Linux 快速上手指南

作者：李航

1 Linux 系统管理基础命令

第一个要学习的命令是？就是如何查看帮助（呵呵，如果不看答案，能否回答出来？）。

Linux 下查看帮助的命令是 man（它是英文单词 manual 的缩写），其常用用法是：

man *命令名*

我们可以使用：man man，查看 man 命令的用法。有关 man 命令的手册页很长，这里仅列出来比较重要的一些内容：Linux 的帮助手册共分为 9 个章节：

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions eg: /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)
9. Kernel routines [Non standard]

那么，这意味着什么呢？这意味着：同一关键字可能在不同的章节中都存在。比如：read 这个关键字，它既出现在第 1 章中，也出现在第 2 章中，所以，你在使用 man 命令时，如果出现这种情况，必须告诉系统到底搜寻的是哪一章节。即在 man 后需要跟一个指明章节号的数字。比如：想要把 read 作为一个系统调用进行查找，那么就应该使用下述命令：

man 2 read

也就是说，仅在第 2 章查找关键字“read”。那么对于上面那种不跟任何章节号的用法，man 命令找到的究竟是哪个章节中的内容呢？此时 man 会从第一章开始查找，直到碰到第一个含有该关键字的章节停下。所以如果直接输入“man read”，那么，找到的结果是第一章中的 read。

1.1 磁盘文件的增删改查

首先，我们来看看磁盘文件的“查”。这里的查分为如下内容：i.查看文件夹下文件的名称。ii.查看某个文件的内容　iii. 查找某个文件的路径名。

1.1.1 查看文件夹下文件的名称

使用 ls 命令。

例子：登入后，敲入 ls，此时将显示当前路径下的文件目录。

```
[root@haha ~]# ls
anaconda-ks.cfg                 rpmbuild
class                            stu
Desktop                           t
Documents                         t.c
Downloads                         Templates
[root@haha ~]#
```

如果要查看文件的详细信息，可以使用：ll 命令

```
[root@haha ~]# ll
total 2261424
-rw-------.  1 root root        1841 Aug 26 12:54 anaconda-ks.cfg
drwxr-xr-x.  3 root root        4096 Jan 12 14:07 class
drwxr-xr-x.  2 root root        4096 Aug 26 18:54 Desktop
drwxr-xr-x.  2 root root        4096 Aug 26 18:54 Documents
drwxr-xr-x.  2 root root        4096 Aug 26 18:54 Downloads
drwxr-xr-x.  2 root root        4096 Jan  5 14:34 gpg
-rw-r--r--.  1 root root       87409 Aug 26 12:54 install.log
…….
```

ll 命令列出所有文件的详细信息。其中第一列表示的是文件类型及权限。Eg: drwxr-xr-x

第一个” d” 代表该文件是一个目录文件。剩下的字符，每 3 位为一组。第一组 rwx 表示该文件的拥有者对该文件的访问权限。r 代表可读，w 代表可写，x 代表可执行。第二组 r-x 代表该文件所属组对其访问权限即：可读，但不可写，可执行。第三组 r-x 代表其它人对该文件的访问权限，即：可读，不可写，可执行。

第二列的内容这里暂时不提，后面再提。

第三列的内容表示该文件的拥有者。

第四列表示该文件的所属组。

第五列表示该文件的大小

第六列表示该文件的修改时间

第七列表示该文件的文件名。

好了，讲了这么多，那么 ll 这些显示的东东究竟在实际中有什么用处呢？其中，最重要的就是判断一个进程是否有权限访问这个文件。有关权限方面的知识，将在后面有关帐户文件管理的部分详细讲述。这里暂且埋下伏笔。

现在大家知道了怎么查看当前目录下所含有的文件列表。那么如果要查看某个特定目录下所含有的文件列表，应该怎么做呢？可以有两种方法：

i. 转到该特定目录下，然后敲入 ls 或 ll 命令。

其中，转到特定目录下使用 cd 命令，eg：

cd *路径名*

注：如果忘了当前路径，可以通过 pwd 命令获得当前路径。

这里需要说明一个概念：HOME 目录，当一个用户刚登陆系统时，其在文件系统中就有一个初始位置，该位置就是 HOME 目录。一个用户的 HOME 目录查看方式就是使用：

echo $HOME

即可。至于上面这条命令究竟是什么意思，我们到学习 shell 编程那一章的时候，再详解。

此外，cd 后面不跟任何参数时，执行会直接转到该用户的 HOME 目录下。

ii. ls 或 ll 命令后跟该特定目录的路径名，eg: ll *路径名*

下面，再介绍几个锦囊妙“技”，解决在实际应用中经常会出现的一些问题：

i. 当文件数目很多，一屏显示不下，那么怎么办？使用如下命令：

ll |less

或

ll |more

其中“|”是管道符。其符号与 C 语言中的按位或相同。

ii. 当敲入多条 ls 命令，使屏显示乱七八糟，那么怎么办？使用如下命令：

clear

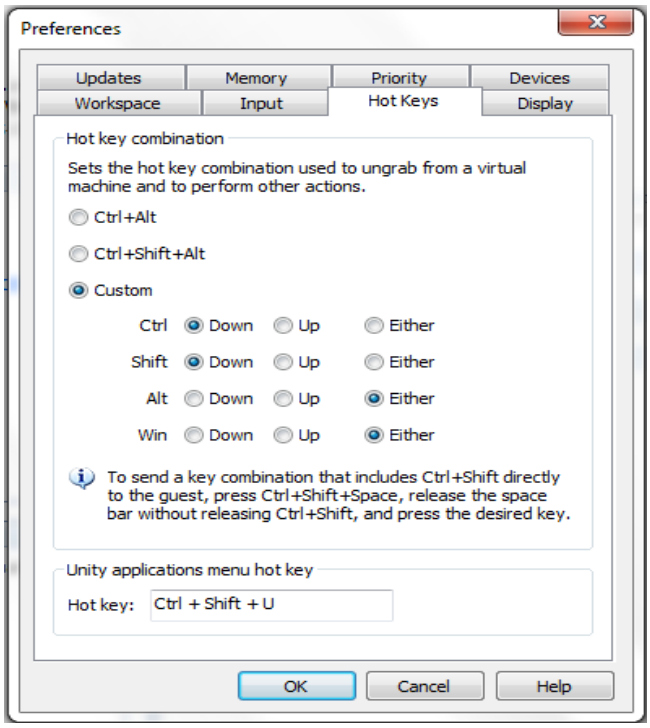
iii. 当项目中的程序文件分散在不同的文件夹中，我想同时看到不同文件夹下的内容，怎么办？

i. 新起一个终端窗口。

ii. 切换到不同的虚拟终端。方法：Ctrl+ALT+Fn (n=1,2,3,…6)

注：有时会出现在虚拟机中切换终端无法成功的情况，这一般是因为 VMware 截获了“Ctrl+ALT”将其当成释放光标的热键。出现这种情况，只需把 VMWare 中的 Edit 菜单中的 preference 标签页中 Hot Keys 改为 Ctrl+Shift 或其它非 Ctrl+Alt 组合即可如下：

。



最后，再留一道思考题：

使用 ll 命令查看 U 盘上的文件，请解释与 Windows 资源管理器所查看的有什么不同，并解释为什么。

1.1.2 查看某个文件的内容

学会了查看文件名及其相关属性信息，下一步自然是查看文件的内容了。文件分 ASCII 文件和二进制文件，那么究竟如何查看这两种类型的文件呢？

对于 ASCII 文件，使用：

cat　*路径名*

如：可以使用 cat　/etc/passwd，显示/etc/passwd 文件的内容。

对于二进制文件，使用 od 命令：

od -Ax -tx1 路径名

上述命令的含义是：以 16 进制查看指定文件的内容。

1.1.2.1 查找某个文件名在文件系统中的位置

查找文件在磁盘上的位置，使用

find *查找起始路径名* -name *文件名*

或者使用:

locate 文件名

其中 locate 方法是最快的，因为它是利用 Linux 系统中的索引数据库（Linux 为磁盘上的文件自动建立了一个索引数据库）进行查找。缺点是：索引数据库的更新不是实时的，所以，有可能出现：一个文件明明在磁盘上存在，却无法找到的情况出现。

1.1.2.2 磁盘文件的增加

磁盘文件的增加主要涉及到：创建新的普通文件、目录文件，拷贝文件，移动文件。下面依次说明：

i. 创建一个普通文件

touch *要创建的新的文件名*

eg: touch x，将在当前目录下创建一个名为 x 的文件

ii. 创建目录文件

mkdir *新的目录文件名*

eg: mkdir d, 将在当前目录下创建一个名为 d 的文件夹，即目录文件

iii. 拷贝文件

cp *源文件路径名* *目的文件路径名*

eg: cp /bin/ls /tmp 将/bin 下的 ls 文件拷贝到/tmp 目录下。

如果要拷贝整个目录，使用：

cp -r 源目录名 目的目录名

即可

iv. 移动文件

mv 源文件路径名 目的文件路径名

eg: mv x ..

将文件 x 移动到父目录下。

1.1.2.3 磁盘文件的删除

磁盘文件的删除主要涉及到：删除普通文件和删除目录文件

i. 删除普通文件

rm *普通文件路径名*

ii. 删除目录文件

rm -rf *目录文件路径名*

1.1.2.4 磁盘文件的修改

磁盘文件的修改主要涉及到：磁盘文件内容的修改以及磁盘文件属性的修改。对于磁盘内容的修修改，我们放到 vi 编辑命令 那一章节完成。对于磁盘文件属性的修改，主要包括：文件名的修改和访问权限的修改。

i. 文件名的修改

mv *原文件名 修改后文件名*

ii. 文件访问权限的修改

chmod 用于修改文件的访问权限位，chown 用来修改文件的拥有者，chgrp 用来修改该文件的所属组。

chmod *权限位 文件名*

eg: chmod 675 f

它将文件 f 的访问权限变为：rw-rwxr-x. 这也就是说它将权限位 3 个看成一组 8 进制的数。6 代表 110，正好表示 r 位打开，w 位打开，x 位关闭。依次类推，所以可以用 3 个 8 进制数分别表示文件拥有者，文件所属组，其它人对该文件的访问权限。

讲到这里，必须说明 Linux 访问权限的实现机制。用一个问题来说明：

ll 命令在超级用户执行时，ll 进程权限是超级用户权限，而 ll 命令被普通用户 stu 执行时，ll 进程权限是普通用户 stu 权限，那么：root 登陆后，在自己的 home 目录下创建一个文件 x。接着使用 ll /root/x，此时，可以看到 x 的详细信息。然后，再切换终端，以普通用户登录，然后执行 ll /root/x，此时，可以看到系统会提示权限不够。这是为什么？不是同一个命令吗？为什么会出现截然不同的结果呢?

这是因为：**虽然是同一命令，但是当 ll 执行时，其拥有的权限是此时所处登陆用户的权限。**比如：当前执行 ll 命令时，所处的登陆用户为: root，那么，ll 程序执行时的权限就是 root。如果所处的登陆用户为 stu，那么，ll 程序执行时的权限就是 stu。

所以，上述 f 权限的含义是：设 f 的拥有者用户名是 stu，所属组是 class，class 中有用户 tom, www，那么：使用 stu 登陆系统，此时所输入命令对 f 访问的权限是可读可写，但不可执行；当使用 tom 或 www 登陆系统，此时输入命令对 f 的访问权限是可读可写可执行；当使用 fei 登陆系统，此时输入命令对 f 的访问权限是可读，不可写，但可执行。

chown 修改一个文件的拥有者。注意，该命令只能 root 用户执行。

chown root f
将 f 的拥有者改为 root。

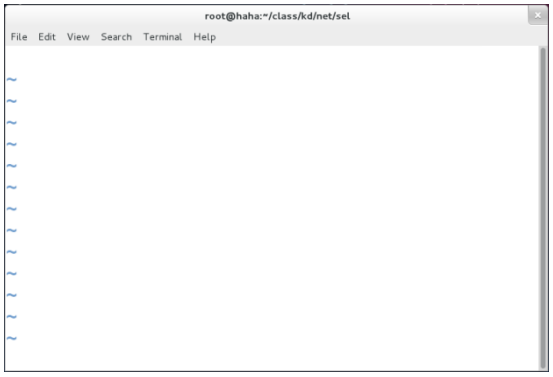
chgrp 修改一个文件的所属组。
chgrp root f
将 f 的所属组改为 root。

1.2 Linux 系统管理命令进阶

上面对磁盘文件的增删改查的学习中，我们还未讨论文件内容的修改命令。本节我们将学习 vim，一个命令行下的文本编辑命令，完成对文件内容的修改。此外，我们要学习 Linux 下 C/C++程序的编译和调试（赫赫，相信好学的同学已经等得手发痒了吧^_^）。

1.2.1 vim

在 Linux 进行文件的编辑，主要用两种工具：vim 和 gedit。其中 gedit 是图形界面下的编辑工具。那么，同学可能会很奇怪，那么我们为什么不讲图形界面下的 gedit，偏偏讲一个非常难用的命令行下的 vim 呢？这个问题留给大家思考
vim 官网的教程，大家猜猜多少页？575 页。所以，我们不能再死记硬背了吧。那么我们怎么学呢？答案很简单，因为它是一个文本编辑器，只需学会：查找，插入，删除，拷贝，撤销，保存，退出。即可。
在 fedora 下使用 vi 命令，即可启动 vim：
vi　文件名



此时：敲入 i，将进入输入文本模式：

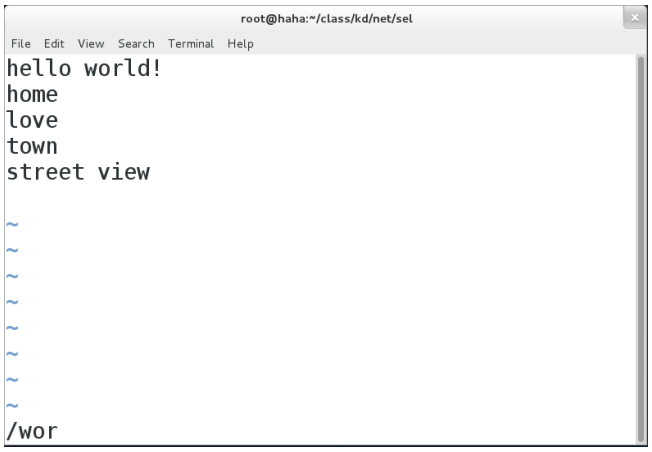
此时，左下角出现“INSERT”表示现在可以进行输入文本操作了。输入完毕后，敲入 ESC，将切换到编辑命令模式下，然后敲入“:”再敲入 wq，再回车就保存退出了。

那么，怎么拷贝呢？首先在编辑时，将光标移动到你想要拷贝的起始位置，然后，敲入 ESC，切换到编辑命令模式下，再敲入 v，进入 VISUAL 模式，再移动光标，移动到你要拷贝的结束位置：

此时，再敲入 y，就将内容拷贝好了。此时，再将光标移动到你要粘贴的位置，再敲入 p，就完成粘贴了！

如果要撤销对文本的修改，只需使用 ESC 进入编辑命令模式，然后再敲入 u，即可。

对于查找，只需使用 ESC 进入编辑命令模式，然后输入 “/关键字”回车即可：



最后，需要指出的是：有时 vi 会异常退出时，然后再使用 vi 命令时，会出现：

E325: ATTENTION
Found a swap file by the name ".tmp.swp"
owned by: root dated: Fri Mar 7 11:51:36 2014
file name: ~root/class/kd/net/sel/tmp
modified: YES
user name: root host name: haha
process ID: 20332
While opening file "tmp"
dated: Fri Mar 7 11:51:24 2014

(1) Another program may be editing the same file. If this is the case,
be careful not to end up with two different instances of the same
file when making changes. Quit, or continue with caution.
(2) An edit session for this file crashed.
If this is the case, use ":recover" or "vim -r tmp"
to recover the changes (see ":help recovery").
If you did this already, delete the swap file ".tmp.swp"
-- More --

这是因为临时文件造成的，此时，只需退出 vi，然后删除临时文件即可。临时文件一般以
.编辑文件名.swp
命名。

1.2.2 编译调试 C/C++程序

1.2.2.1 编译 C/C++程序

在 Linux 下编译 C/C++程序的命令是 gcc/g++，同样官网 gcc/g++的手册页至少 796 页。不过，最常用的命令选项是下面的形式：

- 形式 1：编译 C 程序文件

  gcc　-o　可执行文件名　-g　源文件名

  -o 选项用来指明生成的可执行程序的名字。-g 选项用来添加调试信息到生成的可执行程序。如果没有-g 选项，一般生成的可执行程序大小会更小一些，但不能进行源码一级的调试。这里要特别注意一点，编译 C 语言源文件，源文件名必须以.c 或.h 结尾，否则就会出错。

- 形式 2：编译 C++程序文件

  g++　-o　可执行文件名　-g　源文件名

  同样，-o 选项用来指明生成的可执行程序的名字。-g 选项用来添加调试信息到生成的可执行程序（呵呵，发没发现 gcc 和 g++很类似？实际上 g++是 gcc 的一个外壳，只不过比 gcc 多链接一个 stdc++库而已）。同样注意一点，编译 C++语言文件，源文件名必须以.cpp 或.h 结尾，否则就会出错。

- 形式 3：编译生成.o 文件

  对于 C 程序：

  gcc　-c　以.c 为后缀名的源文件名

  对于 C++程序

  g++ -c　以.cpp 为后缀名的源文件名

  eg:

  gcc -c　stu.c

  将生成一个名为 stu.o 的文件

这里再留一个问题供大家思考：.o 文件和可执行程序文件的区别。

1.2.2.2 调试 C/C++程序

目前，调试程序共有两种方法：i.使用调试器 ii.记录日志文件。这两种方法有各自的优缺点，简单地说，调试器功能强大，能够单步执行，将程序执行过程看得很清楚，但不易发现进程并行执行时才出现的错误。记录日志文件虽然比较笨拙，但它可以发现进程并行执行时才出现的错误

调试 C/C++程序使用 gdb。具体使用方式如下：

gdb *可执行程序名*

进入 gdb 界面后，首先设置端点，然后运行被调试程序。当被调程序运行到断点处就会停下来。此时就可查看变量的值看看是否正确。下面给出一个完整的调试流程：

b main　　注：在 main 函数处设置端点

r　　注：运行被调程序，如果 main 函数执行需要传递参数，使用: r　参数 1　参数 2 ….

p 变量名　　注：显示变量的值。

n　　注：继续向下执行一行。

s　　注：继续向下执行一行。

b　filename:n　　注: 在名为 filename 的文件的第 n 行上设置断点。

c　　注：继续向下运行直至碰到一个断点

p　var=val　注：强制将变量 var 的值置为 val

p　$esp　　注：查看寄存器 esp 的值

l　　注：查看当前执行位置附近的源码。

1.2.3 makefile 的编写

什么是一个 makefile？makefile 就是含有一系列编译指令和规则的文件。那么，我们为什么要用 makefile 呢？原因很简单：编译一个大型系统会消耗大量的时间，此时，如果开发者仅仅因修改一个文件，就需要整个重新编译每个程序文件的话，那也太浪费时间了；而使用 makefile 可以避免这一点。即：只编译那些修改过的文件以及受到文件修改影响的那些文件。

我们还是以一个例子说明 makefile 的写法吧：

两个源文件：t.c 和 f.c， f.c 中含有一个函数 f， t.c 中 main 调用 f， makefile（注意：该文件名就叫 makefile）就可以书写为下面的形式：

```
t: t.o f.o
        gcc -o t -g t.o f.o
t.o: t.c
        gcc -c t.c
f.o: f.c
        gcc -c f.c
```

编写完毕后，只要敲入 make，系统将自动开始编译过程，自动生成一个名为 t 的可执行文件。此时，如果你仅修改该 f.c 的内容，修改完毕后，再敲入 make，它将只重新执行下面两句编译指令：

```
gcc -c f.c
gcc -o t –g t.o f.o
```

而不会在重新编译 t.c。神奇吧？下面我们就剥去 makefile 的神秘面纱，一探究竟。

Makefile 是由一条条的规则构成的，如下：

目标文件名：依赖文件名 …

指令

其中目标文件名就是要生成的目标文件的名称。依赖文件名就是要生成目标文件所要依赖的文件的名称。指令就是要执行的命令。特别注意：指令前面一定是 tab 键，不是空格，否则 make 会出现奇怪的错误。

那么 makefile 究竟是怎么执行的呢？

make 命令实际上是 makefile

的第一条规则是：

```
t: t.o f.o
        gcc -o t -g t.o f.o
```

那么它就将第一条规则中的 t 设置为 makefile 执行的总目标，即最终要生成的文件。然后将依赖文件 t.o 和 f.o 看成是完成该总目标的第一次分解。即：将任务目标 t，分成两个子目标：t.o 和 f.o。根据规则：

```
t.o: t.c
        gcc -c t.c
```

将 t.o 这个任务目标分解为 t.c 这个任务目标。而此时 t.c 已经存在。也就是说 t.o 已经具备完成的基础，那么就开始执行 gcc -c t.c。同样，对于 f.o，遵循同样的规则，生成 f.o。那么此时 f.o t.o 已经存在，那么具备完成目标 t 的能力，那么，此时就执行 gcc -o t -g t.o f.o，生成 t。

现在，我们已经知道第一次执行 makefile 的总体流程，现在我们看看当 make 第一次执行完毕，修改 f.c 后，再执行第二次 make 时的情形（注意：此时，可执行文件 t, t.o,f.o 均存在）：

首先 make 碰到第一条规则：

```
t: t.o f.o
        gcc -o t -g t.o f.o
```

那么，就会知道总目标 t 依赖于两个子目标 t.o,f.o，它会继续向下分解，发现规则：

```
t.o: t.c
        gcc -c t.c
```

而 t.o 的子目标 t.c 无法再继续向下分解，那么，此时它就开始比较 t.o 和 t.c 谁的修改该时间更新。很显然 t.o 的修改时间是新于 t.c 的（因为没有修改 t.c,而且 t.o 是靠 gcc 在 t.c 生成之后生成的）。这说明什么？没有必要重新编译 t.c。那么，make 再往下走，碰到规则：

```
f.o: f.c
        gcc -c f.c
```

此时，make 发现 f.o 的子目标 f.c 的修改时间新于 f.o，这说明在 f.o 生成之后，f.c 进行了改动。而 f.c 无法再继续向下分解，所以，此时，make 将调用 gcc -c f.c

对 f.c 进行重新编译。那么生成新的 f.o，而此时，一旦生成新的 f.o，就会回溯到前面的目标分解，即将 t 目标分解为两个子目标 t.o f.o。此时就意味着 f.o 的修改时间新于 t 的修改时间。那么怎么办？当然是重新编译生成 t 喽。即执行：gcc -o t -g t.o f.o

啰嗦了这么多，总结一下就是：make 会根据规则逐层分解目标，直到 不能分解为止，然后从最底层开始比较目标文件和依赖文件的新旧，当依赖文件新于目标文件时（当目标文件不存在，也当作这种情况处理），则执行规则所规定的指令。然后同法向上回溯，直至达到最高层的总目标。

这也解释了为什么第一次 make 执行完毕后，删除 t.o，再次执行 make，将执行：

gcc -c t.c 和 gcc -o t -g t.o f.o

现在，再举一例，让大家更深入地了解 makefile 的执行流程：

t.c 和 f.c， f.c 中含有一个函数 f， t.c 中 main 调用 f，然后使用如下 makefile 进行多次编译，会出现什么情况

```
t: t.o f.o
        gcc -o t1 -g t.o f.o
t.o: t.c
        gcc -c t.c
f.o: f.c
        gcc -c f.c
```

此时，你会发现，第一次执行 make，会执行每条规则中的指令，从第二次起，每次 make 均只执行: gcc -o t1 -g t.o f.o。为什么？因为虽然此时 t.o 和 f.o 已存在，并且不会改变，但此时目标 t 文件一直不存在（因为生成的是 t1），所以，就会认为 t.o,f.o 文件的修改时间新于 t，所以不断重复执行 gcc -o t1 -g t.o f.o