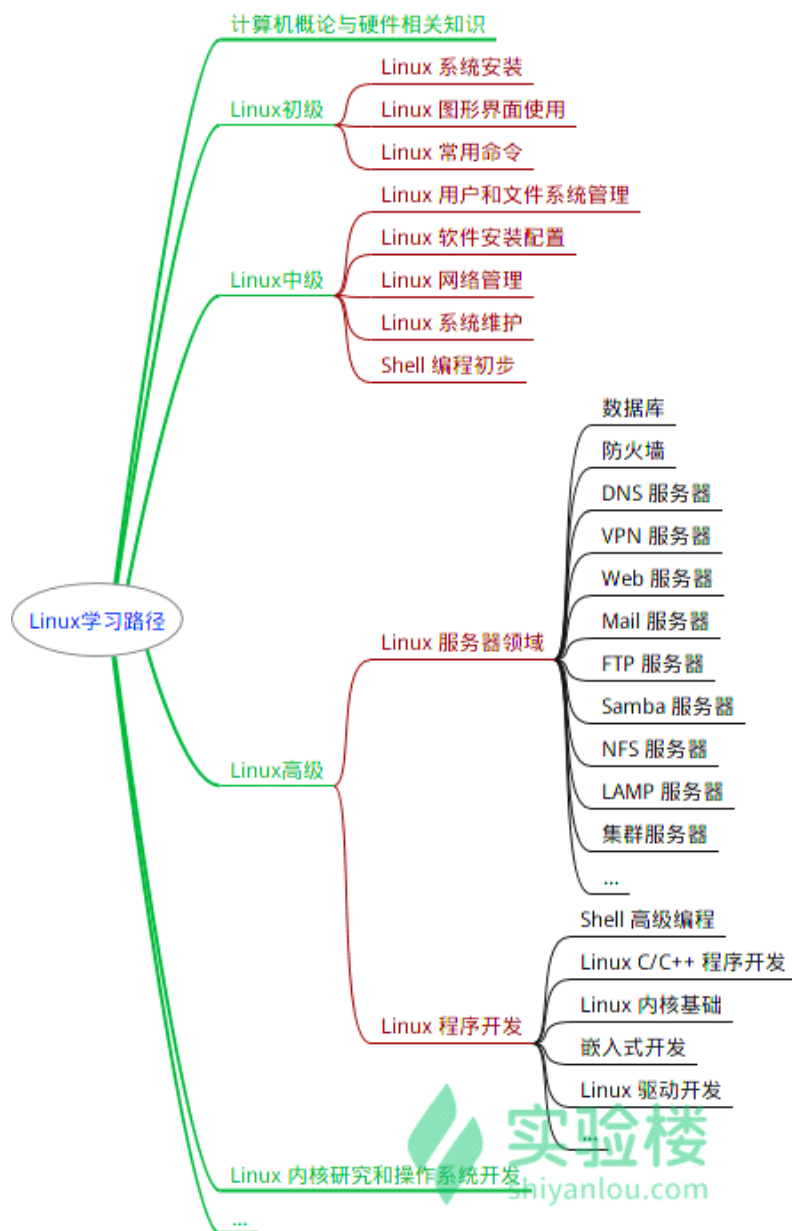


Linux学习笔记

2020年6月30日 11:02

Linux学习路径:



Linux终端快捷键:

按键	作用
Ctrl+d	键盘输入结束或退出终端
Ctrl+s	暂停当前程序，暂停后按下任意键恢复运行
Ctrl+z	将当前程序放到后台运行，恢复到前台为命令fg
Ctrl+a	将光标移至输入行头，相当于Home键
Ctrl+e	将光标移至输入行末，相当于End键
Ctrl+k	删除从光标所在位置到行末
Alt+Backspace	向前删除一个单词
Shift+PgUp	将终端显示向上滚动

Shift+PgDn

将终端显示向下滚动

Shell 常用通配符:

字符	含义
*	匹配 0 或多个字符
?	匹配任意一个字符
[list]	匹配 list 中的任意单一字符
[^list]	匹配 除 list 中的任意单一字符以外的字符
[c1-c2]	匹配 c1-c2 中的任意单一字符 如: [0-9][a-z]
{string1,string2,...}	匹配 string1 或 string2 (或更多)其一字符串
{c1..c2}	匹配 c1-c2 中全部字符 如{1..10}

who 命令其它常用参数

参数	说明
-a	打印能打印的全部
-d	打印死掉的进程
-m	同 <code>am i, mom likes</code>
-q	打印当前登录用户数及用户名
-u	打印当前登录用户登录信息
-r	打印运行等级

用户操作:

添加用户: `sudo adduser <user>`

useradd 只创建用户, 不会创建用户密码和工作目录, 创建完了需要使用 `passwd <username>` 去设置新用户的密码。adduser 在创建用户的同时, 会创建工作目录和密码 (提示你设置), 做这一系列的操作。其实 useradd、userdel 这类操作更像是一种命令, 执行完了就返回。而 adduser 更像是一种程序, 需要你输入、确定等一系列操作。

设置用户密码: `sudo passwd <user>`

切换登录用户: `su -l <user>`

查看打开当前伪终端的用户的用户名: `who am i/who mom likes`

查看当前登录用户的用户名: `whoami`

查看用户所属组: `groups shiyanlou`

将用户添加到sudo用户组: `sudo usermod -G sudo <user>`

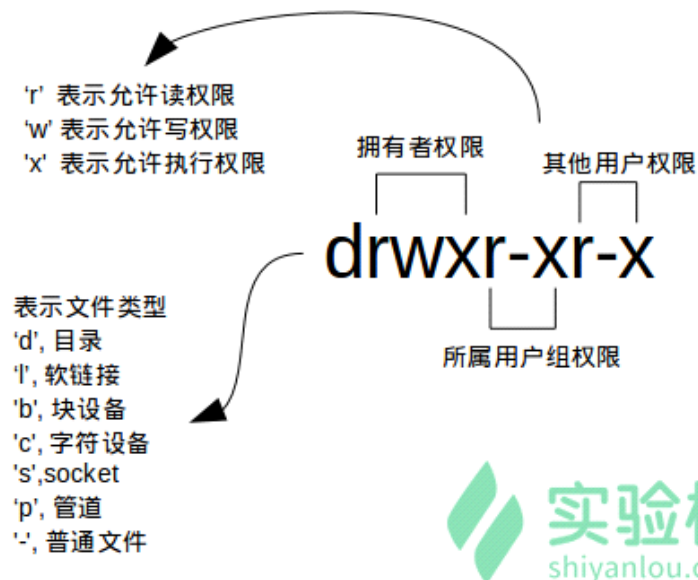
删除用户: `sudo deluser <user> --remove-home`

文件权限:

执行 `ls -l` 后的结果意思:

文件类型和权限	所有者	文件大小	文件名
drwxr-xr-x	2 shiyanlou	shiyanlou	4096 11 月 13 15:27 Documents
链接数	所属用户组	最后修改时间	

实验楼 shiyanlou.com



查看全部文件: `ls -a`

查看某一个目录的完整属性, 而不是显示目录里面的文件属性: `ls -dl <目录名>`

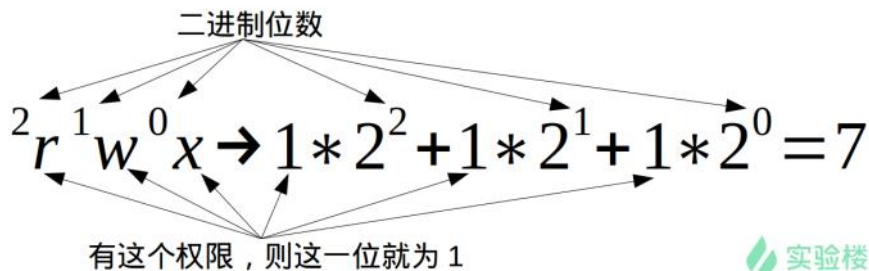
显示所有文件大小, 并以普通人类能看懂的方式呈现: `ls -asSh` 其中小 `s` 为显示文件大小, 大 `S` 为按文件大小排序

变更文件所有者: `sudo chown <user> <filename>`

修改文件权限的两种操作:

1、数字表示: `chmod 600 <filename>`

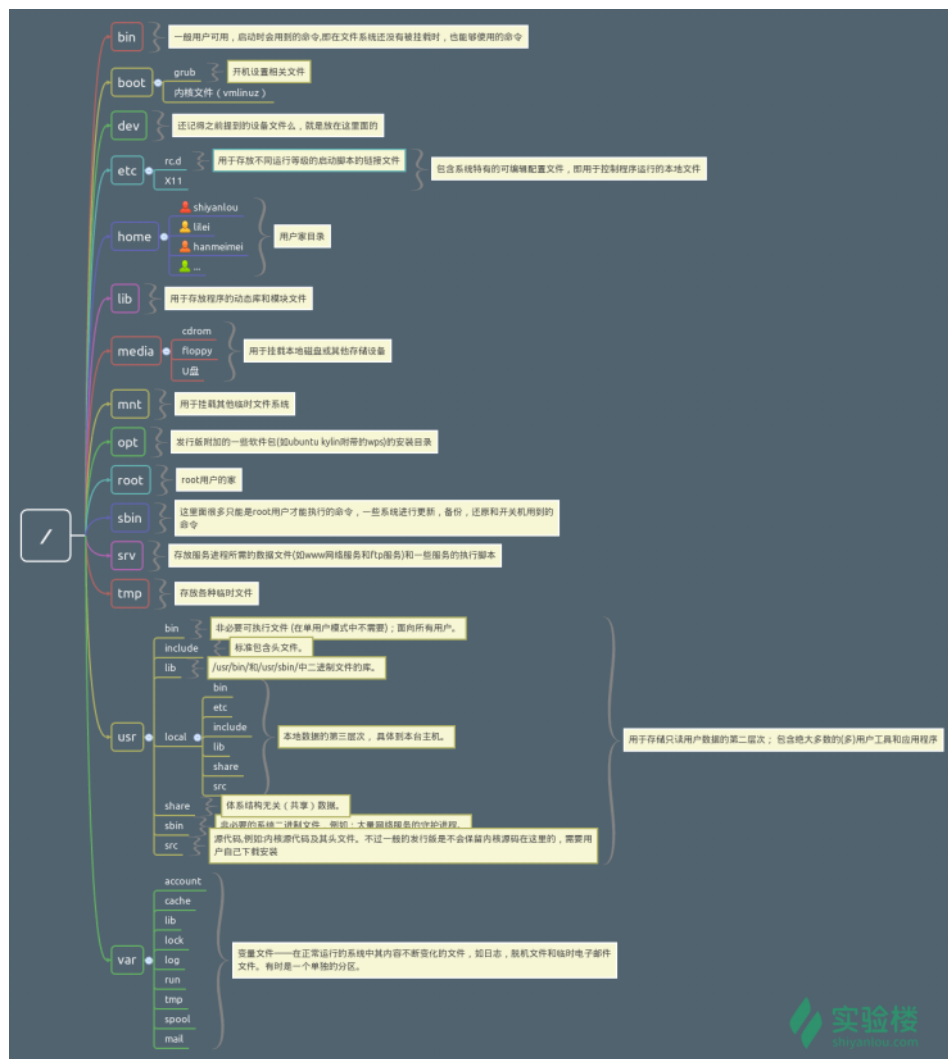
进制和权限的关系:



2、加减赋值操作: `chmod go-rw <filename>`, `g`、`o` 还有 `u` 分别表示 `group` (用户组)、`others` (其他用户) 和 `user` (用户), `+` 和 `-` 分别表示增加和去掉相应的权限。

Linux目录:

目录简介[看不清戳这里](#):



目录操作命令:

新建目录: `mkdir <dirname>`

新建多级目录: `mkdir -p <dirname>/<dirname>/<dirname>`

复制目录: `cp -r/R <source-dirname> <des-dirname>`

删除目录: `rm -r <dirname>` (强制删除: `rm -rf <dirname>`)

进入上一级: `cd ..`

进入当前用户的home目录: `cd ~`

获取当前路径: `pwd`

文件操作:

新建文件: `touch <filename>`

复制文件: `cp <filename> <dirname>`

删除文件: `rm <filename>` (强制删除: `rm -f <filename>`)

移动文件: `mv <filename> <dirname>`

重命名文件: `mv <oldfilename> <newfilename>`

使用通配符批量创建 5 个文件: `touch file{1..5}.txt`

批量将后缀为.txt的文本文件重命名为以.c为后缀的文件:

`rename 's/\.txt/\.c/' *.txt`

批量将文件名和后缀改为大写: `rename 'y/a-z/A-Z/' *.c`

查看文件:

使用 `cat`, `tac` 和 `nl` 命令查看文件:

1、前两个命令都是用来打印文件内容到标准输出 (终端), 其中`cat`为正序显示, `tac`为倒序显示, 可以加上 `-n` 参数显示行号;

2、`nl`命令, 添加行号并打印, 这是个比`cat -n`更专业的行号打印命令。它的常用

的几个参数：

- b : 指定添加行号的方式，主要有两种：
 - b a:表示无论是否为空行，同样列出行号（“cat -n”就是这种方式）
 - b t:只列出非空行的编号并列出（默认为这种方式）
- n : 设置行号的样式，主要有三种：
 - n ln:在行号字段最左端显示
 - n rn:在行号字段最右边显示，且不加 0
 - n rz:在行号字段最右边显示，且加 0
- w : 行号字段占用的位数(默认为 6 位)

使用 more 和 less 命令分页查看文件，打开后默认只显示一屏内容，终端底部显示当前阅读的进度。可以使用 Enter 键向下滚动一行，使用 Space 键向下滚动一屏，按下 h 显示帮助，q 退出。

使用 head 和 tail 命令查看文件，它们一个是只查看文件的头几行（默认为 10 行，不足 10 行则显示全部）和尾几行。

使用 file 命令查看文件的类型：与 Windows 不同的是，如果新建了一个 .txt 文件，Windows 会自动把它识别为文本文件，而 file 命令会识别为一个空文件。在 Linux 中文件的类型不是根据文件后缀来判断的。当在文件里输入内容后才会显示文件类型。

搜索文件：

whereis 只能搜索二进制文件(-b)，man 帮助文件(-m)和源代码文件(-s)。如果想要获得更全面的搜索结果可以使用 locate 命令。

locate 命令可以用来查找指定目录下的不同文件类型，它不只是在目录下查找，还会自动递归子目录进行查找。如果想只统计数目可以加上 -c 参数，-i 参数可以忽略大小写进行查找，whereis 的 -b、-m、-s 同样可以使用。

which 本身是 Shell 内建的一个命令，我们通常使用 which 来确定是否安装了某个指定的程序，因为它只从 PATH 环境变量指定的路径中去搜索命令并且返回第一个搜索到的结果。也就是说，我们可以看到某个系统命令是否存在以及执行的到底是哪一个地方的命令。

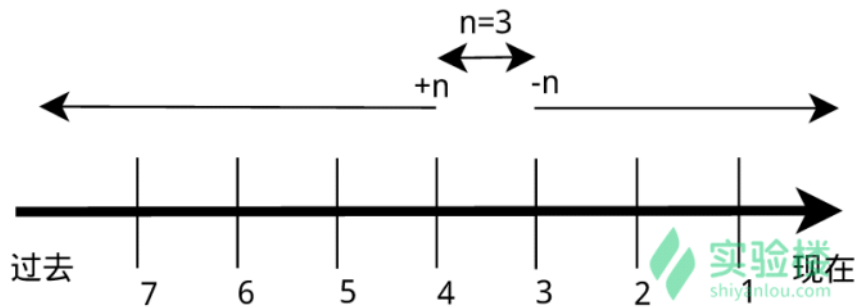
find 应该是这几个命令中最强大的了，它不但可以通过文件类型、文件名进行查找而且可以根据文件的属性（如文件的时间戳，文件的权限等）进行搜索。find 命令的路径是作为第一个参数的，基本命令格式为 `find [path][option] [action]`。

与时间相关的命令参数：

参数	说明
-atime	最后访问时间
-ctime	最后修改文件内容的时间
-mtime	最后修改文件属性的时间

下面以 -mtime 参数举例：

- mtime n: n 为数字，表示为在 n 天之前的“一天之内”修改过的文件
- mtime +n: 列出在 n 天之前（不包含 n 天本身）被修改过的文件
- mtime -n: 列出在 n 天之内（包含 n 天本身）被修改过的文件
- newer file: file 为一个已存在的文件，列出比 file 还要新的文件名



文件压缩与打包:

zip:

使用zip打包文件夹: `zip -r -q -o <name>.zip <dirname>`, `-r` 参数表示递归打包包含子目录的全部内容, `-q` 参数表示为安静模式, 即不向屏幕输出信息, `-o` 表示输出文件, 需在其后紧跟打包输出文件名。使用 `du` 命令查看打包后文件的大小:

`du -h <name>.zip`。

设置压缩级别为 9 和 1 (9 最大, 1 最小), 重新打包: `zip -r -9 -q -o <name>.zip <dirname> -x ~/.zip`。这里添加了一个参数用于设置压缩级别 `-[1-9]`, 1 表示最快压缩但体积大, 9 表示体积最小但耗时最久。最后那个 `-x` 是为了上一次创建的 zip 文件, 否则又会被打包进这一次的压缩文件中, 注意: 这里只能使用绝对路径, 否则不起作用。

用 `du` 命令分别查看默认压缩级别、最低、最高压缩级别及未压缩的文件的大小: `du -h -d 0 *.zip ~ | sort`。其中: `h`, `--human-readable` (顾名思义) `d`, `--max-depth` (所查看文件的深度)。

创建加密 zip 包: `zip -r -e -o <name>.zip <dirname>`。

如果要在 Linux 创建的 zip 压缩文件在 Windows 上解压后没有任何问题, 还需要对命令做一些修改: `zip -r -l -o <name>.zip <dirname>`, 加上 `-l` 参数将 LF 转换为 CR+LF 来达到以上目的。

解压压缩文件到当前目录: `unzip <name>.zip`, 安静模式: `unzip -q <name>.zip -d <dirname>`, `-d`后面是指定目录, 如果没有或自动创建。只查看压缩包内容:

`unzip -l <name>.zip`, 为解决中文乱码的问题, 在解压时使用 `-O` (英文字母, 大写 `o`) 参数指定编码类型: `unzip -O GBK 中文压缩文件.zip`。

tar:

创建一个 tar 包: `tar -P -cf <name>.tar <dirname>`。`-P` 保留绝对路径符, `-c` 表示创建一个 tar 包文件, `-f` 用于指定创建的文件名, 注意文件名必须紧跟在 `-f` 参数之后。还可以加上 `-v` 参数以可视的方式输出打包的文件。解包一个文件 (`-x` 参数) 到指定路径的已存在目录 (`-C` 参数): `tar -xf <name>.tar -C <dirname>`。只查看不解包文件 `-t` 参数: `tar -tf <name>.tar`。保留文件属性和跟随链接 (符号链接或软链接), 有时候我们使用 tar 备份文件当你在其他主机还原时希望保留文件的属性 (`-p` 参数) 和备份链接指向的源文件而不是链接本身 (`-h` 参数): `tar -cphf <name>.tar <dirname>`。只需要在创建 tar 文件的基础上添加 `-z` 参数, 使用 `gzip` 来压缩文件: `tar -czf <name>.tar.gz <dirname>`。解压 `*.tar.gz` 文件: `tar -xzf <name>.tar.gz`。要使用其它的压缩工具创建或解压相应文件只需要更改一个参数即可:

压缩文件格式	参数
<code>*.tar.gz</code>	<code>-z</code>
<code>*.tar.xz</code>	<code>-J</code>
<code>*tar.bz2</code>	<code>-j</code>

总结:

常用命令:

zip:

打包 : `zip something.zip something` (目录请加 `-r` 参数)

解包: `unzip something.zip`

指定路径: `-d` 参数

`tar:`

打包: `tar -cf something.tar something`

解包: `tar -xf something.tar`

指定路径: `-C` 参数

文件系统操作与磁盘管理:

使用 `df` 命令查看磁盘的容量, 对其输出的解释: 物理主机上的 `/dev/sda2` 是对应着主机硬盘的分区, 后面的数字表示分区号, 数字前面的字母 `a` 表示第几块硬盘 (也可能是可移动磁盘), 如果主机上有多块硬盘则可能还会出现 `/dev/sdb`, `/dev/sdc` 这些磁盘设备都会在 `/dev` 目录下以文件的存在形式。“1k-块”这个陌生的东西, 它表示以磁盘块大小的方式显示容量, 后面为相应的以块大小表示的已用和可用容量。加 `-h` 输出更易懂。

使用 `du` 命令查看目录的容量, 加上 `-h` 参数, 以更易读的方式展示, `-d` 参数指定查看目录的深度:

只查看1级目录的信息: `du -h -d 0 ~`

查看2级: `du -h -d 1 ~`

常用参数:

`du -h` #同`--human-readable` 以K, M, G为单位, 提高信息的可读性。

`du -a` #同`--all` 显示目录中所有文件的大小。

`du -s` #同`--summarize` 仅显示总计, 只列出最后加总的值。

来自: <http://man.linuxde.net/du>

eg (找出当前目录下面占用最大的前十个文件):

`du -a | sort -nr | head -n 10`

`dd` 命令简介:

输出到文件: `dd of=<filename> bs=10 count=1` 或者 `dd if=/dev/stdin of=<filename> bs=10 count=1`。

输出到标准输出: `dd if=/dev/stdin of=/dev/stdout bs=10 count=1`

打完了命令后, 需要继续在终端打字, 作为输入。上述命令从标准输入设备读入用户输入 (缺省值, 所以可省略) 然后输出到文件, `bs` (block size) 用于指定块大小 (缺省单位为 Byte, 也可为其指定如 'K', 'M', 'G' 等单位), `count` 用于指定块数量。

`dd`在拷贝的同时还可以实现数据转换: `dd if=/dev/stdin of=test bs=10 count=1 conv=ucase`。

使用 `dd` 命令创建虚拟镜像文件: 从 `/dev/zero` 设备创建一个容量为 256M 的空文件:

eg: `dd if=/dev/zero of=virtual.img bs=1M count=256`

使用 `mkfs` 命令格式化磁盘: eg: `sudo mkfs.ext4 virtual.img`

使用 `mount` 命令挂载磁盘到目录树:

`mount`命令的一般格式如下:

`mount [options] [source] [directory]`

一些常用操作:

`mount [-o [操作选项]] [-t 文件系统类型] [-w|--rw|--ro] [文件系统源] [挂载点]`

eg:

`$ mount -o loop -t ext4 virtual.img /mnt`

也可以省略挂载类型, 很多时候 `mount` 会自动识别

以只读方式挂载

`$ mount -o loop --ro virtual.img /mnt`

或者 `mount -o loop,ro virtual.img /mnt`

```
# 命令格式 sudo umount 已挂载设备名或者挂载点，如：
$ sudo umount /mnt
```

帮助命令：

type命令可以用来区分命令是内建的还是外部的：

得到这样的结果说明是内建命令，内建命令都是在 bash 源码中的 builtins 的.def中：`xxx is a shell builtin`。

得到这样的结果说明是外部命令，外部命令在/usr/bin or /usr/sbin等等中：`xxx is /usr/bin/xxx`。

若是得到alias的结果，说明该指令为命令别名所设定的名称：`xxx is an alias for xx --xxx`。

bash内置的help命令只能用于显示 shell 内建命令的简要帮助信息，帮助信息中显示有该命令的简要说明以及一些参数的使用以及说明。外部命令基本上都有一个参数--help可以得到相应的帮助。

man 工具是显示系统手册页中的内容，也就是一本电子版的字典，这些内容大多数都是对命令的解释信息，还有一些相关的描述。通过查看系统文档中的 man 也可以得到程序的更多相关信息和 Linux 的更多特性。在尝试这个命令时我们会发现最左上角显示“LS (1)”，在这里，“LS”表示手册名称，而“(1)”表示该手册位于第一章。在 man 手册中一共有这么几个章节：

章节数	说明
1	Standard commands （标准命令）
2	System calls （系统调用）
3	Library functions （库函数）
4	Special devices （设备说明）
5	File formats （文件格式）
6	Games and toys （游戏和娱乐）
7	Miscellaneous （杂项）
8	Administrative Commands （管理员命令）
9	其他（Linux 特定的），用来存放内核例行程序的文档。

info 来自自由软件基金会的 GNU 项目，是 GNU 的超文本帮助系统，能够更完整的显示出 GNU 信息。所以得到的信息当然更多。man 和 info 就像两个集合，它们有一个交集部分，但与 man 相比，info 工具可显示更完整的 GNU 工具信息。若 man 页包含的某个工具的概要信息在 info 中也有介绍，那么 man 页中会有“请参考 info 页更详细内容”的字样。

crontab命令：

如果crontab没启动，启动命令：`sudo cron -f &`

添加任务：`crontab -e`

查看添加的任务：`crontab -l`

查看cron是否成功在后台启动：`ps aux | grep cron`或者`pgrep cron`

查看到执行任务命令之后在日志中的信息反馈：`sudo tail -f /var/log/syslog`

删除任务：`crontab -r`

cron 服务监测时间最小单位是分钟，所以 cron 会每分钟去读取一次 /etc/crontab 与 /var/spool/cron/crontabs 里面的内容。如果是系统级别的定时任务，只需要以 sudo 权限编辑 /etc/crontab 文件就可以。在 /etc 目录下，cron 相关的目录有下面几个：

/etc/cron.daily，目录下的脚本会每天执行一次，在每天的 6 点 25 分时运行；

/etc/cron.hourly，目录下的脚本会每小时执行一次，在每小时的 17 分钟时运

行;
/etc/cron.monthly, 目录下的脚本会每月执行一次, 在每月 1 号的 6 点 52 分时运行;
/etc/cron.weekly, 目录下的脚本会每周执行一次, 在每周第七天的 6 点 47 分时运行;
系统默认执行时间可以根据需求进行修改。

命令执行顺序控制与管道:

输入命令时可以通过在命令之间加入分号的形式使命令顺序执行。
可以使用`which`来查找是否安装某个命令(查找到命令则返回1), `&&`是用来实现选择性执行的, 它表示如果前面的命令执行结果(不是表示终端输出的内容, 而是表示命令执行状态的结果)返回 0 则执行后面的, 否则不执行; 可以从`$?`环境变量获取上一次命令的返回结果; `||`是与`&&`相反的控制效果, 当上一条命令执行结果为 $\neq 0$ (`$?\neq 0`)时则执行它后面的命令。

管道是一种通信机制, 通常用于进程间的通信(也可通过 `socket` 进行网络通信), 它表现出来的形式就是将前面每一个进程的输出(`stdout`)直接作为下一个进程的输入(`stdin`)。在使用一些过滤程序时经常会用到的就是匿名管道, 在命令行中由`|`分隔符表示。

`cut` 命令, 打印每一行的某一字段, 以下是应用示例:

打印/etc/passwd文件中以:为分隔符的第 1 个字段和第 6 个字段分别表示用户名和其家目录: `cut /etc/passwd -d ':' -f 1,6`。

打印/etc/passwd文件中每一行的前 N 个字符:

前五个(包含第五个): `cut /etc/passwd -c -5`

前五个之后的(包含第五个): `cut /etc/passwd -c 5-`

第五个: `cut /etc/passwd -c 5`

2到5之间的(包含第五个): `cut /etc/passwd -c 2-5`

`grep`命令的一般形式为: `grep [命令选项]... 用于匹配的表达式 [文件]...`。查看环境变量中以"yanlou"结尾的字符串: `export | grep ".*yanlou$"`, 其中`$`就表示一行的末尾。

`wc` 命令用于统计并输出一个文件中行、单词和字节的数目, 比如输出/etc/passwd文件的统计信息: `wc /etc/passwd`。分别只输出行数、单词数、字节数、字符数和输入文本中最长一行的字节数:

行数: `wc -l /etc/passwd`

单词数: `wc -w /etc/passwd`

字节数: `wc -c /etc/passwd`

字符数: `wc -m /etc/passwd`

最长行字节数: `wc -L /etc/passwd`

注: 对于西文字符来说, 一个字符就是一个字节, 但对于中文字符一个汉字是大于 2 个字节的, 具体数目是由字符编码决定的。

`sort`排序命令是将输入按照一定方式排序, 然后再输出, 它支持的排序有按字典排序, 数字排序, 按月份排序, 随机排序, 反转排序, 指定特定字段进行排序等等。一个例子:
`cat /etc/passwd | sort -t ':' -k 3 -n`。`-t`参数用于指定字段的分隔符, 这里是以":"作为分隔符; `-k` 字段号用于指定对哪一个字段进行排序。这里/etc/passwd文件的第三个字段为数字, 默认情况下是以字典序排序的, `-n`参数指按照数字排序。

`uniq`命令可以用于过滤或者输出重复行, 但只能去连续重复的行, 不是全文去重。可以使用`history`命令查看最近执行过的命令。如果只想查看使用了哪个命令而不需要知道具体干了什么, 那么可能就会要想去掉命令后面的参数然后去掉重复的命令: `history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq`或者`history | cut -c 8- | cut -d ' ' -f`

l | sort -u。

输出重复过的行（重复的只输出一个）及重复次数：`history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq -dc`

输出所有重复的行：`history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq -D`

简单的文本处理：

`tr` 命令可以用来删除一段文本信息中的某些文字，或者将其进行转换。使用方式为：`tr [option]...SET1 [SET2]`

常用选项有：

选项	说明
<code>-d</code>	删除和 <code>set1</code> 匹配的字符，注意不是全词匹配也不是按字符顺序匹配
<code>-s</code>	去除 <code>set1</code> 指定的在输入文本中连续并重复的字符

操作举例：

```
# 删除 "hello shiyanlou" 中所有的 'o', 'l', 'h'
$ echo 'hello shiyanlou' | tr -d 'olh'
e siyanu
# 将 "hello" 中的 ll, 去重为一个 l
$ echo 'hello' | tr -s 'l'
helo
# 将输入文本，全部转换为大写或小写输出
$ echo 'input some text here' | tr '[:lower:]' '[:upper:]'
INPUT SOME TEXT HERE
# 上面的 '[:lower:]' '[:upper:]' 你也可以简单的写作 '[a-z]' '[A-Z]'，当然反过来将大写变小写也是可以的
```

`col` 命令可以将 Tab 换成对等数量的空格键，或反转这个操作。使用方式：`col [option]`

常用的选项有：

选项	说明
<code>-x</code>	将 Tab 转换为空格
<code>-h</code>	将空格转换为 Tab（默认选项）

操作举例：

```
# 查看 /etc/protocols 中的不可见字符，可以看到很多 ^I，这其实就是 Tab 转义成可见字符的符号
$ cat -A /etc/protocols
# 使用 col -x 将 /etc/protocols 中的 Tab 转换为空格，然后再使用 cat 查看，你发现 ^I 不见了
$ cat /etc/protocols | col -x | cat -A
```

```

shiyanolou:~/ $ cat -A /etc/protocols | tail -n 5 [14:28:05]
manet^I138^I^I^I# MANET Protocols [RFC5498]$
hip^I139^IHIP^I^I# Host Identity Protocol$
shim6^I140^ISHim6^I^I# Shim6 Protocol [RFC5533]$
wesp^I141^IWESP^I^I# Wrapped Encapsulating Security Payload$
rohc^I142^IROHC^I^I# Robust Header Compression$
shiyanolou:~/ $ cat /etc/protocols | col -x | cat -A | tail -n 5 [14:28:09]
manet 138 # MANET Protocols [RFC5498]$
hip 139 HIP # Host Identity Protocol$
shim6 140 Shim6 # Shim6 Protocol [RFC5533]$
wesp 141 WESP # Wrapped Encapsulating Security Payload$
rohc 142 ROHC # Robust Header Compression$
shiyanolou:~/ $ [14:28:14]

```

join 命令用于将两个文件中包含相同内容的那一行合并在一起。使用方式: `join [option]... file1 file2`
常用的选项有:

选项	说明
-t	指定分隔符, 默认为空格
-i	忽略大小写的差异
-1	指明第一个文件要用哪个字段来对比, 默认对比第一个字段
-2	指明第二个文件要用哪个字段来对比, 默认对比第一个字段

操作举例:

```

$ cd /home/shiyanolou
# 创建两个文件
$ echo '1 hello' > file1
$ echo '1 shiyanolou' > file2
$ join file1 file2
# 将/etc/passwd与/etc/shadow两个文件合并, 指定以':'作为分隔符
$ sudo join -t':' /etc/passwd /etc/shadow
# 将/etc/passwd与/etc/group两个文件合并, 指定以':'作为分隔符, 分别对比第
4和第3个字段
$ sudo join -t':' -1 4 /etc/passwd -2 3 /etc/group

```

```

shiyanolou:~/ $ echo '1 hello' > file1 [14:31:01]
shiyanolou:~/ $ echo '1 shiyanolou' > file2 [14:31:04]
shiyanolou:~/ $ join file1 file2 [14:31:07]
1 hello shiyanolou
shiyanolou:~/ $ sudo join -t':' /etc/passwd /etc/shadow [14:31:10]
root:x:0:0:root:/root:/bin/bash:*:17590:0:99999:7:::
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin:*:17590:0:99999:7:::
bin:x:2:2:bin:/bin:/usr/sbin/nologin:*:17590:0:99999:7:::
sys:x:3:3:sys:/dev:/usr/sbin/nologin:*:17590:0:99999:7:::
sync:x:4:65534:sync:/bin:/bin/sync:*:17590:0:99999:7:::
games:x:5:60:games:/usr/games:/usr/sbin/nologin:*:17590:0:99999:7:::
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin:*:17590:0:99999:7:::
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin:*:17590:0:99999:7:::
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin:*:17590:0:99999:7:::
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin:*:17590:0:99999:7:::
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin:*:17590:0:99999:7:::

```

```
shiyanolou:~/$ sudo join -t ':' -1 4 /etc/passwd -2 3 /etc/group [14:32:33]
0:root:x:0:root:/root:/bin/bash:root:x:
1:daemon:x:1:daemon:/usr/sbin:/usr/sbin/nologin:daemon:x:
2:bin:x:2:bin:/bin:/usr/sbin/nologin:bin:x:
3:sys:x:3:sys:/dev:/usr/sbin/nologin:sys:x:
join: /etc/passwd:6: is not sorted: games:x:5:60:games:/usr/games:/usr/sbin/nologin
7:lp:x:7:lp:/var/spool/lpd:/usr/sbin/nologin:lp:x:
8:mail:x:8:mail:/var/mail:/usr/sbin/nologin:mail:x:
join: /etc/group:11: is not sorted: uucp:x:10:
9:news:x:9:news:/var/spool/news:/usr/sbin/nologin:news:x:
10:uucp:x:10:uucp:/var/spool/uucp:/usr/sbin/nologin:uucp:x:
13:proxy:x:13:proxy:/bin:/usr/sbin/nologin:proxy:x:
33:www-data:x:33:www-data:/var/www:/usr/sbin/nologin:www-data:x:
34:backup:x:34:backup:/var/backups:/usr/sbin/nologin:backup:x:
38:list:x:38:Mailing List Manager:/var/list:/usr/sbin/nologin:list:x:
39:irc:x:39:ircd:/var/run/ircd:/usr/sbin/nologin:irc:x:
41:gnats:x:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin:gnats:x:
65534:nobody:x:65534:nobody:/nonexistent:/usr/sbin/nologin:nogroup:x:
102:systemd-timesync:x:100:systemd Time Synchronization,,,:/run/systemd:/bin/false:systemd-timesync:x:
103:systemd-network:x:101:systemd Network Management,,,:/run/systemd/netif:/bin/false:systemd-network:x:
104:systemd-resolve:x:102:systemd Resolver,,,:/run/systemd/resolve:/bin/false:systemd-resolve:x:
105:systemd-bus-proxy:x:103:systemd Bus Proxy,,,:/run/systemd:/bin/false:systemd-bus-proxy:x:
shiyanolou:~/$
```

paste这个命令与join 命令类似，它是在不对比数据的情况下，简单地将多个文件合并一起，以Tab隔开。使用方式：paste [option] file...
常用的选项有：

选项	说明
-d	指定合并的分隔符，默认为 Tab
-s	不合并到一行，每个文件为一行

操作举例：

```
shiyanolou:~/$ echo hello > file1 [14:33:15]
shiyanolou:~/$ echo shiyanolou > file2 [14:33:22]
shiyanolou:~/$ echo www.shiyanolou.com > file3 [14:33:28]
shiyanolou:~/$ paste -d ':' file1 file2 file3 [14:33:35]
hello:shiyanolou:www.shiyanolou.com
shiyanolou:~/$ paste -s file1 file2 file3 [14:33:50]
hello
shiyanolou
www.shiyanolou.com
shiyanolou:~/$
```

数据流重定向：

两个重定向操作：>（清空再写入），>>（追加在后面）。注：<，<<除方向不同，大致相似。

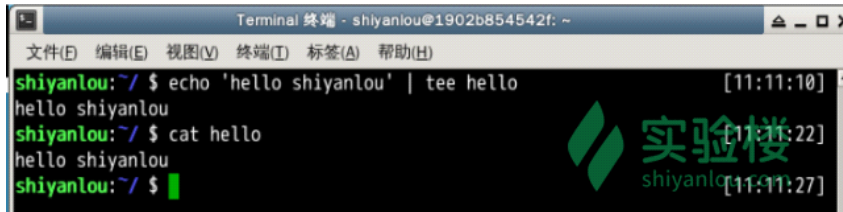
Linux 默认提供了三个特殊设备，用于终端的显示和输出，分别为stdin（标准输入, 对应于你在终端的输入），stdout（标准输出，对应于终端的输出），stderr（标准错误输出，对应于终端的输出）。它们对应的文件描述符：

文件描述符	设备文件	说明
0	/dev/stdin	标准输入
1	/dev/stdout	标准输出
2	/dev/stderr	标准错误

要隐藏某些错误或者警告时可以参照如下例子：

使用cat 命令同时读取两个文件，其中一个存在，另一个不存在，将标准错误重定向到标准输出，再将标准输出重定向到文件，注意要将重定向到文件写到前面：
`cat Documents/test.c hello.c >somefile 2>&1`。或者只用bash提供的特殊的重定向符号"&"将标准错误和标准输出同时重定向到文件：`cat Documents/test.c hello.c &>somefilehell`。

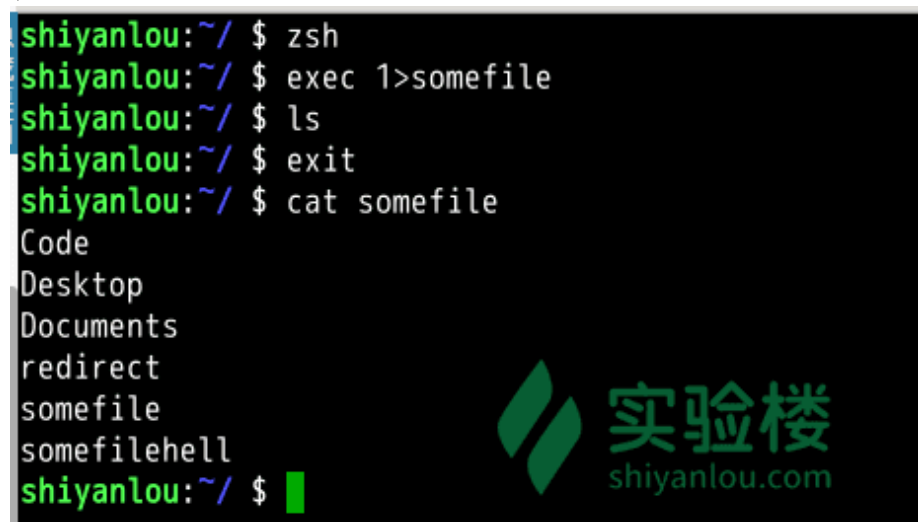
除了需要将输出重定向到文件，也需要将信息打印在终端时可以使用tee命令来实现：
`echo 'hello shiyanlou' | tee hello`。

A terminal window titled "Terminal 终端 - shiyanlou@1902b854542f: ~" showing the execution of the command `echo 'hello shiyanlou' | tee hello`. The output "hello shiyanlou" is displayed on the terminal and also written to the file "hello". Subsequent `cat hello` commands confirm the content. A green "实验楼" (Experiment Building) logo is visible in the background.

```
shiyanlou:~/ $ echo 'hello shiyanlou' | tee hello
hello shiyanlou
shiyanlou:~/ $ cat hello
hello shiyanlou
shiyanlou:~/ $
```

可以使用exec命令实现“永久”重定向。exec命令的作用是使用指定的命令替换当前的Shell，即使用一个进程替换当前进程，或者指定新的重定向：

```
# 先开启一个子 Shell
$ zsh
# 使用exec替换当前进程的重定向，将标准输出重定向到一个文件
$ exec 1>somefile
# 后面你执行的命令的输出都将被重定向到文件中, 直到你退出当前子shell, 或取消exec的重定向（后面将告诉你怎么做）
$ ls
$ exit
$ cat somefile
```

A terminal window showing the execution of `zsh`, `exec 1>somefile`, `ls`, `exit`, and `cat somefile`. The output of `ls` is redirected to the file "somefile". A green "实验楼" (Experiment Building) logo is visible in the background.

```
shiyanlou:~/ $ zsh
shiyanlou:~/ $ exec 1>somefile
shiyanlou:~/ $ ls
Code
Desktop
Documents
redirect
somefile
somefilehell
shiyanlou:~/ $
```

查看当前 Shell 进程中打开的文件描述符：`cd /dev/fd/;ls -Al`。

同样使用exec命令可以创建新的文件描述符：

```
$ zsh
$ exec 3>somefile
# 先进入目录，再查看，否则你可能不能得到正确的结果，然后再回到上一次的目录
$ cd /dev/fd/;ls -Al;cd -
# 注意下面的命令>与&之间不应该有空格，如果有空格则会出错
$ echo "this is test" >&3
$ cat somefile
$ exit
```



```
Terminal 终端 - shiyanlou@1902b854542f: ~
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

shiyanlou:~/$ zsh [11:12:07]
shiyanlou:~/$ exec 3>somefile [11:12:11]
shiyanlou:~/$ cd /dev/fd;ls -Al; cd - [11:12:25]
总用量 0
lrwx----- 1 shiyanlou shiyanlou 64 7月 24 11:12 0 -> /dev/pts/0
lrwx----- 1 shiyanlou shiyanlou 64 7月 24 11:12 1 -> /dev/pts/0
lrwx----- 1 shiyanlou shiyanlou 64 7月 24 11:12 10 -> /dev/pts/0
lr-x----- 1 shiyanlou shiyanlou 64 7月 24 11:12 12 -> /usr/share/zsh/functions
/Completion.zwc
lrwx----- 1 shiyanlou shiyanlou 64 7月 24 11:12 2 -> /dev/pts/0
l-wx----- 1 shiyanlou shiyanlou 64 7月 24 11:12 3 -> /home/shiyanlou/somefile
~
shiyanlou:~/$ echo "this is test" >&3 [11:12:37]
shiyanlou:~/$ cat somefile [11:12:54]
this is test
shiyanlou:~/$ exit [11:12:57]
shiyanlou:~/$ [11:13:02]
```

可以使用如下操作将打开的 3 号文件描述符关闭：

```
$ exec 3>&-
$ cd /dev/fd;ls -Al;cd -
```

在类 UNIX 系统中，/dev/null，或称空设备，是一个特殊的设备文件，它通常被用于丢弃不需要的输出流，或作为用于输入流的空文件，这些操作通常由重定向完成。读取它则会立即得到一个 EOF。可以利用/dev/null屏蔽命令的输出：`cat Documents/test.c 1>/dev/null 2>&1`。

xargs 是一条 UNIX 和类 UNIX 操作系统的常用命令。它的作用是将参数列表转换成小块分段传递给其他命令，以避免参数列表过长的问题。操作示例：`cut -d: -f1 < /etc/passwd | sort | xargs echo`。这个命令用于将/etc/passwd文件按:分割取第一个字段排序后，使用echo命令生成一个列表。

正则表达式基础：

正则表达式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（英语：Regular Expression，在代码中常简写为 regex、regexp 或 RE），计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。许多程序设计语言都支持利用正则表达式进行字符串操作。例如，在 Perl 中就内建了一个功能强大的正则表达式引擎。正则表达式这个概念最初是由 UNIX 中的工具软件（例如sed和grep）普及开的。正则表达式通常缩写成“regex”，单数有 regexp、regex，复数有 regexps、regexes、regexen。一个正则表达式通常被称为一个模式（pattern），为用来描述或者匹配一系列符合某个句法规则的字符串。

正则表达式有多种不同的风格，下面列举一些常用的作为 PCRE 子集的适用于perl和python编程语言及grep或egrep的正则表达式匹配规则：（注：PCRE（Perl Compatible Regular Expressions 中文含义：perl 语言兼容正则表达式）是一个用 C 语言编写的正则表达式函数库，由菲利普·海泽(Philip Hazel)编写。PCRE 是一个轻量级的函数库，比 Boost 之类的正则表达式库小得多。PCRE 十分易用，同时功能也很强大，性能超过了 POSIX 正则表达式库和一些经典的正则表达式库。）

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符。例如，“n”匹配字符“n”。“\n”匹配一个换行符。序列“\\”匹配“\”而“\(”则匹配“(”。
^	匹配输入字符串的开始位置。
\$	匹配输入字符串的结束位置。

{n}	n 是一个非负整数。匹配确定的 n 次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个 o。
{n,}	n 是一个非负整数。至少匹配 n 次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooooo”中的所有 o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。
{n,m}	m 和 n 均为非负整数，其中 n<=m。最少匹配 n 次且最多匹配 m 次。例如，“o{1,3}”将匹配“foooooo”中的前三个 o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。
*	匹配前面的子表达式零次或多次。例如，zo*能匹配“z”、“zo”以及“zoo”。*等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。
?	当该字符紧跟在任何一个其他限制符(*,+,?,{n},{n,},{n,m})后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”。
.	匹配除“\n”之外的任何单个字符。要匹配包括“\n”在内的任何字符，请使用像“(. \n)”的模式。
(pattern)	匹配 pattern 并获取这一匹配的子字符串。该子字符串用于向后引用。要匹配圆括号字符，请使用“\("或“\)”。
x y	匹配 x 或 y。例如，“z food”能匹配“z”或“food”。“(z f)ood”则匹配“zood”或“food”。
[xyz]	字符集合（character class）。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。其中特殊字符仅有反斜线\保持特殊含义，用于转义字符。其它特殊字符如星号、加号、各种括号等均作为普通字符。脱字符^如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。连字符-如果出现在字符串中间表示字符范围描述；如果出现在首位则仅作为普通字符。
[^xyz]	排除型（negate）字符集合。匹配未列出的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。
[^a-z]	排除型的字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。

优先级为从上到下从左到右，依次降低：

运算符	说明
\	转义符
(), (?:), (?!), []	括号和中括号


```
Terminal 终端 - shiyanlou@4abda5e9cedb: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

shiyanlou:~/ $ cat test
shiyanlou
www.shiyanlou.com
C
linux
dat_struct
python
Mysql
shell
shiyanlou:~/ $ grep -c shiyanlou test
2
shiyanlou:~/ $ grep -i -n c test
2:www.shiyanlou.com
3:C
5:dat_struct
shiyanlou:~/ $ grep -v shell test
shiyanlou
www.shiyanlou.com
C
linux
dat_struct
python
Mysql
shiyanlou:~/ $ grep shell test
shell
shiyanlou:~/ $
```

使用基本正则表达式，BRE:

```
# 将匹配以'z'开头以'o'结尾的所有字符串
$ echo 'zero\nzo\nzoo' | grep 'z.*o'
# 将匹配以'z'开头以'o'结尾，中间包含一个任意字符的字符串
$ echo 'zero\nzo\nzoo' | grep 'z.o'
# 将匹配以'z'开头，以任意多个'o'结尾的字符串
$ echo 'zero\nzo\nzoo' | grep 'zo*'
```

```
Terminal 终端 - shiyanlou@4abda5e9cedb: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

shiyanlou:~/ $ echo 'zero\nzo\nzoo' | grep 'z.*o'
zero
zo
zoo
shiyanlou:~/ $ echo 'zero\nzo\nzoo' | grep 'z.o'
zoo
shiyanlou:~/ $ echo 'zero\nzo\nzoo' | grep 'zo*'
zero
zo
zoo
shiyanlou:~/ $
```

```
# grep默认是区分大小写的，这里将匹配所有的小写字母
$ echo '1234\nabcd' | grep '[a-z]'
# 将匹配所有的数字
$ echo '1234\nabcd' | grep '[0-9]'
# 将匹配所有的数字
$ echo '1234\nabcd' | grep '[:digit:]'
# 将匹配所有的小写字母
$ echo '1234\nabcd' | grep '[:lower:]'
# 将匹配所有的大写字母
$ echo '1234\nabcd' | grep '[:upper:]'
# 将匹配所有的字母和数字，包括0-9, a-z, A-Z
$ echo '1234\nabcd' | grep '[:alnum:]'
# 将匹配所有的字母
```

```
$ echo '1234\nabcd' | grep '[:alpha:]'
```

```
shiyanlou:~/ $ echo '1234\nabcd' | grep '[a-z]' [11:27:02]
abcd
shiyanlou:~/ $ echo '1234\nabcd' | grep '[0-9]' [11:27:07]
1234
shiyanlou:~/ $ echo '1234\nabcd' | grep '[:digit:]' [11:27:13]
1234
shiyanlou:~/ $ echo '1234\nabcd' | grep '[:lower:]' [11:27:26]
abcd
shiyanlou:~/ $ echo '1234\nabcd' | grep '[:upper:]' [11:27:33]
shiyanlou:~/ $ echo '1234\nabcd\nABCD' | grep '[:upper:]' [11:27:38]
ABCD
shiyanlou:~/ $ echo '1234\nabcd\nABCD' | grep '[:alnum:]' [11:27:58]
1234
abcd
ABCD
shiyanlou:~/ $ echo '1234\nabcd\nABCD' | grep '[:alpha:]' [11:28:10]
abcd
ABCD
shiyanlou:~/ $
```

下面包含完整的特殊符号及说明：

特殊符号	说明
<code>[:alnum:]</code>	代表英文大小写字母及数字，亦即 0-9，A-Z，a-z
<code>[:alpha:]</code>	代表任何英文大小写字母，亦即 A-Z，a-z
<code>[:blank:]</code>	代表空白键与 [Tab] 按键两者
<code>[:cntrl:]</code>	代表键盘上面的控制按键，亦即包括 CR，LF，Tab，Del... 等等
<code>[:digit:]</code>	代表数字而已，亦即 0-9
<code>[:graph:]</code>	除了空白字节（空白键与 [Tab] 按键）外的其他所有按键
<code>[:lower:]</code>	代表小写字母，亦即 a-z
<code>[:print:]</code>	代表任何可以被列印出来的字符
<code>[:punct:]</code>	代表标点符号（punctuation symbol），亦即： " ' ? ! ; : # \$...
<code>[:upper:]</code>	代表大写字母，亦即 A-Z
<code>[:space:]</code>	任何会产生空白的字符，包括空白键，[Tab]，CR 等等
<code>[:xdigit:]</code>	代表 16 进位的数字类型，因此包括： 0-9，A-F，a-f 的数字与字节

注：之所以要使用特殊符号，是因为上面的[a-z]不是在所有情况下都管用，这还与主机当前的语系有关，即设置在LANG环境变量的值，zh_CN.UTF-8 的话[a-z]，即为所有小写字母，其它语系可能是大小写交替的如，“a A b B... z Z”，[a-z]中就可能包含大写字母。所以在使用[a-z]时请确保当前语系的影响，使用[:lower:]则不会有这个问题。

使用扩展正则表达式，ERE：

```
# 只匹配"zo"
$ echo 'zero\nzo\nzoo' | grep -E 'zo{1}'
# 匹配以"zo"开头的所有单词
$ echo 'zero\nzo\nzoo' | grep -E 'zo{1,}'
```

sed工具在 man 手册里面的全名为“sed - stream editor for filtering and transforming text ”，意即，用于过滤和转换文本的流编辑器。

sed 命令基本格式：

```
sed [参数]... [执行命令] [输入文件]...  
# 形如：  
$ sed -i 's/sad/happy/' test # 表示将test文件中的“sad”替换为“happy”
```

参数	说明
-n	安静模式，只打印受影响的行，默认打印输入数据的全部内容
-e	用于在脚本中添加多个执行命令一次执行，在命令行中执行多个命令通常不需要加该参数
-f filename	指定执行 filename 文件中的命令
-r	使用扩展正则表达式，默认为标准正则表达式
-i	将直接修改输入文件内容，而不是打印到标准输出设备

sed 执行命令格式：

```
[n1][,n2]command  
[n1][~step]command  
# 其中一些命令可以在后面加上作用范围，形如：  
$ sed -i 's/sad/happy/g' test # g表示全局范围  
$ sed -i 's/sad/happy/4' test # 4表示指定行中的第四个匹配字符串
```

注：其中 n1,n2 表示输入内容的行号，它们之间为逗号则表示从 n1 到 n2 行，如果为~波浪号则表示从 n1 开始以 step 为步进的所有行；command 为执行动作，下面为一些常用动作指令：

命令	说明
s	行内替换
c	整行替换
a	插入到指定行的后面
i	插入到指定行的前面
p	打印指定行，通常与-n参数配合使用
d	删除指定行

示例：

```
Terminal 终端 - shiyanlou@4abda5e9cedb: ~
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ cp /etc/passwd ~ [11:33:30]
shiyanlou:~/ $ nl passwd | sed -n '2,5p' [11:33:37]
 2 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
 3 bin:x:2:2:bin:/bin:/usr/sbin/nologin
 4 sys:x:3:3:sys:/dev:/usr/sbin/nologin
 5 sync:x:4:65534:sync:/bin:/bin/sync
shiyanlou:~/ $ nl passwd | sed -n '1~2p' [11:33:55]
 1 root:x:0:0:root:/root:/bin/bash
 3 bin:x:2:2:bin:/bin:/usr/sbin/nologin
 5 sync:x:4:65534:sync:/bin:/bin/sync
 7 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
 9 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
11 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
13 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
15 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
17 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbi
n/nologin
19 systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:
/bin/false
21 systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/
false
23 _apt:x:104:65534::/nonexistent:/bin/false
25 colord:x:106:112:colord colour management daemon,,,:/var/lib/colord:/bin
/false
27 pulse:x:108:113:PulseAudio daemon,,,:/var/run/pulse:/bin/false
29 usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
31 labex:x:6000:6000::/home/labex:/usr/bin/zsh
33 mongodb:x:112:65534::/home/mongodb:/bin/false
shiyanlou:~/ $ [11:34:17]
```

AWK是一种优良的文本处理工具，Linux 及 Unix 环境中现有的功能最强大的数据处理引擎之一。其名称得自于它的创始人 Alfred Aho（阿尔弗雷德·艾侯）、Peter Jay Weinberger（彼得·温伯格）和 Brian Wilson Kernighan（布莱恩·柯林汉）姓氏的首个字母。AWK 程序设计语言，三位创建者已将它正式定义为“样式扫描和处理语言”。它允许您创建简短的程序，这些程序读取输入文件、为数据排序、处理数据、对输入执行计算以及生成报表，还有无数其他的功能。最简单地讲，AWK 是一种用于处理文本的编程语言工具。awk 所有的操作都是基于 pattern(模式)—action(动作)对来完成的，如下面的形式：

```
$ pattern {action}
```

其中 pattern 通常是表示用于匹配输入的文本的“关系式”或“正则表达式”，action 则是表示匹配后将执行的动作。在一个完整 awk 操作中，这两者可以只有其中一个，如果没有 pattern 则默认匹配输入的全部文本，如果没有 action 则默认为打印匹配内容到屏幕。awk处理文本的方式，是将文本分割成一些“字段”，然后再对这些字段进行处理，默认情况下，awk 以空格作为一个字段的分割符，不过这不是固定的。

awk命令基本格式：

```
awk [-F fs] [-v var=value] [-f prog-file | 'program text'] [file...]
```

其中-F参数用于预先指定前面提到的字段分隔符（还有其他指定字段的方式），-v用于预先为awk程序指定变量，-f参数用于指定awk命令要执行的程序文件，或者在不加-f参数的情况下直接将程序语句放在这里，最后为awk需要处理的文本输入，且可以同时输入多个文本文件。

示例：



```
Terminal 终端 - shiyanlou@5a97d9a0cf02: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

shiyanlou:~/ $ cat test
I like linux
www.shiyanlou.com
shiyanlou:~/ $ awk '{
quote> print
quote> }' test
I like linux
www.shiyanlou.com
shiyanlou:~/ $
```

将 test 的第一行的每个字段单独显示为一行:

```
$ awk '{
> if(NR==1){
> print $1 "\n" $2 "\n" $3
> } else {
> print}
> }' test
```

或者

```
$ awk '{
> if(NR==1){
> OFS="\n"
> print $1, $2, $3
> } else {
> print}
> }' test
```



```
Terminal 终端 - shiyanlou@5a97d9a0cf02: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

shiyanlou:~/ $ awk '{
quote> if(NR==1){
quote> print $1 "\n" $2 "\n" $3
quote> } else {
quote> print}
quote> }' test
I
like
linux
www.shiyanlou.com
shiyanlou:~/ $
```

awk语言的分支选择语句if, 它的使用和很多高级语言如C/C++语言基本一致, NR与OFS这两个是awk内建的变量, NR表示当前读入的记录数, 可以简单的理解为当前处理的行数, OFS表示输出时的字段分隔符, 默认为" "空格, 如上图所见, 将字段分隔符设置为\n换行符, 所以第一行原本以空格为字段分隔的内容就分别输出到单独一行了。然后是\$N其中 N 为相应的字段号, 这也是awk的内建变量, 它表示引用相应的字段, 因为我们这里第一行只有三个字段, 所以只引用到了\$3。除此之外另一个这里没有出现的\$0, 它表示引用当前记录(当前行)的全部内容。

将 test 的第二行的以点为分段的字段换成以空格为分隔:

```
$ awk -F'.' '{
> if(NR==2){
> print $1 "\t" $2 "\t" $3
> }}' test
```

或者

```
$ awk '
> BEGIN{
> FS="."
> }
```

```
> OFS="\t" # 如果写为一行，两个动作语句之间应该以";"号分开
> }{
> if(NR==2){
> print $1, $2, $3
> }}' test
```

```
Terminal 终端 - shiyanlou@5a97d9a0cf02: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ awk '
quote> BEGIN{
quote> FS="."
quote> OFS="\t"
quote> }{
quote> print $1,$2,$3
quote> }' test
I like linux
www    shiyanlou    com
shiyanlou:~/ $
```

这里的-F参数，前面已经介绍过，它是用来预先指定待处理记录的字段分隔符。需要注意的是除了指定OFS我们还可以在print 语句中直接打印特殊符号如这里的 \t, print 打印的非变量内容都需要用""一对引号包围起来。上面另一个版本，展示了实现预先指定变量分隔符的另一种方式，即使用BEGIN，就这个表达式指示了，其后的动作将在所有动作之前执行，这里是FS赋值了新的"."点号代替默认的" "空格。

awk 常用的内置变量：

变量名	说明
FILENAME	当前输入文件名，若有多个文件，则只表示第一个。如果输入是来自标准输入，则为空字符串
\$0	当前记录的内容
\$N	N 表示字段号，最大值为NF变量的值
FS	字段分隔符，由正则表达式表示，默认为" "空格
RS	输入记录分隔符，默认为"\n"，即一行为一个记录
NF	当前记录字段数
NR	已经读入的记录数
FNR	当前输入文件的记录数，请注意它与 NR 的区别
OFS	输出字段分隔符，默认为" "空格
ORS	输出记录分隔符，默认为"\n"

软件安装：

通常 Linux 上的软件安装主要有四种方式：

1. 在线安装
2. 从磁盘安装 deb 软件包
3. 从二进制软件包安装
4. 从源代码编译安装

APT 是 Advance Packaging Tool（高级包装工具）的缩写，是 Debian 及其派生发行版的软件包管理器，APT 可以自动下载，配置，安装二进制或者源代码格式的软件包，因此简化了 Unix 系统上管理软件的过程。APT 最早被设计成 dpkg 的前端，用来处理 deb 格式的软件包。现在经过 APT-RPM 组织修改，APT 已经可以安装在支持 RPM 的系统管理 RPM 包。这个包管理器包含以 apt- 开头的多个工具，如 apt-get apt-cache apt-cdrom 等，在 Debian 系列的发行版中使用。

当在执行安装操作时，首先apt-get 工具会在本地的一个数据库中搜索关于软件的相关

信息，并根据这些信息在相关的服务器上下载软件安装，我们需要定期从服务器上下载一个软件包列表，使用 `sudo apt-get update` 命令来保持本地的软件包列表是最新的（有时也需要手动执行这个操作，比如更换了软件源），而这个表里会有软件依赖信息的记录，对于软件依赖，举个例子：安装 `w3m` 软件的时候，而这个软件需要 `libgc1c2` 这个软件包才能正常工作，这个时候 `apt-get` 在安装软件的时候会一并替我们安装了，以保证 `w3m` 能正常的工作。

`apt-get` 是用于处理 `apt`包的公用程序集，我们可以用它来在线安装、卸载和升级软件包等，下面列出一些`apt-get`包含的常用的一些工具：

工具	说明
<code>install</code>	其后加上软件包名，用于安装一个软件包
<code>update</code>	从软件源镜像服务器上下载/更新用于更新本地软件源的软件包列表
<code>upgrade</code>	升级本地可更新的全部软件包，但存在依赖问题时将不会升级，通常会在更新之前执行一次 <code>update</code>
<code>dist-upgrade</code>	解决依赖关系并升级(存在一定危险性)
<code>remove</code>	移除已安装的软件包，包括与被移除软件包有依赖关系的软件包，但不包含软件包的配置文件
<code>autoremove</code>	移除之前被其他软件包依赖，但现在不再被使用的软件包
<code>purge</code>	与 <code>remove</code> 相同，但会完全移除软件包，包含其配置文件
<code>clean</code>	移除下载到本地的已经安装的软件包，默认保存在 <code>/var/cache/apt/archives/</code>
<code>autoclean</code>	移除已安装的软件的旧版本软件包

下面是一些`apt-get`常用的参数：

参数	说明
<code>-y</code>	自动回应是否安装软件包的选项，在一些自动化安装脚本中使用这个参数将十分有用
<code>-s</code>	模拟安装
<code>-q</code>	静默安装方式，指定多个 <code>q</code> 或者 <code>-q=#</code> , <code>#</code> 表示数字，用于设定静默级别，这在你不想要在安装软件包时屏幕输出过多时很有用
<code>-f</code>	修复损坏的依赖关系
<code>-d</code>	只下载不安装
<code>--reinstall</code>	重新安装已经安装但可能存在问题的软件包
<code>--install-suggests</code>	同时安装 APT 给出的建议安装的软件包

升级三连：

```
# 更新软件源
$ sudo apt-get update
# 升级没有依赖问题的软件包
$ sudo apt-get upgrade
# 升级并解决依赖关系
$ sudo apt-get dist-upgrade
```

软件搜索命令apt-cache则是针对本地数据进行相关操作的工具，search 顾名思义在本地的数据库中寻找有关 softname1 softname2 相关软件的信息，其使用格式为：

```
sudo apt-cache search softname1 softname2 softname3.....
```

dpkg 是 Debian 软件包管理器的基础，它被伊恩·默多克创建于 1993 年。dpkg 与 RPM 十分相似，同样被用于安装、卸载和供给和 .deb 软件包相关的信息。dpkg 本身是一个底层的工具。上层的工具，像是 APT，被用于从远程获取软件包以及处理复杂的软件包关系。“dpkg”是“Debian Package”的简写。

dpkg常用参数介绍：

参数	说明
-i	安装指定 deb 包
-R	后面加上目录名，用于安装该目录下的所有 deb 安装包
-r	remove，移除某个已安装的软件包
-I	显示deb包文件的信息
-s	显示已安装软件的信息
-S	搜索已安装的软件包
-L	显示已安装软件包的目录信息

进程：

每一个进程都会是一个进程组的成员，而且这个进程组是唯一存在的，他们是依靠 PGID（process group ID）来区别的，而每当一个进程被创建的时候，它便会成为其父进程所在组中的一员。一般情况，进程组的 PGID 等同于进程组的第一个成员的 PID，并且这样的进程称为该进程组的领导者，也就是领导进程，进程一般通过使用 getpgrp() 系统调用来寻找其所在组的 PGID，领导进程可以先终结，此时进程组依然存在，并持有相同的 PGID，直到进程组中最后一个进程终结。与进程组类似，每当一个进程被创建的时候，它便会成为其父进程所在 Session 中的一员，每一个进程组都会在一个 Session 中，并且这个 Session 是唯一存在的，


Session 主要是针对一个 tty 建立，Session 中的每个进程都称为一个工作(job)。每个会话可以连接一个终端(control terminal)。当控制终端有输入输出时，都传递给该会话的前台进程组。Session 意义在于将多个 jobs 囊括在一个终端，并取其中的一个 job 作为前台，来直接接收该终端的输入输出以及终端信号。其他 jobs 在后台运行。前台（foreground）就是在终端中运行，能与你交互的，后台（background）就是在终端中运行，但是你不能与其任何的交互，也不会显示其执行的过程。

bash(Bourne-Again shell)支持工作控制（job control），而 sh（Bourne shell）并不支持。并且每个终端或者说 bash 只能管理当前终端中的 job，不能管理其他终端中的 job。

可以通过 & 这个符号，让命令在后台中运行：

```
ls &
```


```
shiyanolou:~/ $ ll & [1:32:13]
[1] 236
total 8.0K
drwxrwxr-x 2 shiyanolou shiyanolou 4.0K Nov 27 2015 Code
drwxrwxr-x 2 shiyanolou shiyanolou 4.0K Nov 27 2015 Desktop
[1] + 236 done      ls --color=tty -lh
shiyanolou:~/ $ [1:32:24]
```



图中所显示的 [1] 236 分别是该 job 的 job number 与该进程的 PID，而最后一行的 Done 表示该命令已经在后台执行完毕。

还可以通过 `ctrl + z` 使当前工作停止并丢到后台中去：

```
shiyanolou:~/ $ tail -f /var/log/dpkg.log [1:44:51]
^Z
[2] + 316 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ [1:44:59]
```



被停止并放置在后台的工作可以使用这个命令来查看：

`jobs`

```
shiyanolou:~/ $ tail -f /var/log/dpkg.log [1:44:51]
^Z
[2] + 316 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ jobs [1:44:59]
[1] - suspended tail -f /var/log/dpkg.log
[2] + suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ tail -f /var/log/dpkg.log [1:46:28]
^Z
[3] + 325 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ jobs [1:51:04]
[1] suspended tail -f /var/log/dpkg.log
[2] - suspended tail -f /var/log/dpkg.log
[3] + suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ [1:51:07]
```

其中第一列显示的为被放置后台 job 的编号，而第二列的 + 表示最近(刚刚、最后)被放置后台的 job，同时也表示预设的工作，也就是若是有什么针对后台 job 的操作，首先对预设的 job，- 表示倒数第二（也就是在预设之前的一个）被放置后台的工作，倒数第三个（再之前的）以后都不会有这样的符号修饰，第三列表示它们的状态，而最后一列表示该进程执行的命令。

可以通过这样的一个命令将后台的工作拿到前台来：

#后面不加参数提取预设工作，加参数提取指定工作的编号


#ubuntu 在 zsh 中需要 %，在 bash 中不需要 %

`fg [%jobnumber]`

```
shiyanolou:~/ $ tail -f /var/log/dpkg.log [1:44:51]
^Z
[2] + 316 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ jobs [1:44:59]
[1] - suspended tail -f /var/log/dpkg.log
[2] + suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ tail -f /var/log/dpkg.log [1:46:28]
^Z
[3] + 325 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ jobs [1:51:04]
[1] suspended tail -f /var/log/dpkg.log
[2] - suspended tail -f /var/log/dpkg.log
[3] + suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ [1:51:07]
```



```
shiyanolou:~/ $ fg %2 [2:00:21]
[2] 316 continued tail -f /var/log/dpkg.log
```




若是想让其后台运作就使用这样一个命令：

#与fg类似，加参则指定，不加参则取预设

`bg [%jobnumber]`

```
shiyanolou:~/ $ fg %2 [2:00:21]
[2] 316 continued tail -f /var/log/dpkg.log

^Z
[2] + 316 suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ [2:02:17]
shiyanolou:~/ $ jobs [2:02:20]
[1] suspended tail -f /var/log/dpkg.log
[2] + suspended tail -f /var/log/dpkg.log
[3] - suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ bg %2 [2:02:22]
[2] 316 continued tail -f /var/log/dpkg.log
shiyanolou:~/ $ [2:02:40]
shiyanolou:~/ $ jobs [2:02:44]
[1] - suspended tail -f /var/log/dpkg.log
[2] running tail -f /var/log/dpkg.log
[3] + suspended tail -f /var/log/dpkg.log
shiyanolou:~/ $ [2:02:46]
shiyanolou:~/ $ bg [2:04:31]
[3] - 325 continued tail -f /var/log/dpkg.log
shiyanolou:~/ $ jobs [2:04:37]
[1] + suspended tail -f /var/log/dpkg.log
[2] running tail -f /var/log/dpkg.log
[3] - running tail -f /var/log/dpkg.log
shiyanolou:~/ $ [2:04:42]
```



既然有方法将被放置在后台的工作提至前台或者让它从停止变成继续运行在后台，当然也有方法删除一个工作，或者重启等等：

#kill的使用格式如下

`kill -signal %jobnumber`

#signal从1-64个信号值可以选择，可以这样查看

`kill -l`

其中常用的有这些信号值：

信号值	作用
-1	重新读取参数运行，类似与 restart

-2	如同 ctrl+c 的操作退出
-9	强制终止该任务
-15	正常的方式终止该任务

若是在使用 kill + 信号值然后直接加 pid，你将会对 pid 对应的进程进行操作，若是在使用 kill+信号值然后 %jobnumber，这时所操作的对象是 job，这个数字就是就当前 bash 中后台的运行的 job 的 ID。

可以通过 top 实时的查看进程的状态，以及系统的一些信息（如 CPU、内存信息等），还可以通过 ps 来静态查看当前的进程信息，同时还可以使用 pstree 来查看当前活跃进程的树形结构。

top示例：

```
top - 11:05:18 up 8 days, 17:12, 1 user, load average: 0.29, 0.20, 0.25
Tasks: 26 total, 1 running, 25 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 1.0 sy, 0.0 ni, 97.9 id, 0.0 wa, 0.1 hi, 0.0 si, 0.0
KiB Mem: 8176740 total, 8032104 used, 144636 free, 313088 buffers
KiB Swap: 0 total, 0 used, 0 free. 2704284 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0   21896   1628   1356  S   0.0   0.0   0:00.02 init.sh
   31 shiyanl+   20   0  195140  26664   9484  S   0.0   0.3   0:00.21 Xvnc
   40 shiyanl+   20   0   30664   1800   1452  S   0.0   0.0   0:00.00 vncconfig
   44 shiyanl+   20   0    4440    684    576  S   0.0   0.0   0:00.00 sh
   46 root        20   0   50512   1524   1152  S   0.0   0.0   0:00.00 su
   47 root        20   0   56844  14888   3640  S   0.0   0.2   0:00.14 supervis+
   48 shiyanl+   20   0  927476  45916   6016  S   0.0   0.6   0:01.07 node
   60 shiyanl+   20   0  168616   7456   6056  S   0.0   0.1   0:00.05 xfce4-se+
   67 shiyanl+   20   0   24436    596    340  S   0.0   0.0   0:00.00 dbus-lau+
   69 shiyanl+   20   0   39244   1168    776  S   0.0   0.0   0:00.02 dbus-dae+
   74 shiyanl+   20   0   39448   2468   2020  S   0.0   0.0   0:00.01 xfconfd
   78 shiyanl+   20   0   10616    316     0  S   0.0   0.0   0:00.00 ssh-agent
   80 shiyanl+   20   0  176920  11588   8224  S   0.0   0.1   0:00.80 xfw4
   84 shiyanl+   20   0  269660  12300   9052  S   0.0   0.2   0:00.13 xfce4-pa+
   86 shiyanl+   20   0  168572   6588   5244  S   0.0   0.1   0:00.01 Thunar
   88 shiyanl+   20   0  437112  17220  10808  S   0.0   0.2   0:00.22 xfdesktop
   89 root        20   0   61368   3064   2384  S   0.0   0.0   0:00.00 sshd
   95 shiyanl+   20   0  245068   8224   5052  S   0.0   0.1   0:00.68 xfsettin+
  105 shiyanl+   20   0   46452    964    456  S   0.0   0.0   0:00.00 scim-hel+
  106 shiyanl+   20   0  257548   8092   4444  S   0.0   0.1   0:00.01 scim-pan+
  108 shiyanl+   20   0   61080   8760    220  S   0.0   0.1   0:00.00 scim-tau+
  157 shiyanl+   20   0  149188   7332   5864  S   0.0   0.1   0:00.02 panel-6-+
```

top 显示的第一排：

内容	解释
top	表示当前程序的名称
11:05:18	表示当前的系统的时间
up 8 days, 17:12	表示该机器已经启动了多长时间
1 user	表示当前系统中只有一个用户
load average: 0.29, 0.20, 0.25	分别对应 1、5、15 分钟内 cpu 的平均负载

实际生活中需要将得到的load average 数据值除以CPU核数来看。可以通过以下的命令来查看 CPU 的个数与核心数：

#查看物理CPU的个数

#cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l

#每个cpu的核心数

cat /proc/cpuinfo |grep "physical id"|grep "0"|wc -l

在实际生活中，比较有经验的运维或者系统管理员会将临界值定为 0.7。这里的指数都是除以核心数以后的值：

若是 load < 0.7 并不会去关注它；

若是 0.7 < load < 1 的时候我们就需要稍微关注一下了，虽然还可以应付但是这

个值已经离临界不远了；

若是 $\text{load} = 1$ 的时候就需要警惕了，因为这个时候已经没有更多的资源了，已经在全力以赴了；

若是 $\text{load} > 5$ 的时候系统已经快不行了，这个时候就需要加班解决问题了

top 的第二行数据基本上是进程的一个情况统计：

内容	解释
Tasks: 26 total	进程总数
1 running	1 个正在运行的进程数
25 sleeping	25 个睡眠的进程数
0 stopped	没有停止的进程数
0 zombie	没有僵尸进程数

top 的第三行数据基本上是 CPU 的一个使用情况的统计了：

内容	解释
Cpu(s): 1.0%us	用户空间进程占用 CPU 百分比
1.0% sy	内核空间运行占用 CPU 百分比
0.0%ni	用户进程空间内改变过优先级的进程占用 CPU 百分比
97.9%id	空闲 CPU 百分比
0.0%wa	等待输入输出的 CPU 时间百分比
0.1%hi	硬中断(Hardware IRQ)占用 CPU 的百分比
0.0%si	软中断(Software IRQ)占用 CPU 的百分比
0.0%st	(Steal time) 是 hypervisor 等虚拟服务中，虚拟 CPU 等待实际 CPU 的时间的百分比

top 的第四行数据基本上是内存的一个使用情况的统计了：

内容	解释
8176740 total	物理内存总量
8032104 used	使用的物理内存总量
144636 free	空闲内存总量
313088 buffers	用作内核缓存的内存量

注：系统中可用的物理内存最大值并不是 free 这个单一的值，而是 free + buffers + swap 中的 cached 的和。

top 的第五行数据基本上是交换区的一个使用情况的统计了：

内容	解释
total	交换区总量
used	使用的交换区总量
free	空闲交换区总量
cached	缓冲的交换区总量, 内存中的内容被换出到交换区，而后又被换入到内存，但使用过的交换区尚未被覆盖

再下面就是进程的情况了：

列名	解释
PID	进程 id

USER	该进程的所属用户
PR	该进程执行的优先级 priority 值
NI	该进程的 nice 值
VIRT	该进程任务所使用的虚拟内存的总数
RES	该进程所使用的物理内存数，也称之为驻留内存数
SHR	该进程共享内存的大小
S	该进程进程的状态：S=sleep R=running Z=zombie
%CPU	该进程 CPU 的利用率
%MEM	该进程内存的利用率
TIME+	该进程活跃的总时间
COMMAND	该进程运行的名字

注：NICE 值叫做静态优先级，是用户空间的一个优先级值，其取值范围是-20 至 19。这个值越小，表示进程”优先级”越高，而值越大“优先级”越低。nice 值中的 -20 到 19，中 -20 优先级最高，0 是默认的值，而 19 优先级最低，PR 值表示 Priority 值叫动态优先级，是进程在内核中实际的优先级值，进程优先级的取值范围是通过一个宏定义的，这个宏的名称是 MAX_PRIO，它的值为 140。Linux 实际上实现了 140 个优先级范围，取值范围是从 0-139，这个值越小，优先级越高。而这其中的 0 - 99 是实时进程的值，而 100 - 139 是给用户的。其中 PR 中的 100 to 139 值部分有这么一个对应 $PR = 20 + (-20 \text{ to } +19)$ ，这里的 -20 to +19 便是 nice 值，所以说两个虽然都是优先级，而且有千丝万缕的关系，但是他们的值，他们的作用范围并不相同，** VIRT **任务所使用的虚拟内存的总数，其中包含所有的代码，数据，共享库和被换出 swap 空间的页面等所占据空间的总数。

top常用交互命令：

常用交互命令	解释
q	退出程序
I	切换显示平均负载和启动时间的信息
P	根据 CPU 使用百分比大小进行排序
M	根据驻留内存大小进行排序
i	忽略闲置和僵死的进程，这是一个开关式命令
k	终止一个进程，系统提示输入 PID 及发送的信号值。一般终止进程用 15 信号，不能正常结束则使用 9 信号。安全模式下该命令被屏蔽。

ps示例：

```
ps aux
```

```

shiyanolou:~/ $ ps aux [15:29:58]
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   21896 1632 ?        Ss   15:29   0:00 /bin/bash /etc/shiyanolou
shiyanol+  35  0.6  0.2 193288 24524 ?        Sl   15:29   0:00 /usr/bin/Xvnc :1 -des
shiyanol+  44  0.0  0.0   30664 1792 ?        S    15:29   0:00 /usr/bin/vncconfig -n
shiyanol+  48  0.0  0.0    4440   656 ?        S    15:29   0:00 /bin/sh /etc/xdg/xfce4
root       50  0.0  0.0    50512 1508 ?        S    15:29   0:00 su - shiyanolou
root       51  0.5  0.1   56844 14884 ?        S    15:29   0:00 /usr/bin/python /usr/
shiyanol+  54  5.2  0.5 927476 44692 ?        Ssl  15:29   0:01 node /usr/local/b
shiyanol+  64  0.2  0.0 168612 7464 ?        S    15:29   0:00 xfce4-session
shiyanol+  67  0.0  0.0    24436 596 ?        S    15:29   0:00 /usr/bin/dbus-launch
shiyanol+  68  0.1  0.0   39232 1176 ?        Ss   15:29   0:00 //bin/dbus-daemon --f
shiyanol+  73  0.0  0.0   39448 2496 ?        S    15:29   0:00 /usr/lib/x86_64-linux
shiyanol+  82  0.0  0.0    10616 312 ?        Ss   15:29   0:00 /usr/bin/ssh-agent -s
shiyanol+  84  3.7  0.1 176916 11528 ?        S    15:29   0:01 xfwm4
shiyanol+  88  0.6  0.1 269584 12320 ?        Sl   15:29   0:00 xfce4-panel
shiyanol+  90  0.0  0.0 168572 6592 ?        S    15:29   0:00 Thunar --daemon
root      92  0.0  0.0   61368 3072 ?        S    15:29   0:00 /usr/sbin/sshd -D
shiyanol+  93  0.9  0.2 437012 17196 ?        Sl   15:29   0:00 xfdesktop
shiyanol+  96  3.4  0.1 245408 8572 ?        Ssl  15:29   0:00 xfsettingsd
root      98  0.1  0.0   95096 3904 ?        Ss   15:29   0:00 sshd: shiyanolou
shiyanol+ 112  0.0  0.0   95096 1820 ?        R    15:29   0:00 sshd: shiyanolou
shiyanol+ 113  3.4  0.0   39304 3780 pts/0    Ss   15:29   0:00 -zsh
shiyanol+ 137  0.1  0.0 149188 7288 ?        S    15:29   0:00 /usr/lib/x86_64-linux
shiyanol+ 140  0.0  0.0   46452   956 ?        Ss   15:29   0:00 /usr/lib/x86_64-linux
shiyanol+ 141  0.0  0.1 257888 8416 ?        Ssl  15:29   0:00 /usr/lib/x86_64-linux
shiyanol+ 143  0.0  0.1   61080 8760 ?        Ss   15:29   0:00 /usr/lib/x86_64-linux
shiyanol+ 237  0.0  0.0   15564 1148 pts/0    R+   15:30   0:00 ps aux
shiyanolou:~/ $

```

ps axjf

```

shiyanolou:~/ $ ps axjf [15:30:10]
PPID      PID PGID  SID TTY      TPGID STAT  UID    TIME COMMAND
0          1   1    1 ?        -1 Ss    0      0:00 /bin/bash /etc/shiyanolou
1         35  16   16 ?        -1 Sl   5000   0:00 /usr/bin/Xvnc :1 -des
1         44  16   16 ?        -1 S     5000   0:00 /usr/bin/vncconfig -n
1         48  16   16 ?        -1 S     5000   0:00 /bin/sh /etc/xdg/xfce4
48         64  16   16 ?        -1 S     5000   0:00 \_ xfce4-session
1         50   1    1 ?        -1 S     0      0:00 su - shiyanolou bash -
50         54  54   54 ?        -1 Ssl   5000   0:01 \_ node /usr/local/b
1         51   1    1 ?        -1 S     0      0:00 /usr/bin/python /usr/
51         92  92   92 ?        -1 S     0      0:00 \_ /usr/sbin/sshd -D
92         98  98   98 ?        -1 Ss    0      0:00 \_ sshd: shiyanolou
98        112  98   98 ?        -1 S     5000   0:00 \_ sshd: shi
112       113 113 113 pts/0    242 Ss    5000   0:00 \_ -zsh
113       242 242 113 pts/0    242 R+   5000   0:00 \_ p
1         67  16   16 ?        -1 S     5000   0:00 /usr/bin/dbus-launch
1         68  68   68 ?        -1 Ss    5000   0:00 //bin/dbus-daemon --f
1         73  68   68 ?        -1 S     5000   0:00 /usr/lib/x86_64-linux
1         82  82   82 ?        -1 Ss    5000   0:00 /usr/bin/ssh-agent -s
1         84  16   16 ?        -1 S     5000   0:01 xfwm4
1         88  16   16 ?        -1 Sl   5000   0:00 xfce4-panel
88       137  16   16 ?        -1 S     5000   0:00 \_ /usr/lib/x86_64-l
1         90  16   16 ?        -1 S     5000   0:00 Thunar --daemon
1         93  16   16 ?        -1 Sl   5000   0:00 xfdesktop
1         96  96   96 ?        -1 Ssl   5000   0:00 xfsettingsd
1        140 140 140 ?        -1 Ss    5000   0:00 /usr/lib/x86_64-linux
1        141 141 141 ?        -1 Ssl   5000   0:00 /usr/lib/x86_64-linux
1        143 143 143 ?        -1 Ss    5000   0:00 /usr/lib/x86_64-linux
shiyanolou:~/ $

```

总体了解这些信息的意义：

内容	解释
F	进程的标志（process flags），当 flags 值为 1 则表示此子程序只是 fork 但没有执行 exec，为 4 表示此程序使用超级管理员 root 权限
USER	进程的拥有用户
PID	进程的 ID
PPID	其父进程的 PID
SID	session 的 ID
TPGID	前台进程组的 ID
%CPU	进程占用的 CPU 百分比
%MEM	占用内存的百分比

NI	进程的 NICE 值
VSZ	进程使用虚拟内存大小
RSS	驻留内存中页的大小
TTY	终端 ID
S or STAT	进程状态
WCHAN	正在等待的进程资源
START	启动进程的时间
TIME	进程消耗 CPU 的时间
COMMAND	命令的名称和参数

注：TPGID栏写着-1 的都是没有控制终端的进程，也就是守护进程。
STAT表示进程的状态，而进程的状态有很多，如下表所示：

状态	解释
R	Running. 运行中
S	Interruptible Sleep. 等待调用
D	Uninterruptible Sleep. 不可中断睡眠
T	Stoped. 暂停或者跟踪状态
X	Dead. 即将被撤销
Z	Zombie. 僵尸进程
W	Paging. 内存交换
N	优先级低的进程
<	优先级高的进程
s	进程的领导者
L	锁定状态
l	多线程状态
+	前台进程

注：其中的 D 是不能被中断睡眠的状态，处在这种状态的进程不接受外来的任何 signal，所以无法使用 kill 命令杀掉处于 D 状态的进程，无论是 kill，kill -9 还是 kill -15，一般处于这种状态可能是进程 I/O 的时候出问题了。
ps部分常用参数：


```
shiyanolou:~/ $ ps -l [16:00:44]
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  5000  397   396  0  80   0 -  9927 sigsus pts/0    00:00:00 zsh
0 R  5000  418   397  0  80   0 -  1783 -      pts/0    00:00:00 ps
shiyanolou:~/ $ [16:01:38]
```




自定义所需要的参数显示:

```
shiyanolou:~/ $ ps -afxo user,ppid,pid,pgid,command [16:56:14]
USER      PPID  PID  PGID COMMAND
root      0     1     1  /bin/bash /etc/shiyanolou/sbin/init.sh
shiyanol+ 1    30    23  /usr/bin/Xvnc :1 -desktop 27104ce97c0f:1 (shiyanolou)
shiyanol+ 1    39    23  /usr/bin/vncconfig -nowin
root      1    43     1  su - shiyanolou bash -c codebox run /home/shiyanolou/
shiyanol+ 43   47    47  \_ node /usr/local/bin/codebox run /home/shiyanolou/
shiyanol+ 1    45    23  /bin/sh /etc/xdg/xfce4/xinitrc -- /etc/X11/xinit/xs
shiyanol+ 45   59    23  \_ xfce4-session
root      1    46     1  /usr/bin/python /usr/bin/supervisord -n
root      46   90    90  \_ /usr/sbin/sshd -D
root      90  160   160  \_ sshd: shiyanolou [priv]
shiyanol+ 160  171   160  \_ sshd: shiyanolou@pts/0
shiyanol+ 171  172   172  \_ -zsh
shiyanol+ 172  346   346  \_ ps -afxo user,ppid,pid,pgid,com
shiyanol+ 1    62    23  /usr/bin/dbus-launch --sh-syntax --exit-with-sessio
shiyanol+ 1    63    63  //bin/dbus-daemon --fork --print-pid 5 --print-addr
shiyanol+ 1    71    63  /usr/lib/x86_64-linux-gnu/xfce4/xfconf/xfconfd
shiyanol+ 1    77    77  /usr/bin/ssh-agent -s
shiyanol+ 1    79    23  xfwm4
shiyanol+ 1    83    23  xfce4-panel
shiyanol+ 83   99    23  \_ /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-1
shiyanol+ 1    85    23  Thunar --daemon
shiyanol+ 1    87    23  xfdesktop
shiyanol+ 1    92    92  xfsettingsd
shiyanol+ 1   102   102  /usr/lib/x86_64-linux-gnu/scim-1.0/scim-helper-mana
shiyanol+ 1   103   103  /usr/lib/x86_64-linux-gnu/scim-1.0/scim-panel-gtk -
shiyanol+ 1   105   105  /usr/lib/x86_64-linux-gnu/scim-1.0/scim-launcher -d
shiyanolou:~/ $ [16:56:22]
```

通过 `ps tree` 可以很直接的看到相同的进程数量，最主要的是可以看到所有进程之间的相关性:

```
shiyanolou:~/ $ pstree [16:48:04]
init.sh--Thunar
|
|-Xvnc--4*[{Xvnc}]
|   |
|   |-dbus-daemon
|   |-dbus-launch
|   |-scim-helper-man
|   |-scim-launcher
|   |-scim-panel-gtk---{scim-panel-gtk}
|   |-sh---xfce4-session
|   |-ssh-agent
|   |-su---node---5*[{node}]
|   |-supervisord---sshd---sshd---sshd---zsh---pstree
|   |-vncconfig
|   |-xfce4-panel-+-panel-6-systray
|   |   `--{xfce4-panel}
|   |-xfconfd
|   |-xfdesktop--2*[{xfdesktop}]
|   |-xfsettingsd---{xfsettingsd}
|   `--xfwm4
shiyanolou:~/ $ [16:48:07]
```



#参数选择:


#-A : 各程序树之间以 ASCII 字元來連接;

#-p : 同时列出每个 process 的 PID;

#-u : 同时列出每个 process 的所屬账户名称。

pstree -up

```
shiyanolou:~/ $ pstree -up [16:51:05]
init.sh(1)--Thunar(90,shiyanolou)
|
|-Xvnc(35,shiyanolou)--{Xvnc}(36)
|   |
|   |   |-Xvnc(37)
|   |   |-Xvnc(38)
|   |   `--Xvnc(39)
|   |
|   |-dbus-daemon(68,shiyanolou)
|   |-dbus-launch(67,shiyanolou)
|   |-scim-helper-man(140,shiyanolou)
|   |-scim-launcher(143,shiyanolou)
|   |-scim-panel-gtk(141,shiyanolou)---{scim-panel-gtk}(142)
|   |-sh(48,shiyanolou)---xfce4-session(64)
|   |-ssh-agent(82,shiyanolou)
|   |-su(50)---node(54,shiyanolou)--{node}(59)
|   |   |
|   |   |   |-{node}(133)
|   |   |   |-{node}(134)
|   |   |   |-{node}(135)
|   |   |   `--{node}(136)
|   |   |
|   |   |-supervisord(51)---sshd(92)---sshd(1398)---sshd(1409,shiyanolou)---+
|   |   |-vncconfig(44,shiyanolou)
|   |   |-xfce4-panel(88,shiyanolou)-+-panel-6-systray(137)
|   |   |   `--{xfce4-panel}(138)
|   |   |-xfconfd(73,shiyanolou)
|   |   |-xfdesktop(93,shiyanolou)-+-{xfdesktop}(97)
|   |   |   `--{xfdesktop}(129)
|   |   |-xfsettingsd(96,shiyanolou)---{xfsettingsd}(132)
|   |   `--xfwm4(84,shiyanolou)
shiyanolou:~/ $ [16:51:12]
```



进程优先级的值就是上文所提到的 PR 与 nice 来控制与体现了,而 nice 的值是可以
通过 nice 命令来修改的,而需要注意的是 nice 值可以调整的范围是 -20 ~ 19,其中
root 有着至高无上的权力,既可以调整自己的进程也可以调整其他用户的程序,并且是
所有的值都可以用,而普通用户只可以调制属于自己的进程,并且其使用的范围只能是
0 ~ 19,因为系统为了避免一般用户抢占系统资源而设置的一个限制。

示例:

#打开一个程序放在后台,或者用图形界面打开

nice -n -5 vim &

#用 ps 查看其优先级

```
ps -afxo user,ppid,pid,stat,pri,ni,time,command | grep vim
```

还可以用 `renice` 来修改已经存在的进程的优先级: `renice -5 pid`

日志系统:

常见的日志一般存放在 `/var/log` 中, 常见的系统日志有:

日志名称	记录信息
<code>alternatives.log</code>	系统的一些更新替代信息记录
<code>apport.log</code>	应用程序崩溃信息记录
<code>apt/history.log</code>	使用 <code>apt-get</code> 安装卸载软件的信息记录
<code>apt/term.log</code>	使用 <code>apt-get</code> 时的具体操作, 如 <code>package</code> 的下载、打开等
<code>auth.log</code>	登录认证的信息记录
<code>boot.log</code>	系统启动时的程序服务的日志信息
<code>btmpt</code>	错误的信息记录
<code>Consolekit/history</code>	控制台的信息记录
<code>dist-upgrade</code>	<code>dist-upgrade</code> 这种更新方式的信息记录
<code>dmesg</code>	启动时, 显示屏幕上内核缓冲信息, 与硬件有关的信息
<code>dpkg.log</code>	<code>dpkg</code> 命令管理包的日志。
<code>faillog</code>	用户登录失败详细信息记录
<code>fontconfig.log</code>	与字体配置有关的信息记录
<code>kern.log</code>	内核产生的信息记录, 在自己修改内核时有很大帮助
<code>lastlog</code>	用户的最近信息记录
<code>wtmp</code>	登录信息的记录。 <code>wtmp</code> 可以找出谁正在进入系统, 谁使用命令显示这个文件或信息等
<code>syslog</code>	系统信息记录

`wtmp`, `lastlog`这两个日志并不是 ASCII 文件而是被编码成了二进制文件, 所以并不能直接使用 `less`、`cat`、`more` 这样的工具来查看, 需要使用 `last` 与 `lastlog` 工具来提取其中的信息。

`syslog` 是一个系统日志记录程序, 在早期的大部分 Linux 发行版都是内置 `syslog`, 让其作为系统的默认日志收集工具, 虽然随着时代的进步与发展, `syslog` 已经年老体衰跟不上时代的需求, 所以他被 `rsyslog` 所代替了, 较新的 Ubuntu、Fedora 等等都是默认使用 `rsyslog` 作为系统的日志收集工具, `rsyslog` 的全称是 `rocket-fast system for log`, 它提供了高性能, 高安全功能和模块化设计。`rsyslog` 能够接受各种各样的来源, 将其输入, 输出的结果到不同的目的地。`rsyslog` 可以提供超过每秒一百万条消息给目标文件。这样能实时收集日志信息的程序是有其守护进程的, 如 `rsyslog` 的守护进程便是 `rsyslogd`。

`rsyslog` 的配置文件有两个, 一个是 `/etc/rsyslog.conf`, 一个是 `/etc/rsyslog.d/50-default.conf`。

第一个主要是配置的环境, 也就是 `rsyslog` 加载什么模块, 文件的所属者等; 而第二个主要是配置的 `Filter Conditions`。

`rsyslog` 主要是由 `Input`、`Output`、`Parser` 这样三个模块构成的, 并且了解到数据的简单走向, 首先通过 `Input module` 来收集消息, 然后将得到的消息传给 `Parser`

module, 通过分析模块的层层处理, 将真正需要的消息传给 Output module, 然后便输出至日志文件中。

rsyslog 通过 Facility 的概念来定义日志消息的来源, 以便对日志进行分类, Facility 的种类有:

类别	解释
kern	内核消息
user	用户信息
mail	邮件系统消息
daemon	系统服务消息
auth	认证系统
authpriv	权限系统
syslog	日志系统自身消息
cron	计划安排
news	新闻信息
local0~7	由自定义程序使用

而另外一部分 priority 也称之为 serverity level, 除了日志的来源以外, 对统一源产生日志消息还需要进行优先级的划分, 而优先级的类别有以下几种:

类别	解释
emergency	系统已经无法使用了
alert	必须立即处理的问题
critical	很严重了
error	错误
warning	警告信息
notice	系统正常, 但是比较重要
informational	正常
debug	debug 的调试信息
panic	很严重但是已淘汰不常用
none	没有优先级, 不记录任何日志消息

日志这部分内容没怎么看懂, 更多可参考:

<https://www.shiyanlou.com/courses/1/learning/?id=1945>。