/ibus-dconf
 1636 tty2     Sl     0:00 /usr/lib/ibus/ibus-x11 --kill-daemon
 1708 tty2     Sl+    0:01 /usr/lib/gnome-settings-daemon/gsd-power
 1710 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-print-notification
 1711 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-rfkill
 1714 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-screensaver-proxy
 1717 tty2     Sl+    0:03 /usr/lib/gnome-settings-daemon/gsd-sharing
 1720 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-smartcard
后面的篇幅太长 ...略掉
```

实例5

```
kylechen@kyle:~$ ps a
  PID TTY      STAT   TIME COMMAND
 1444 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SH
 1446 tty2     Sl+   19:57 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1
 1457 tty2     Sl+    0:00 /usr/lib/gnome-session/gnome-session-binary --session
 1590 tty2     Sl+   31:32 /usr/bin/gnome-shell
 1628 tty2     Sl     0:00 ibus-daemon --xim --panel disable
 1632 tty2     Sl     0:00 /usr/lib/ibus/ibus-dconf
 1636 tty2     Sl     0:00 /usr/lib/ibus/ibus-x11 --kill-daemon
 1708 tty2     Sl+    0:01 /usr/lib/gnome-settings-daemon/gsd-power
 1710 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-print-notification
 1711 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-rfkill
 1714 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-screensaver-proxy
 1717 tty2     Sl+    0:03 /usr/lib/gnome-settings-daemon/gsd-sharing
 1720 tty2     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-smartcard
后面的篇幅太长 ...略掉
```

实例6

```
kylechen@kyle:~$ ps c
  PID TTY      STAT   TIME COMMAND
 1444 tty2     Ssl+   0:00 gdm-x-session
 1446 tty2     Sl+   20:05 Xorg
 1457 tty2     Sl+    0:00 gnome-session-b
 1590 tty2     Sl+   31:46 gnome-shell
 1628 tty2     Sl     0:00 ibus-daemon
 1632 tty2     Sl     0:00 ibus-dconf
 1636 tty2     Sl     0:00 ibus-x11
 1708 tty2     Sl+    0:01 gsd-power
 1710 tty2     Sl+    0:00 gsd-print-notif
 1711 tty2     Sl+    0:00 gsd-rfkill
 1714 tty2     Sl+    0:00 gsd-screensaver
 1717 tty2     Sl+    0:03 gsd-sharing
 1720 tty2     Sl+    0:00 gsd-smartcard
 1725 tty2     Sl+    0:00 gsd-xsettings
 1728 tty2     Sl+    0:00 gsd-wacom
 1735 tty2     Sl+    0:00 gsd-sound
 1746 tty2     Sl+    0:00 gsd-a11y-settin
 1747 tty2     Sl+    0:00 gsd-clipboard
 1751 tty2     Sl+    0:04 gsd-color
 1754 tty2     Sl+    0:00 gsd-datetime
 1755 tty2     Sl+    0:02 gsd-housekeepin
 1756 tty2     Sl+    0:00 gsd-keyboard
 1759 tty2     Sl+    0:01 gsd-media-keys
 1763 tty2     Sl+    0:00 gsd-mouse
 1787 tty2     Sl+    0:00 gsd-printer
 1807 tty2     Sl+    0:00 gsd-disk-utilit
 1955 tty2     SLl+  38:57 chrome
后面的篇幅太长 ...略掉
```

```
kylechen@kyle:~$ ps -aux --sort -pmem
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
kylechen  5330  0.4  9.6 2675660 377160 tty2   Sl+  2月25   4:39 /opt/XMind ZEN/XMind --type=renderer --no-sandbox
kylechen  1955  3.7  9.5 1592304 373052 tty2   SLl+ 2月25  39:07 /opt/google/chrome/chrome
kylechen 14282  4.0  7.9 1240340 312420 tty2   Sl+  2月25  35:06 /opt/google/chrome/chrome --type=renderer --disabl
kylechen 19242  0.4  6.9 1008672 273388 tty2   Sl+  00:25   3:11 /opt/google/chrome/chrome --type=renderer --disab
kylechen 24091  3.2  5.3 916448 209380 tty2    Sl+  12:50   0:38 /opt/google/chrome/chrome --type=renderer --disab
kylechen  1590  3.1  5.3 3998928 208124 tty2   Sl+  2月25  32:04 /usr/bin/gnome-shell
kylechen 23859  0.6  5.3 899524 207904 tty2    Sl+  12:46   0:08 /opt/google/chrome/chrome --type=renderer --disab
kylechen 19254  0.1  4.9 866404 194272 tty2    Sl+  00:25   0:50 /opt/google/chrome/chrome --type=renderer --disab
kylechen  2439  0.0  3.5 1497400 140056 tty2   SLl+ 2月25   0:20 /usr/bin/gnome-software --gapplication-service
kylechen 19681  1.3  2.9 640148 114188 tty2    Sl+  00:32  10:14 /proc/self/exe --type=gpu-process --field-trial-h
kylechen 19231  0.1  2.8 835928 110572 tty2    Sl+  00:25   0:59 /opt/google/chrome/chrome --type=renderer --disab
kylechen  5263  0.2  2.1 2234672 84160 tty2    Sl+  2月25   2:06 /opt/XMind ZEN/XMind
后面篇幅过长 略掉
```

```
kylechen@kyle:~$ ps -e f
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
    4 ?        I<     0:00  \_ [kworker/0:0H]
    6 ?        I<     0:00  \_ [mm_percpu_wq]
    7 ?        S      0:00  \_ [ksoftirqd/0]
    8 ?        I      0:54  \_ [rcu_sched]
    9 ?        I      0:00  \_ [rcu_bh]
   10 ?        S      0:00  \_ [migration/0]
   11 ?        S      0:00  \_ [watchdog/0]
   12 ?        S      0:00  \_ [cpuhp/0]
   13 ?        S      0:00  \_ [cpuhp/1]
   14 ?        S      0:00  \_ [watchdog/1]
   15 ?        S      0:00  \_ [migration/1]
   16 ?        S      0:02  \_ [ksoftirqd/1]
   18 ?        I<     0:00  \_ [kworker/1:0H]
   19 ?        S      0:00  \_ [cpuhp/2]
   20 ?        S      0:00  \_ [watchdog/2]
   21 ?        S      0:00  \_ [migration/2]
   22 ?        S      0:01  \_ [ksoftirqd/2]
   24 ?        I<     0:00  \_ [kworker/2:0H]
   25 ?        S      0:00  \_ [cpuhp/3]
   26 ?        S      0:00  \_ [watchdog/3]
   27 ?        S      0:00  \_ [migration/3]
   28 ?        S      0:00  \_ [ksoftirqd/3]
   30 ?        I<     0:00  \_ [kworker/3:0H]
   31 ?        S      0:00  \_ [kdevtmpfs]
   32 ?        I<     0:00  \_ [netns]
   33 ?        S      0:00  \_ [rcu_tasks_kthre]
   34 ?        S      0:00  \_ [kauditd]
后面略掉
```

pstree

功能

清晰明了的用树形图显示所有进程的层次关系

输入语法

pstree

kill

功能

发送信号到指定进程

输入语法

```
有两种方式
1.kill [命令参数]
2.kill [信号参数] [指定进程]
```

```
kill [命令参数]

-l  后面加信号名称 显示指定信号对应的编号 如果后面不加信号名称 则显示所有信号对应编号
-u 后面加用户名称 表示杀死指定用户的所用进程 即给指定用户的所有进程发送SIGTERM信号
```

```
kill [信号参数] [指定进程]

信号参数--就是你要给指定进程发送的信号值 可以是名称的形式比如-SIGKILL ,-SIGHUP 也可以是编号的形式 比如-9 ,-1 分别对应SIGKILL ,SIG
指定进程--就是你想要给其发送信号的进程pid 可以通过一些手段比如ps/pstree/top等获取


注意点:

1.也可以不指定信号参数 这样默认发送的就是编号为15的SIGTERM信号

2.当用kill向这些进程发送信号时，必须是这些进程的主人。
如果试图给一个没有权限或者不存在的进程发送信号，就会得到一个错误信息。

3.只有第9种信号(SIGKILL)才可以无条件终止进程，其他信号进程都有权利忽略。
 下面是常用的信号：
SIGHUP    1    终端断线
SIGINT     2    中断（同 Ctrl + C）
SIGQUIT    3    退出（同 Ctrl + \）
SIGTERM   15    终止
SIGKILL    9    强制终止
SIGCONT   18    继续（与STOP相反， fg/bg命令）
SIGSTOP    19    暂停（同 Ctrl + Z）

4.init进程是不可杀的
```

实例

```
实例1 ： 显示所有信号及其编号
kylechen@kyle:~$ kill -l
 1) SIGHUP  2) SIGINT  3) SIGQUIT  4) SIGILL  5) SIGTRAP
 6) SIGABRT  7) SIGBUS  8) SIGFPE  9) SIGKILL 10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR
31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

```
实例2 ：  显示SIGHIP的编号
kylechen@kyle:~$ kill -l SIGHUP
1
```

```
实例3: 给用户kylechen的所有进程发送默认信号SIGTERM
kylechen@kyle:~$ kill -u kylechen
```

```
实例4  给进程26467发送默认信号 SIGTERM
kylechen@kyle:~$ kill 26467
```

```
实例5  给进程26467发送编号为9的信号 SIGKILL
kylechen@kyle:~$ kill  -9 26467
```

```
实例5 给进程26467发送名称为SIGHUP的信号
kylechen@kyle:~$ kill  -SIGHUP 26467
```

ulimit

功能

限制用户使用系统的某些资源 包括可以开启的文件数量， 可以使用的 CPU 时间，可以使用的内存总量等等。

输入

配额可以使unlimited 这样就设定为无限制

磁盘相关命令

du

功能

显示指定的目录或文件所占用的磁盘空间。

输入语法

```
du [参数][目录路径或文件路径]

常用参数:
-a 显示指定目录下所有单独的文件大小 后面不加文件或者目录路径则表示本目录
-b 以byte为单位显示
-c 除了显示单独的目录或者文件的大小外，同时也显示所有目录或文件的总和。
-h 以K，M，G为单位显示，提高信息的可读性。
如果不加参数则显示当前目录下总的大小
```

实例

```
kylechen@kyle:~/test$ du
20 .
```

```
ylechen@kyle:~/test$ du -a
16 ./test.jpg
0 ./readme.txt
20 .
```

```
kylechen@kyle:~/test$ du -ab
14491 ./test.jpg
0 ./readme.txt
18587 .
```

```
kylechen@kyle:~/test$ du -abc
14491 ./test.jpg
0 ./readme.txt
18587 .
18587 总用量
```

df

功能

显示目前在Linux系统上的文件系统的磁盘使用情况统计。

输入语法

```
df [参数]
```

```
df  不加参数 显示整个磁盘空间中文件系统的使用情况
df -h 以人类友好格式显示文件系统硬盘空间使用情况 这样就会在显示合适的单位 比如 K M G
df -T 显示文件系统类型
df -t ext4 仅列出某些文件系统 比如这里是限制只列出ext4系统的文件系统
df -x ext4 排除指定类型的文件系统 比如这里是列出了除 ext4 类型以外的全部文件系统
```

输出信息

```
Filesystem – Linux 系统中的文件系统
1K-blocks – 文件系统的大小，用 1K 大小的块来表示。
Used – 以 1K 大小的块所表示的已使用数量。
Available – 以 1K 大小的块所表示的可用空间的数量。
Use% – 文件系统中已使用的百分比。
Mounted on – 已挂载的文件系统的挂载点。
```

实例

```
kylechen@kyle:~/test$ df
文件系统                         1K-块      已用     可用 已用% 挂载点
udev                          1936656         0  1936656    0% /dev
tmpfs                          392016      2068   389948    1% /run
/dev/mapper/ubuntu--vg-root 243559804 181980468 49137532   79% /
tmpfs                         1960060    311900  1648160   16% /dev/shm
tmpfs                            5120         4     5116    1% /run/lock
tmpfs                         1960060         0  1960060    0% /sys/fs/cgroup
/dev/loop0                       1024      1024        0  100% /snap/gnome-logs/81
/dev/loop1                     127360    127360        0  100% /snap/vscode/93
/dev/loop2                       4352      4352        0  100% /snap/gnome-calculator/544
/dev/loop3                     144128    144128        0  100% /snap/gnome-3-26-1604/98
/dev/loop4                       1024      1024        0  100% /snap/gnome-logs/73
/dev/loop5                        256       256        0  100% /snap/gtk2-common-themes/9
/dev/loop6                     126976    126976        0  100% /snap/vscode/89
/dev/loop7                     164096    164096        0  100% /snap/gnome-3-28-1804/116
/dev/loop8                      93568     93568        0  100% /snap/core/8689
/dev/loop9                      56064     56064        0  100% /snap/core18/1668
/dev/loop11                     56064     56064        0  100% /snap/core18/1650
/dev/loop10                      4352      4352        0  100% /snap/gnome-calculator/536
/dev/loop12                     15232     15232        0  100% /snap/remmina/3995
/dev/loop13                    267008    267008        0  100% /snap/kde-frameworks-5-core18/32
/dev/loop14                       256       256        0  100% /snap/gtk2-common-themes/5
/dev/loop15                     15232     15232        0  100% /snap/remmina/3968
/dev/loop16                     15104     15104        0  100% /snap/gnome-characters/399
/dev/sda1                      523248      2284   520964    1% /boot/efi
/dev/loop17                    160512    160512        0  100% /snap/gnome-3-28-1804/110
/dev/loop18                    127872    127872        0  100% /snap/vscode/87
/dev/loop19                     15104     15104        0  100% /snap/gnome-characters/375
/dev/loop20                     98688     98688        0  100% /snap/typora-alanzanattadev/2
/dev/loop21                      3840      3840        0  100% /snap/gnome-system-monitor/123
/dev/loop22                     98176     98176        0  100% /snap/kdenlive/22
/dev/loop23                     98304     98304        0  100% /snap/kdenlive/13
/dev/loop24                     46080     46080        0  100% /snap/gtk-common-themes/1440
/dev/loop25                    144128    144128        0  100% /snap/gnome-3-26-1604/97
/dev/loop26                     45312     45312        0  100% /snap/gtk-common-themes/1353
/dev/loop27                    259584    259584        0  100% /snap/electronic-wechat/7
/dev/loop28                     93568     93568        0  100% /snap/core/8592
/dev/loop29                      3840      3840        0  100% /snap/gnome-system-monitor/127
tmpfs                          392012        16   391996    1% /run/user/121
tmpfs                          392012        36   391976    1% /run/user/1000
```

```
kylechen@kyle:~/test$ df -h
文件系统                      容量  已用  可用 已用% 挂载点
udev                          1.9G     0  1.9G    0% /dev
tmpfs                         383M  2.1M  381M    1% /run
/dev/mapper/ubuntu--vg-root   233G  174G   47G   79% /
tmpfs                         1.9G  307M  1.6G   17% /dev/shm
tmpfs                         5.0M  4.0K  5.0M    1% /run/lock
tmpfs                         1.9G     0  1.9G    0% /sys/fs/cgroup
/dev/loop0                    1.0M  1.0M     0  100% /snap/gnome-logs/81
/dev/loop1                    125M  125M     0  100% /snap/vscode/93
/dev/loop2                    4.3M  4.3M     0  100% /snap/gnome-calculator/544
/dev/loop3                    141M  141M     0  100% /snap/gnome-3-26-1604/98
/dev/loop4                    1.0M  1.0M     0  100% /snap/gnome-logs/73
/dev/loop5                    256K  256K     0  100% /snap/gtk2-common-themes/9
/dev/loop6                    124M  124M     0  100% /snap/vscode/89
/dev/loop7                    161M  161M     0  100% /snap/gnome-3-28-1804/116
/dev/loop8                     92M   92M     0  100% /snap/core/8689
/dev/loop9                     55M   55M     0  100% /snap/core18/1668
/dev/loop11                    55M   55M     0  100% /snap/core18/1650
/dev/loop10                   4.3M  4.3M     0  100% /snap/gnome-calculator/536
/dev/loop12                    15M   15M     0  100% /snap/remmina/3995
/dev/loop13                   261M  261M     0  100% /snap/kde-frameworks-5-core18/32
/dev/loop14                   256K  256K     0  100% /snap/gtk2-common-themes/5
/dev/loop15                    15M   15M     0  100% /snap/remmina/3968
/dev/loop16                    15M   15M     0  100% /snap/gnome-characters/399
/dev/sda1                     511M  2.3M  509M    1% /boot/efi
/dev/loop17                   157M  157M     0  100% /snap/gnome-3-28-1804/110
/dev/loop18                   125M  125M     0  100% /snap/vscode/87
/dev/loop19                    15M   15M     0  100% /snap/gnome-characters/375
/dev/loop20                    97M   97M     0  100% /snap/typora-alanzanattadev/2
/dev/loop21                   3.8M  3.8M     0  100% /snap/gnome-system-monitor/123
/dev/loop22                    96M   96M     0  100% /snap/kdenlive/22
/dev/loop23                    96M   96M     0  100% /snap/kdenlive/13
/dev/loop24                    45M   45M     0  100% /snap/gtk-common-themes/1440
/dev/loop25                   141M  141M     0  100% /snap/gnome-3-26-1604/97
/dev/loop26                    45M   45M     0  100% /snap/gtk-common-themes/1353
/dev/loop27                   254M  254M     0  100% /snap/electronic-wechat/7
/dev/loop28                    92M   92M     0  100% /snap/core/8592
/dev/loop29                   3.8M  3.8M     0  100% /snap/gnome-system-monitor/127
tmpfs                         383M   16K  383M    1% /run/user/121
tmpfs                         383M   36K  383M    1% /run/user/1000
```

```
kylechen@kyle:~/test$ df -T
文件系统                     类型          1K-块      已用      可用 已用% 挂载点
udev                         devtmpfs   1936656         0   1936656    0% /dev
tmpfs                        tmpfs       392016      2068    389948    1% /run
/dev/mapper/ubuntu--vg-root ext4     243559804 181980572 49137428   79% /
tmpfs                        tmpfs      1960060    302084   1657976   16% /dev/shm
tmpfs                        tmpfs         5120         4      5116    1% /run/lock
tmpfs                        tmpfs      1960060         0   1960060    0% /sys/fs/cgroup
/dev/loop0                   squashfs      1024      1024         0  100% /snap/gnome-logs/81
/dev/loop1                   squashfs    127360    127360         0  100% /snap/vscode/93
/dev/loop2                   squashfs      4352      4352         0  100% /snap/gnome-calculator/544
/dev/loop3                   squashfs    144128    144128         0  100% /snap/gnome-3-26-1604/98
/dev/loop4                   squashfs      1024      1024         0  100% /snap/gnome-logs/73
/dev/loop5                   squashfs       256       256         0  100% /snap/gtk2-common-themes/9
/dev/loop6                   squashfs    126976    126976         0  100% /snap/vscode/89
/dev/loop7                   squashfs    164096    164096         0  100% /snap/gnome-3-28-1804/116
/dev/loop8                   squashfs     93568     93568         0  100% /snap/core/8689
/dev/loop9                   squashfs     56064     56064         0  100% /snap/core18/1668
/dev/loop11                  squashfs     56064     56064         0  100% /snap/core18/1650
/dev/loop10                  squashfs      4352      4352         0  100% /snap/gnome-calculator/536
/dev/loop12                  squashfs     15232     15232         0  100% /snap/remmina/3995
/dev/loop13                  squashfs    267008    267008         0  100% /snap/kde-frameworks-5-core18/32
/dev/loop14                  squashfs       256       256         0  100% /snap/gtk2-common-themes/5
/dev/loop15                  squashfs     15232     15232         0  100% /snap/remmina/3968
/dev/loop16                  squashfs     15104     15104         0  100% /snap/gnome-characters/399
/dev/sda1                    vfat        523248      2284    520964    1% /boot/efi
/dev/loop17                  squashfs    160512    160512         0  100% /snap/gnome-3-28-1804/110
/dev/loop18                  squashfs    127872    127872         0  100% /snap/vscode/87
/dev/loop19                  squashfs     15104     15104         0  100% /snap/gnome-characters/375
/dev/loop20                  squashfs     98688     98688         0  100% /snap/typora-alanzanattadev/2
/dev/loop21                  squashfs      3840      3840         0  100% /snap/gnome-system-monitor/123
/dev/loop22                  squashfs     98176     98176         0  100% /snap/kdenlive/22
/dev/loop23                  squashfs     98304     98304         0  100% /snap/kdenlive/13
/dev/loop24                  squashfs     46080     46080         0  100% /snap/gtk-common-themes/1440
/dev/loop25                  squashfs    144128    144128         0  100% /snap/gnome-3-26-1604/97
/dev/loop26                  squashfs     45312     45312         0  100% /snap/gtk-common-themes/1353
/dev/loop27                  squashfs    259584    259584         0  100% /snap/electronic-wechat/7
/dev/loop28                  squashfs     93568     93568         0  100% /snap/core/8592
/dev/loop29                  squashfs      3840      3840         0  100% /snap/gnome-system-monitor/127
tmpfs                        tmpfs       392012        16    391996    1% /run/user/121
tmpfs                        tmpfs       392012        36    391976    1% /run/user/1000
```

```
kylechen@kyle:~/test$ df -t ext4
文件系统                        1K-块      已用      可用 已用% 挂载点
/dev/mapper/ubuntu--vg-root 243559804 181980664 49137336   79% /
```

```
kylechen@kyle:~/test$ df -x ext4
文件系统         1K-块    已用     可用 已用% 挂载点
udev           1936656       0 1936656    0% /dev
tmpfs           392016    2068  389948    1% /run
tmpfs          1960060  299520 1660540   16% /dev/shm
tmpfs             5120       4    5116    1% /run/lock
tmpfs          1960060       0 1960060    0% /sys/fs/cgroup
/dev/loop0        1024    1024       0  100% /snap/gnome-logs/81
/dev/loop1      127360  127360       0  100% /snap/vscode/93
/dev/loop2        4352    4352       0  100% /snap/gnome-calculator/544
/dev/loop3      144128  144128       0  100% /snap/gnome-3-26-1604/98
/dev/loop4        1024    1024       0  100% /snap/gnome-logs/73
/dev/loop5         256     256       0  100% /snap/gtk2-common-themes/9
/dev/loop6      126976  126976       0  100% /snap/vscode/89
/dev/loop7      164096  164096       0  100% /snap/gnome-3-28-1804/116
/dev/loop8       93568   93568       0  100% /snap/core/8689
/dev/loop9       56064   56064       0  100% /snap/core18/1668
/dev/loop11      56064   56064       0  100% /snap/core18/1650
/dev/loop10       4352    4352       0  100% /snap/gnome-calculator/536
/dev/loop12      15232   15232       0  100% /snap/remmina/3995
/dev/loop13     267008  267008       0  100% /snap/kde-frameworks-5-core18/32
/dev/loop14        256     256       0  100% /snap/gtk2-common-themes/5
/dev/loop15      15232   15232       0  100% /snap/remmina/3968
/dev/loop16      15104   15104       0  100% /snap/gnome-characters/399
/dev/sda1       523248    2284  520964    1% /boot/efi
/dev/loop17     160512  160512       0  100% /snap/gnome-3-28-1804/110
/dev/loop18     127872  127872       0  100% /snap/vscode/87
/dev/loop19      15104   15104       0  100% /snap/gnome-characters/375
/dev/loop20      98688   98688       0  100% /snap/typora-alanzanattadev/2
/dev/loop21       3840    3840       0  100% /snap/gnome-system-monitor/123
/dev/loop22      98176   98176       0  100% /snap/kdenlive/22
/dev/loop23      98304   98304       0  100% /snap/kdenlive/13
/dev/loop24      46080   46080       0  100% /snap/gtk-common-themes/1440
/dev/loop25     144128  144128       0  100% /snap/gnome-3-26-1604/97
/dev/loop26      45312   45312       0  100% /snap/gtk-common-themes/1353
/dev/loop27     259584  259584       0  100% /snap/electronic-wechat/7
/dev/loop28      93568   93568       0  100% /snap/core/8592
/dev/loop29       3840    3840       0  100% /snap/gnome-system-monitor/127
tmpfs           392012      16  391996    1% /run/user/121
tmpfs           392012      36  391976    1% /run/user/1000
```

网络相关命令

netstat

功能

显示网络状态。

输入

```
netstat [参数]

-a-显示所有socket
-l 显示所有处于监听状态的socket
-t 显示所有使用tcp协议的socket
-u-显示UDP传输协议的连线状况。
-s 显示所有网络协议的统计信息。
```

输出

```
1. 列出所有端口 (包括监听和未监听的)
列出所有端口:     netstat -a
列出所有tcp端口:  netstat -at
列出所有udp端口:  netstat -au
2. 列出所有处于监听状态的 Sockets
只显示监听端口:          netstat -l
只列出所有监听tcp端口:   netstat -lt
只列出所有监听udp端口:   netstat -lu
只列出所有监听UNIX端口:  netstat -lx
3. 显示每个协议的统计信息
显示所有端口的统计信息 netstat -s
```

ifconfig

功能

用于显示或设置网络设备

输入

```
语法
ifconfig [网络设备][down up ][add<地址>][del<地址>][<hw<网络设备类型><硬件地址>][mtu<字节>][netmask<子网掩码>][IP地址]
参数说明:

add<地址> 设置网络设备IPv6的IP地址。
del<地址> 删除网络设备IPv6的IP地址。
down 关闭指定的网络设备。
up 启动指定的网络设备。
<hw<网络设备类型><硬件地址> 设置网络设备的类型与硬件MAC地址。

mtu<字节> 设置网络设备的MTU。
netmask<子网掩码> 设置网络设备的子网掩码。
[IP地址] 指定网络设备的IP地址。
```

实例

显示网络设备信息

```
# ifconfig
eth0   Link encap:Ethernet HWaddr 00:50:56:0A:0B:0C
     inet addr:192.168.0.3 Bcast:192.168.0.255 Mask:255.255.255.0
     inet6 addr: fe80::250:56ff:fe0a:b0c/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
     RX packets:172220 errors:0 dropped:0 overruns:0 frame:0
     TX packets:132379 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:87101880 (83.0 MiB) TX bytes:41576123 (39.6 MiB)
     Interrupt:185 Base address:0x2024

lo    Link encap:Local Loopback
     inet addr:127.0.0.1 Mask:255.0.0.0
     inet6 addr: ::1/128 Scope:Host
     UP LOOPBACK RUNNING MTU:16436 Metric:1
     RX packets:2022 errors:0 dropped:0 overruns:0 frame:0
     TX packets:2022 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:0
     RX bytes:2459063 (2.3 MiB) TX bytes:2459063 (2.3 MiB)
```

启动关闭指定网卡

```
# ifconfig eth0 down
# ifconfig eth0 up
```

为网卡配置和删除IPv6地址

```
# ifconfig eth0 add 33ffe:3240:800:1005::2/ 64 //为网卡诿之IPv6地址

# ifconfig eth0 del 33ffe:3240:800:1005::2/ 64 //为网卡删除IPv6地址
```

用ifconfig修改MAC地址

```
# ifconfig eth0 down //关闭网卡
# ifconfig eth0 hw ether 00:AA:BB:CC:DD:EE //修改MAC地址
# ifconfig eth0 up //启动网卡
# ifconfig eth1 hw ether 00:1D:1C:1D:1E //关闭网卡并修改MAC地址
# ifconfig eth1 up //启动网卡
```

配置IP地址

```
# ifconfig eth0 192.168.1.56
//给eth0网卡配置IP地址
# ifconfig eth0 192.168.1.56 netmask 255.255.255.0
// 给eth0网卡配置IP地址,并加上子掩码
# ifconfig eth0 192.168.1.56 netmask 255.255.255.0 broadcast 192.168.1.255
// 给eth0网卡配置IP地址,加上子掩码,加上个广播地址
```

启用和关闭ARP协议

```
# ifconfig eth0 arp  //开启
# ifconfig eth0 -arp  //关闭
```

设置最大传输单元

```
# ifconfig eth0 mtu 1500
//设置能通过的最大数据包大小为 1500 bytes
```

bash相关操作

在这里插入图片描述

数据流重导向

功能

将命令在终端的标准输出 标准出错 重定向到别的地方比如文件中 或者读取文件的内容来取代标准输入

输入语法

```
1. 标准输入 (stdin) ：代码为 0 ，使用 < 或 << ；一般和cat配合使用将一个文件的内容输出到另外一个文件
2. 标准输出 (stdout)：代码为 1 ，使用 > 或 >> ；将命令的标准输出重定向到另外一个位置 比如文件中
3.  标准错误输出(stderr)：代码为 2 ，使用 2> 或 2>> 将命令的标注出错重定向到另外一个位置比如文件中
```

实例

```
实例1:
将命令"ll / "的标准输出存到 ~/rootfile这个文件中  (也可以用1>)
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件 则这个文件的内容将会被全部替换
[dmtsai@study ~]$ ll / > ~/rootfile
```

```
实例2:
将命令"ll / "的标准输出存到 ~/rootfile这个文件中(也可以用1>>)
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件 则会将标准输出的内容添加到这个文件原有内容的末尾
[dmtsai@study ~]$ ll / >> ~/rootfile
```

```
实例3:
将命令"ll / "的标准错误存到 ~/rootfile这个文件中
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件 则这个文件的内容将会被全部替换
[dmtsai@study ~]$ ll / 2> ~/rootfile
```

```
实例4:
将命令"ll / "的标准错误存到 ~/rootfile这个文件中
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件   则会将标准错误的内容添加到这个文件原有内容的末尾
[dmtsai@study ~]$ ll / 2>>~/rootfile
```

```
实例5:
将命令"ll / "的标准输出存到 stdoutput.txt        将标准错误输出到stderr.txt
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件 则这个文件的内容将会被全部替换
[dmtsai@study ~]$ ll / 1>stdoutput.txt 2>~/stderr.txt
```

```
实例6:
将命令"ll / "的标准输出和标准错误都存到 stdoutput.txt
如果原本没有这个文件 则这个文件将被创建
如果原本有这个文件   则这个文件的内容将会被全部替换
[dmtsai@study ~]$ ll / 1>stdoutput.txt  2>&1  正确写法
[dmtsai@study ~]$ ll / 1>stdoutput.txt  2>~/stdoutput.txt    错误写法 会导致文件中两种输出是乱序的
```

```
实例7:
将命令"ll / "的标准输出存到 stdoutput.txt         标准错误存到垃圾桶(/dev/null)
  标准错误不会显示 标准输出存到文件stdoutput.txt中
[dmtsai@study ~]$ ll /   1>~/stdoutput.txt 2>/dev/null
```

```
实例8:
利用 cat 指令来建立一个文件的简单流程
[dmtsai@study ~]$ cat > catfile
testing
cat file test
<==这里按下 [ctrl]+d 来离开
```

```
实例9:
用 标准输入 取代键盘的输入以建立新文件的简单流程
也就是将/.bashrc的内容传到catfile里面 (<是覆盖 <<是补充到文件末尾)
[dmtsai@study ~]$ cat > catfile < ~/.bashrc
[dmtsai@study ~]$ ll catfile ~/.bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Mar 6 06:06 /home/dmtsai/.bashrc
-rw-rw-r--. 1 dmtsai dmtsai 231 Jul 9 18:58 catfile
# 注意看，这两个文件的大小会一模一样！几乎像是使用 cp 来复制一般！
```

```
实例10:
键盘的输入以建立新文件 以指定的字符作为结束符
[dmtsai@study ~]$ cat > catfile << "eof"
> This is a test.
> OK now stop
> > eof <==输入这关键词，立刻就结束而不需要输入 [ctrl]+d

[dmtsai@study ~]$ cat catfile
This is a test.
OK now stop <==只有这两行，不会存在关键词那一行
```

多个命令同时执行

功能

某些情况下，很多指令我想要一次输入去执行，而不想要分次执行时

输入语法

```
方法一:用分号连接 命令之间没有关联
sync; sync; shutdown -h now
```

```
方法二:用&&连接
cmd1 && cmd2
1. 若 cmd1 执行完毕且正确执行($?=0)，则开始执行 cmd2。
2. 若 cmd1 执行完毕且为错误 ($?≠0)，则 cmd2 不执行。
```

```
方法三:用||连接
cmd1 || cmd2
3. 若 cmd1 执行完毕且正确执行($?=0)，则 cmd2 不执行。
4. 2. 若 cmd1 执行完毕且为错误 ($?≠0)，则开始执行 cmd
```

命令的多行显示

功能

如果命令太长可以用多行显示

输入

用\符号分割

实例

```
[dmtsai@study ~]$ cp /var/spool/mail/root /etc/crontab \
> /etc/fstab /root
上面这个指令用途是将三个文件复制到 /root 这个目录下而已。
```

命令的通配符

功能

对命令进行模糊匹配

输入

在这里插入图片描述

实例

```
范例一:找出 /etc/ 底下以 cron 为开头的档名
[dmtsai@study ~]$ ll -d /etc/cron*
<==加上 -d 是为了仅显示目录而已

范例二:找出 /etc/ 底下文件名『刚好是五个字母』的文件名
[dmtsai@study ~]$ ll -d /etc/?????

范例三:找出 /etc/ 底下文件名含有数字的文件名
[dmtsai@study ~]$ ll -d /etc/*[0-9]*
<==记得中括号左右两边均需 *

范例四:找出 /etc/ 底下,档名开头非为小写字母的文件名:
[dmtsai@study ~]$ ll -d /etc/[^a-z]*
<==注意中括号左边没有 *

范例五:将范例四找到的文件复制到 /tmp/upper 中
[dmtsai@study ~]$ mkdir /tmp/upper; cp -a /etc/[^a-z]* /tmp/upper
```

命令的定时执行

功能

即crontab命令 ,设定某个命令的周期性执行

输入

首先要介绍一下 crond，因为 crontab 命令需要 crond服务支持。cron 是 Linux 下用来周期地执行某种任务或等待处理某些事件的一个守护进程， 和 Windows 中的计划任务有些类似。所以要先打开cron服务

crontab

命令的执行历史

功能

即history命令 ,获取你在终端输入的命令历史

```
[dmtsai@study ~]$ history [n]
[dmtsai@study ~]$ history [-c]
 n :数字,意思是『要列出最近输入的 n条命令 』
-c :将目前的 shell 中的所有 history 内容全部消除
```

命令的别名

功能

即 给命令指定一个别名 主要用在命令比较长的时候

```
 alias lm='ls -al | more' 比如这个把ls -al | more重命名为lm
```

```
unalias lm  撤销刚才的重命名
```

命令的解释

功能

即获取命令的manpage 用man命令

变量相关命令

变量的声明赋值

功能

相当于是声明一个变量 同时给变量赋值

输入

声明方法1: 通过 赋值符号 =
声明方法2: 通过read

用read以后 变量赋值是通过**终端等待用户的键盘输入** 来执行

```
[dmtsai@study ~]$ read [-pt] variable
选项与参数:
-p ： 后面可以接提示字符!
-t ： 后面可以接等待的『秒数!』
```

实例

```
变量的普通赋值形式
[dmtsai@study ~]$ CHENWEIJIE=0820
```

```
让用户由键盘输入一内容，将该内容变成名为 atest 的变量
[dmtsai@study ~]$ read ates
This is a test <==此时光标会等待你输入！请输入左侧文字看看
[dmtsai@study ~]$ echo ${atest}
This is a test <==你刚刚输入的数据已经变成一个变量内
```

```
让用户由键盘输入一内容，会有提示的字符串提示你输入 ，然后将该内容变成名为 name 的变量
kylechen@kyle:~$ read -p"请输入你的姓名:" -t 5 name
请输入你的姓名:
kylechen@kyle:~$ echo $name
chenweijie
```

变量的查看转换

功能

查看或转化指定的变量

输入

```
echo 查看指定单个变量的值 可以用用**echo PATH** 或者 **echo ${PATH}** 格式
env  查看所有环境变量
set 查看所有变量(包括用户设定的变量和环境变量,其他bash接口变量)
export  变量名=变量值 将自定义变量转化成环境变量

关于环境变量:
1.  环境变量是用来定义系统运行环境的一些参数，比如每个用户不同的家目录（HOME）、邮件存放位置（MAIL）等。
2.   环境变量所有终端下所有程序都可直接取用的   可以理解为 环境变量=全局变量   自定义变量=局部变量
3.   可以通过export将自定义变量设定为当前终端下的环境变量 但是也就是只能当前终端下的父子进程可以取用 其他终端不行
 4.  可以通过自定义变量的export语句添加到个人目录下的.bashrc文件中 这样当前用户的所有程序都可以使用这个环境变量
5.   可以通过将自定义变量的export语句添加到/etc/profile文件下 这样所有用户的所有程序都可以使用这个环境变量
```

实例

```
输出自定义变量myname

[dmtsai@study ~]$ echo ${myname}
 <==这里并没有任何数据~因为这个变量尚未被设定！是空的[
 dmtsai@study ~]$ myname=VBird
 [dmtsai@study ~]$ echo ${myname}
 VBird <==出现了！因为这个变量已经被设定
```

```
输出变量PATH
[dmtsai@study ~]$ echo  $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/b
```

```
输出变量PATH
[dmtsai@study ~]$ echo  ${PATH}
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/b
```

```
列出目前的 shell 环境下的所有环境变量与其内容。
[dmtsai@study ~]$ env
```

```
列出目前的 shell 环境下的所有变量。
[dmtsai@study ~]$ set
```

```
将自定义的变量CHENWEIJIE转换成环境变量
export CHENWEIJIE=0820
```

变量的类型指定

功能

变量声明的时候指定类型

输入

```
$ declare [-aixr] variable

选项与参数:
-a ：将后面名为 variable 的变量定义成为数组 (array) 类型
-i ：将后面名为 variable 的变量定义成为整数数字 (integer) 类型
-x ：用法与 export 一样，就是将后面的 variable 变成环境变量;
-r ：将变量设定成为 readonly 类型，该变量不可被更改内容，也不能 unset
```

实例

```
范例一：让变量 sum 进行 100+300+50 的加总结果
[dmtsai@study ~]$ sum=100+300+5
[dmtsai@study ~]$ echo ${sum}100+300+50 <==咦！怎么没有帮我计算加总？因为这是文字型态的变量属性啊！[dmtsai@study ~]$ declare
[dmtsai@study ~]$ echo ${sum}
450
```

```
范例二：将 sum 变成环境变量
[dmtsai@study ~]$ declare -x sum
[dmtsai@study ~]$ export | grep sum
declare -ix sum="450" <==果然出现了！包括有 i 与 x 的宣告！

范例三：让 sum 变成只读属性，不可更动！[dmtsai@study ~]$ declare -r sum
[dmtsai@study ~]$ sum=tesgtin-bash: sum: readonly variable <==老天爷～不能改这个变数了！

范例四：让 sum 变成非环境变量的自定义变量吧
[dmtsai@study ~]$ declare +x sum <== 将 - 变成 + 可以进行『取消』动作

[dmtsai@study ~]$ declare -p sum <== -p 可以单独列出变量的类型
declare -ir sum="450" <== 看吧！只剩下 i, r 的类型，不具有 x
```

变量的比较判断

功能

类似test命令 利用中括号等运算符判断里面的变量是否为空 里面的变量比较是否为真等任务

输入

```
[ "$HOME" == "$MAIL"]  判断变量HOME和MAIL是否相同
 [ -z "${HOME}" ]  判断变量HOME是否为空

如上所示 中括号的使用方法与 test 几乎一模一样啊
只不过需要有以下注意点
```

使用注意点:

```
1.在中括号内的变量，最好都以双引号括号起来；  因为可能出现以下问题
```

在这里插入图片描述

```
2/在中括号内的常数，最好都以单或双引号括号
```

```
3.在中括号 [] 内的每个组件都需要有空格键来分隔；
使用中括号必须要特别注意，因为中括号用在很多地方，
包括通配符与正规表示法等等，
所以如果要在 bash 的语法当中使用中括号作为 shell 的判断式时，
必须要注意中括号的两端需要有空格符来分隔
 假设我空格键使用『□』符号来表示，那么，在这些地方你都需要有空格键：
```

在这里插入图片描述

管道相关命令

在这里插入图片描述

cut

功能

主要的用途在于将『同一行里面的数据进行分解！』 处理的讯息是以『行』为单位 以某些字符当作分割的参数，然后来将数据加以切割， 以取得我们所需要的数据。

输入

```
cut[参数]
```

实例

grep

功能

刚刚的 cut 是将一行讯息当中，取出某部分我们想要的， 而 grep 则是分析一行讯息， 若当中有我 们所需要的信息，就将该行拿出来～

输入

在这里插入图片描述

实例

在这里插入图片描述

sort

功能

把输出的结果进行排序

输入

在这里插入图片描述

实例

uniq

功能

将重复的资料仅列出一个显示

输入

在这里插入图片描述

实例

在这里插入图片描述

wc

功能

统计有多少字符 多少行

输入

在这里插入图片描述

实例

在这里插入图片描述

Shell脚本相关命令

在这里插入图片描述

介绍

一句话说明shell脚本是个啥 shell脚本就是将多个shell指令汇集到一起去完成一个复杂的功能 类似windows下的批处理文件 一般以sh为文件后缀

语法

程序结构

一张图说明shell脚本的结构是怎样的
如上图所示 右边就是最简单的一个shell脚本，功能是打印出"Hello World 左边就是它对应的一个程序结构 ,依次是shell版本声明 注释部分声明 环境变量声明 程序主体 退出返回 下面依次讲解

shell版本声明

第一行 *#!/bin/bash 声明这个 脚本 使用的shell版本:*
因为shell有很多种类型 比如sh,bash,csh,tcsh
如果不指定版本 系统会不知道用哪一个版本的shell去处理这个脚本
所以会报错
一般linux发行版默认内置的是bash 所以一般是写*#!/bin/bash*

注释部分声明

说明程序的作者 功能 编辑时间等等信息

环境变量声明

环境变量是用来定义系统运行环境的一些参数的变量 比如每个用户不同的家目录（HOME）、邮件存放位置（MAIL）等。具体可以看后端开发必须掌握的Linux命令[变量篇]中的讲解

其中比较重要的一个环境变量是PATH
它代表的是一系列路径的字符串集合
它默认存放了下面的路径地址 如
usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin等等

比如当你在shell脚本中调用命令echo的时候
本质上系统就会从PATH存放的路径集合中
去寻找这些路径中是不是包含了命令echo对应的可执行文件
如果找的到 就执行 如果找不到的话你调用echo就会报错

这样就有一个问题
假如你调用的一个比较陌生的命令
它所对应的文件路径
不在PATH默认的路径集合中中

你就需要把它加入PATH (用冒号 等于号)然后在对它重新声明为环境变量(用export)

这样才可以 否则还是会报错的
一般常用的命令比如ls echo 等等
是默认已经包含在PATH变量的路径集合中了 所以开头不需要声明
这里只是为了讲解程序的标准结构故引出来了

程序主体

普通语句

就是用普通的命令组成的语句

条件语句

简单的if ....then

实例:

复杂的if ....then

实例:

case....esac

实例:

循环语句

while do done

当while中的条件为真的时候就继续执行
实例:

until do done

当until中的条件为真的时候就结束执行
实例:

for...do...done

实例:

for...do...done的数值处理结构

实例:

退出返回

在我们这个例子当中，使用 exit 0 ， 这代表离开 script 并且回传一个 0 给系统， 所以执行完这个 script 后，若接着下达 echo $? 则可得到0 值！利用这个 exit n (n 是数字) 的功能，我们还可以自定义错误返回值 从而使程序更聪明

执行

假设我们编写好的shell脚本的文件名为hello.sh，文件位置在/data/shell目录中并已有执行权限

执行方法一:通过路径

```
1、相对路径
cd /data/shell
./hello.sh

2、绝对路径
/data/shell/hello.sh
```

执行方法二:通过sh或bash

```
sh
cd /data/shell
sh hello.sh


bash
cd /data/shell
bash hello.sh
```

执行方法三:通过 source 或.(点号)

```
source
cd /data/shell
source hello.sh

.
cd /data/shell
. hello.sh
```

区别:

1.绝对路径和相对路径没有什么区别，两种方式都需要提前赋予脚本以执行权限。

2.sh或者bash方式是把脚本当做bash的调用来处理，所以，脚本不需要有执行权限就可以执行。前两种方式都是在当前shell中打开一个子shell来执行脚本内容，当脚本内容结束，则子shell关闭，回到父shell中。

3.source或者.方式是使脚本内容在当前shell里执行，而不是单独开子shell执行。

4.开子shell与不开子shell的区别就在于，环境变量的继承关系，如在子shell中设置的当前变量， 不做特殊通道处理的话，父shell是不可见的。而在当前shell中执行的话，则所有设置的环境变量都是直接生效可用的。

调试