Linux入门笔记——常用命令

在linux系统中，指令一般由 `command [options] [arguments]`构成

选项分为 **短选项** 与 **长选项**，短选项由单个连字符跟一个或多个字母，长选项由两个连字符跟一个完整单词

多个短选项可以组合在一起

参数是命令操作的对象，一般是文件，目录名或其他输入

基础操作与文件管理

文件与目录操作

`ls`：查看文件和目录（list）

`ls [选项] [文件或目录]`

- `-l`：以长格式显示，包含权限、所有者、大小、修改日期等详细信息
- `-a`：显示所有文件，包括以.开头的隐藏文件
- `-h`：配合 `-l`使用，以可读格式（KB，MB，GB）显示文件大小
- `-R`：递归列出，遍历文件的所有子目录
- `-t`：按时间排序列出

一般情况下 `-R`和 `-r`是可以互换使用的，都表示递归，但一些命令（如chmod）更倾向于 `-R`，并且在不同linu和unix系统中 `-R`更加通用

```
#查看当前目录下所有文件的详细信息
$ ls -l
drwxr-xr-x 2 user group 4096 Sep 5 10:30 my_project
-rw-r--r-- 1 user group 1024 Sep 5 10:31 README.md
```

`cd`：切换目录（change directory）

`cd [目录]`

- `cd ..`：返回上一级
- `cd ~`或 `cd`：返回当前用户目录
- `cd -`：返回上次目录
- `cd /`：返回根目录

路径可以是绝对路径（以/开头）或相对路径（不以/开头）
例如，如果当前在 `/home/user`，要进入project目录，可以使用`cd project`。要进入 `/var/www`，则必须使用 `cd /var/www`

```
cd /var/www/html
```

---

pwd：显示当前路径（print working directory）
显示当前绝对路径

```
$ pwd
/home/user/my_project
```

---

mkdir：创建目录（make diirectory）

mkdir [选项] [目录名]

- -p：递归创建目录，如果父目录不存在会一并创建

如不使用 -p选项，否则父目录不存在会报错，并且父目录需要写入权限

```
mkdir -p src/utils
```

---

rm：删除文件或目录（remove）

rm [选项] [文件或目录]

- -r：递归删除，用于删除目录及其所有内容
- -f：强制删除，不提示确认
- -i：每次删除前询问
- -d：删除空目录 一般情况下，rm是不会删除目录的

删除文件：从目录中移除该文件的链接，而不是真正擦除磁盘上的数据，只有所有链接都删除后，文件才会被系统回收

删除目录：递归删除目录中的所有文件和子目录，直到目录为空，然后删除目录本身，**即删除目录需要加上 -r**

rm -rf是一个非常危险的命令，一旦执行数据几乎无法恢复！使用时务必确认路径无误

```
#删除名为old_file.txt的文件
rm old_file.txt

#递归删除名为old_project的目录及其所有内容
rm -r old_project
```

`cp`：复制文件或目录（copy）

`cp [选项] [源文件或目录] [目标文件或目录]`

- `-r`：递归复制，用于复制目录及其所有内容
- `-i`：在覆盖同名文件前进行提示
- `-p`：目标位置的文件权限可能会受影响，使用-p保留源文件的权限

复制目录时一定要使用 `-r`选项

```
#将file.txt复制到同一目录下，并命名为file_copy.txt
cp file.txt file_copy.txt

#将file.txt复制到上级目录的docs目录中
cp file.txt ../docs

#递归复制project_template/目录到new_project/
cp -r project_template/ new_project/
```

`mv`：移动或重命名文件（move）

`mv [选项] [源文件或目录] [目标文件或目录]`

- `-i`：覆盖同名文件时询问

```
#将old_name.txt重命名为new_name.txt
mv old_name.txt new_name.txt

#将new_app目录移动到/var/www/目录下
mv new_app/ /var/www/
```

文件内容操作

`cat`：显示文件内容（concatenate）

`cat [选项] [文件]`

```
#显示script.js文件的所有内容
cat script.js

#将两个文件的内容连接起来并一起显示
cat file1.txt file2.txt
```

适用于查看较小文件，不建议查看大型日志文件等，造成刷屏

---

head / tail：查看文件开头或结尾

```
head [选项] [文件]
tail [选项] [文件]
```

- -n：后接数字，表示查看含行数，若数字前加-，表示除了该行

另外，对于tail还有

- -f：随文件增长即时输出新增数据

```
#查看my_code.py文件的前5行
head -n 5 my_code.py

#查看my_code.py文件的最后10行
tail -n 10 my_code.py

#持续监控server.log文件
tail -f server.log
```

---

less：分页查看文件 前文提到不用cat查看大型文件，一般用less查看

```
less [选项] [文件]
```
常用快捷键：

- space：下一页
- b：上一页
- j：下一行
- k：上一行
- /关键词：对关键词进行搜索
- q：退出
- :e：查看新文件

```
#打开一个大型日志文件
less huge_log_file.log

#通过管道将cat的输出传给less
cat huge_log_file.log | less
```

文件查找与搜索

find：在文件系统中查找文件

```
find [查找路径] [匹配条件] [执行动作]
```
查找路径：

- .：表示当前目录，最常用的目录

- `/`：根目录

```
#在当前目录开始查找test.py
find . -name "test.py"
```

匹配条件：

- `-name "文件名"`：按文件名精确匹配
- `-iname "文件名"`：不区分大小写地按文件名匹配
- `-type [类型]`：指定要查找的文件类型
    - `f`：普通文件
    - `d`：目录
    - `l`：符号链接

```
#查找名为my_app的目录
find . -type d -name "my_app"
```

- `-size [大小]`：根据文件大小查找
    - `+`：大于
    - `-`：小于
    - `c`：字节
    - `k`：KB
    - `M`：MB
    - `G`：GB

```
#查找大于1MB的文件
find . -size +1M
```

此外对于条件常用逻辑组合：

- `-a`：and
- `-o`：or
- `!`：not

```
#查找所有.txt或.md文件
find . -type f -name "*.txt" -o -name "*.md"
```

执行动作：

- `-print`：打印出匹配的文件的完整路径，这是默认行为
- `-delete`：删除找到的文件，比较危险，谨慎使用

```

```

```
#删除所有以.log 结尾的文件
find . -name "*.log" -delete
```

- `-exec [命令] {} \;`：对每一个找到的文件执行指定的命令
    - `{}`：代表find找到的文件
    - `\;`：表示命令结束

但一般会使用 `-exec [命令] {} +`，这个指令与上一个类似，不过会将所有找到的文件作为参数一次性传递给命令，而不是逐个执行，因此对于一般命令会更加高效

```
#找到所有.js文件并用ls -l查看详细信息
find . -name "*.js" -exec ls -l {} \;

#删除所有 .tmp 文件
find . -name "*.tmp" -exec rm {} +
```

注意：对于文件与目录类型的参数，双引号不是必须的，当文件名含有空格或特殊字符时才必须加双引号

---

grep：在文件中搜索文本内容

grep通常使用于文本文件，不适合二进制文件
在搜索特殊字符时注意用引号括起

`grep [选项] "匹配模式" [文件名]`

- `-i`：忽略大小写
- `-n`：显示匹配内容的行号
- `-r`：递归搜索，在指定目录下所有文件中查找
- `-A [N]`：显示匹配行及其后面的N行
- `-B [N]`：显示匹配行及其前面的N行
- `-C [N]`：显示匹配行及其前后各N行

匹配模式支持普通文本和正则表达式，要搜索文件可以同时指定多个，也可以利用管道等

```
#在当前目录下所有文件中查找包含main.js的文本
grep -r "main.js" .

#显示包含error的行以及其前后各3行
grep -C 3 "error" app.log

#匹配多种可能
grep a.c app.log
```

对于一些常用的正则表达式：

char	meaning	example	explain
.	匹配任意单个字符	a.c	匹配abc、adc等
*	匹配其前一个字符的零个或多个	go*gle	匹配gogle、google、gooogle等
^	匹配行首	^start	匹配以 start 开头的行
$	匹配行尾	end$	匹配以 end 结尾的行
[]	匹配方括号内的任意一个字符	[aeiou]	匹配任意一个元音字母
[^]	匹配不在方括号内的任意一个字符	[^aeiou]	匹配任意一个非元音字母
<	匹配单词的开头	<word	匹配以 word 开头的单词
>	匹配单词的结尾	word>	匹配以 word 结尾的单词

由此有常用的组合 .*贪婪匹配，表示任意多个任意字符

压缩与解压

tar：归档和压缩

归档即打包的意思

tar本身只打包，不压缩，为了实现压缩，通常会与其他压缩工具如gzip或bzip2结合使用，因此常看到.tar.gz或.tar.bz2这样的文件

tar [选项] [归档文件名] [要操作的文件/目录]

- -c：创建归档文件（create）
- -x：解压归档文件（extract）
- -f：指定归档文件的文件名，这个选项必须紧跟文件名（file）
- -v：显示详细的归档或解压过程（verbose）
- -z：通过gzip压缩或解压（gzip），用于.tar.gz或.tgz文件
- -j：通过bzip2解压或压缩（bzip2），用于.tar.bz2或.tbz文件
- -J：通过xz解压或压缩，通常用于.tar.xz文件
- -C：在指定解压目录解压

```
#将my_project目录打包并压缩为my_project.tar.gz
tar -czvf my_project.tar.gz my_project/

#解压my_project.tar.gz到当前目录
tar -xzvf my_project.tar.gz

#将archive.tar.gz解压到/home/user/docs目录下
tar -xzvf archive.tar.gz -C /home/user/docs
```

---

zip：压缩与 unzip：解压

tar命令可以进行分布操作，可以只打包不压缩，也能完整保留文件的所有权限、时间戳、所有者和组等信息，因此在linux/unix系统中tar往往是首选

但如果要跨平台数据交换，为了保证对方顺利解压，需要用到 zip，zip格式是跨平台标准，windows，macos和linux都内置了对.zip文件的支持，且zip可以一次性完成打包和压缩，也可以进行增量压缩，即可以方便的从压缩包中提取单个文件，而无需解压整个压缩包（似乎没用

zip [选项] [压缩包名] [参数]

- -r：递归压缩
- -e：创建加密的压缩包，会提示输入密码

```
#将file1.txt和file2.txt压缩到files.zip
zip files.zip file1.txt file2.txt

#将整个my_project目录压缩成my_project.zip
zip -r my_project.zip my_project/
```

注意：对于tar和zip在压缩目录时都是内置且默认递归的，但是在用zip压缩目录时通常加上-r来明确目的

unzip [选项] [压缩包名] [参数]

- -d：指定压缩目录
- -l：列出压缩包中的内容，但不解压

```
#解压files.zip到当前目录
unzip files.zip

#将archive.zip解压到/home/user/docs目录下
unzip archive.zip -d /home/user/docs
```

用户、权限与系统管理

在linux中，每个文件和目录都有三组权限：所有者（user）、所属组（group）和其他用户（other）

每组权限又分为三种：读（read，r）、写（write，w）和运行（execute，x）

---

chmod：修改文件或目录权限（change mode）

chmod [选项] [权限模式] [文件/目录名]

- -R：递归修改，注意用R，以免与r（只读）混淆

权限模式：

- 数字法：权限用三个八进制数字表示，每个数字代表一个权限级别
  - 第一位：所有者

    - 第二位：所有组
    - 第三位：其他用户
    - 权限值：r=4,w=2,x=1 讲需要的权限值相加

例如：755=rwx+r-x+r-x

- 符号法：更加灵活可以增量或减量修改权限
    - u：所有者
    - g：所有组
    - o：其他用户
    - a：所有用户
    - +：添加权限
    - -：移除权限
    - =：设置权限

例如：chmod u+x script.sh给所有者添加运行权限

---

chown：修改文件或目录所有者（change owner）

chown [选项] [新所有者]:[新所属组] [文件/目录名]

- -R：递归修改

对于[新所有者]:[新所属组]，缺省表示不修改

```
#只修改所属组，所有者不变
chown :newgroup file.txt
```

chown命令只能由root用户或具有相应权限的用户执行 而chmod可以由文件或目录所有者和root执行

---

whoami：显示当前用户名

---

sudo：以管理员权限执行命令

在linux系统中，任何设计到系统级资源或者需要特殊权限的操作，都需要用到sudo

- 安装、卸载或更新软件
- 管理系统服务
- 修改系统配置文件
- 读写或删除受保护的目录或文件
- 管理用户或组

sudo [选项] [命令]

系统状态与进程管理

进程 是程序的一个运行实例，当执行一个命令或打开一个程序时，就会创建一个进程，linux作为一个多任务操作系统，可以同时运行成千上万个进程

默认情况下，在终端执行的任何命令，都会作为前台进程运行，只能在一个命令执行完或手动终止后能输入下一个命令，但linux真正的多任务能力体现在后台进程上

在命令末尾加上 &可以使命令在后台进行：[命令] &

---

top/htop：实时查看系统资源和进程

top
top提供一个动态的、实时的系统性能概览，按 q键退出，按 k键，输入进程pid可以杀死该进程

```
$ top
#顶部显示系统概览：系统时间、运行时间、登录用户、平均负载等
top - 10:30:00 up 2 days,  1:00,  1 user,  load average: 0.10, 0.15, 0.20
#接下来是进程列表，按CPU或内存排序
PID   USER   PR   NI   VIRT   RES   SHR  S   %CPU   %MEM   TIME+   COMMAND
1234  user   20   0    1.5g   120m  50m  S   5.0    3.0    0:05.12 node
```

htop 是 top的增强版，提供了更加美观，更易交互的界面，适于初学者

htop需要手动安装

---

ps：查看当前进程

与top的实时性不同，ps用于查看系统某一时刻的进程状态，像一个“进程快照”

ps [选项]

- ps aux：最常用的组合，a显示所有用户的进程，u详细显示，x显示没有控制终端的进程
- ps -ef：e显示所有进程，f以树形结构显示进程关系

控制终端：
当你打开一个终端，它会创建一个交互式会话，在这个会话中你输入的命令会作为前台进程运行，它们的输入来自键盘，输出显示在终端屏幕上，这个终端即称为进程的控制终端

没有控制终端的进程（守护进程）：
当一个进程不需要与用户交互，并且需要在后台持续运行以提供服务时，就会脱离控制终端，称为没有控制终端的进程，一般有以下特点：

- 与用户无关
- 在后台运行
- 提供系统服务

守护进程由系统启动时自动运行，脱离任何终端，在后台工作直到系统关闭，保证了服务的持续可用性和稳定性

在需要快速查找某个特定进程是否存在以及相关信息时，可搭配 grep使用

```
#查找所有与nginx相关的进程
$ ps aux | grep nginx
```

在这个指令中，ps aux输出的是一大段文本，包含所有进程的详细信息，而 |的作用是将其哪一个命令的输出作为后一个命令的输入，因此即为在接受到的信息中查找有nginx的行

---

kill/killall：终止进程

kill 是一个系统调用，它向 Linux 内核请求，让内核向指定的进程发送一个信号 因此它的本意并非是“杀死”进程，而是 **“向进程发送信号”**

linux中定义了许多信号，如：

- SIGINT(2)：在终端按 Ctrl+C，用于中断一个命令
- SIGTERM(15)：终止信号，kill的默认信号
- SIGKILL(9)：强制杀死信号，这是无法被忽略或捕获的信号

`kill [信号] [PID]`

- `kill PID`：发送默认的SIGTERM信号，请求进程优雅退出
- `kill -9 PID`：发送SIGKILL信号，强制立即终止进程

优雅退出：
当kill命令向进程发送SIGTERM信号时，它不是直接终止进程，而是提醒进程准备好退出

此时进程可以：

- 执行清理任务：保存当前工作，释放内存，关闭文件，保存日志
- 退出：清理完成后自动终止
- 忽略信号：有些程序设计成忽略，为了防止被意外终止

```
#先用ps找到进程PID
$ ps aux | grep my_app
user  12345  0.5  2.0  123456  50000  pts/0 S+  10:00  0:05  python  my_app.py
#终止该进程
$ kill 12345
```

如果不确定进程的pid，但知道名称时，用killall会比较方便，不用先ps再kill

`killall [信号] [进程名]`

- `killall nginx`：终止所有名为nginx的进程
- `killall -9 nginx`：强制终止所有名为nginx的进程

```
#直接终止所有名为my_app的进程
$ killall my_app
```

---

df：查看磁盘使用情况

df [选项] [文件/目录]

- -h：以人类可读的格式显示
- -T：显示文件系统类型

```
$ df -h
Filesystem   Size   Used   Avail   Use%   Mounted on
/dev/sda1    50G    25G    25G     50%    /
tmpfs        3.9G   0      3.9G    0%     /dev/shm
```

文件系统： 操作系统用来组织和管理存储设备上数据的方法和结构，在检查问题的时候有用，对现阶段过于神秘

软件与包管理

包管理器是linux系统最核心的工具之一，能实现轻松的安装、更新、卸载软件

不同linux的发行版使用不同的包管理器

- Debian/Ubuntu：apt或apt-get
- Red Hat/CentOS：yum或dnf

---

update：更新可用软件包列表

apt update
能够更新本地的软件包索引，从软件仓库服务器下载最新的软件包列表，但不会安装或升级任何软件包

在安装新软件或升级系统前，执行该命令以确保获取的是最新可用的版本

```
#更新本地软件包索引
sudo apt update
```

---

install：安装软件包

apt install [软件包名]
检查本地索引，查找软件包及其所有依赖项，然后从软件仓库下载并安装它们

```
#安装nginx Web服务器
sudo apt install nginx
#安装git
sudo apt install git
```

---

upgrade：升级已安装的软件包

`apt upgrade`
根据apt update后的最新索引，找出所有可以升级的软件包，并下载安装新版本

```
#升级所有软件包
sudo apt upgrade
```

---

remove：卸载软件包

`apt remove [软件包名]`
卸载指定的软件包，但会**保留其配置文件**

```
#卸载nginx
sudo apt remove nginx
```

---

purge：完全卸载软件包

`apt purge`
彻底卸载指定的软件包，**包括其配置文件**

search：搜索软件包

`apt search [关键词]`
在不确定包名时，可用于搜索软件仓库中可用的软件包

网络与远程连接

---

ssh：远程登录到服务器（Secure Shell）

`ssh [用户名]@[远程主机名或IP地址]`

```
#登录到IP地址为192.168.1.100的服务器，使用devuser用户
ssh devuser@192.168.1.100
```

客户端利用ssh与远程服务器建立加密连接，服务器验证你的身份（通过密码或密钥），然后供一个Shell环境，所有数据传输都是加密的，防止被监听

---

scp：在本地和远程服务器之间复制文件（Secure Copy）

scp是基于SSH协议的文件传输工具，用于在本地和远程主机之间安全地复制文件和目录

`scp [选项] [源文件] [目标位置]`

- 从本地复制到远程：

```
#将本地的index.html文件复制到服务器的/var/www/html目录
scp index.html devuser@192.168.1.100:/var/www/html/
#复制整个my_project目录到服务器
scp -r my_project devuser@192.168.1.100:/var/www/
```

- 从远程复制到本地：

```
#将服务器的error.log文件复制到本地的/home/devuser/logs目录
scp devuser@192.168.1.100:/var/log/nginx/error.log /home/devuser/logs/
```

---

ping：测试网络连接

ping通过发送ICMP Echo Request（回声请求）数据包来工作。如果目标主机可达，它会回复一个ICMP Echo Reply（回声应答）

ping命令是网络诊断中最简单、常用的工具，能检查主机是否能到达目标主机

`ping [选项] [目标主机]`

- `-c [次数]`：指定发送数据包的次数，可以避免无限循环测试
- `-i [秒]`：指定发送数据包的时间间隔

目标主机：可以是ip地址（如192.168.1.1）或域名（如google.com）

```
#检查能否连接到example.com
ping example.com
#按Ctrl+C停止

#发送5个数据包到google.com
ping -c 5 google.com
```

---

ip或ifconfig：查看和配置网络接口

这两个命令通过查询操作系统内核中关于网络接口的信息来工作，获取和配置网络信息（查看我的ip地址、网卡是否启用等）

最常使用 `ip a`或 `ip addr`：显示所有网络接口的IP地址和状态

```
#查看所有网卡的详细信息，包括IP地址
ip a
```

```
# 输出示例:
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    inet 192.168.1.10/24 brd 192.168.1.255 scope global eth0
```

示例解析:
第一行提供了网卡的基本信息，类似个人简历

- `2: eth0:`:
  - `2:` ：网卡编号
  - `eth0`: 网卡名称
- `<BROADCAST,MULTICAST,UP,LOWER_UP>`:
  - `BROADCAST`: 表示网卡支持广播通信
  - `MULTICAST`: 表示网卡支持多播通信
  - `UP`: 表示网卡已启用，若被禁用显示 `DOWN`
  - `LOWER_UP`: 表示物理连接正常（网线已插入或Wi-Fi已连接），若不正常显示 `LOWER_DOWN`
- `mtu 1500`: mtu(Maximum Transmission Unit): 最大传输单元，这个网络接口一次可发送最大数据包（单位是字节）
- `state UP`: 再次确认网卡状态，`UP`表示网卡正在运行

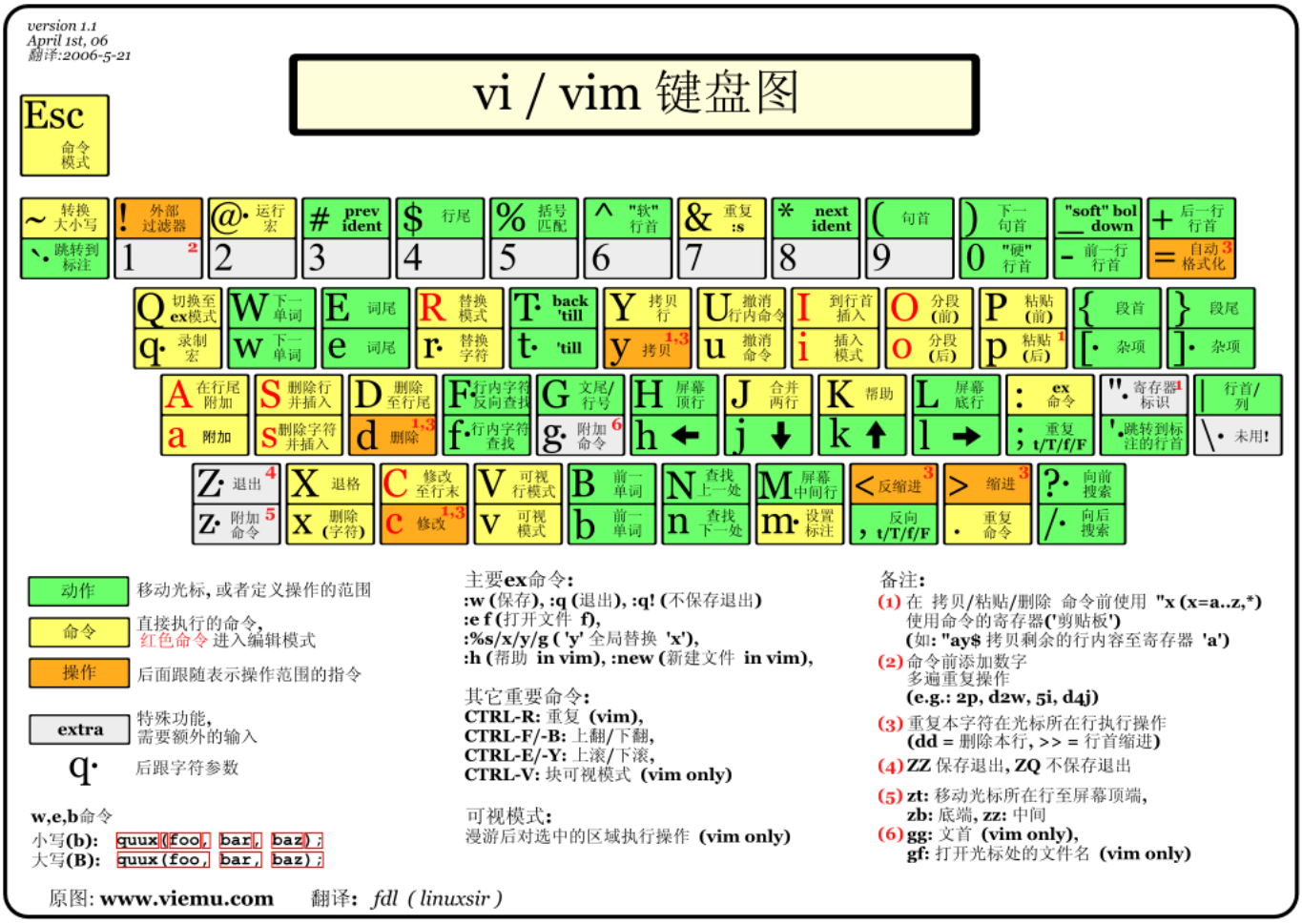
第二行详细说明网卡的网络地址配置

- `inet 192.168.1.10/24`:
  - `inet`: 表示这是一个ipv4地址
  - `192.168.1.10`: 网卡的ip地址，设备在网络中的唯一标识
  - `/24`: 子网掩码
- `brd 192.168.1.255`: brd (broadcast): 广播地址，用于向该网络中的所有设备发送数据
- `scope global`: 表示这个IP地址可以在全球网络中使用或在本地网络中被访问

`ifconfig`作用与 `ip`类似，不过略有过时，在许多新发行版可能需要单独安装

Vim文本编辑器

关于vim的基本操作，结合下图和vimtutor



除此之外还有一些技巧：

快速移动与跳转

- `Ctrl+o`：跳转到上一次光标所在的位置
- `Ctrl+i`：跳转到下一次光标所在的位置
- `g;`：跳转到上一次编辑（插入或删除）的位置
- `g,`：跳转到下一次编辑的位置

多文件操作

- 缓冲区（Buffer）
  - `vim file1.txt file2.txt`：同时打开多个文件
  - `:ls`：列出所有打开的缓冲区
  - `:bN`：切换到编号为N的缓冲区
  - `:bn`：切换到下一个缓冲区
  - `:bp`：切换到上一个缓冲区
  - `:bd`：关闭当前缓冲区
- 分割窗口（Split Window）
  - `:sp filename`：水平分割窗口，打开新文件
  - `:vsp filename`：垂直分割窗口，打开新文件
  - `Ctrl+w+w`：在多个分割窗口之间切换
  - `Ctrl+w+q`：关闭当前分割窗口

查找与替换

- `:%s/old/new/gc`：逐一确认替换，g表示全局（global），c表示确认（confirm）
- `:n1,n2s/old/new/g`：在n1~n2行之间替换
  尽管vim本身没有强大的多文件搜索功能，但能与外部命令结合使用
- `:grep '关键词' 文件名`：在vim中调用grep，搜索结果会加载到quickfix列表中
- `:copen`：打开quickfix列表，可在列表中跳转到搜索结果

此外，还可以实现连续行注释
`:n1,n2s/^/#/g`：将n1~n2行开头添加#
当然也可以用可视模式插入注释

宏
宏（Macro）是Vim中一个非常强大的功能，能录制一系列的按键操作，然后重复执行

关于宏的用法不多赘述
