Linux & Shell

无15 向明义

一、简介（课中略去）

1.1 发展历史

Unix是一种古老的操作系统（上世纪七十年代）。在Unix之前，操作系统基本都只能进行批处理作业，相当于你给计算机一批任务，然后计算机逐项去自动执行，执行完了反馈给你结果，在此过程中是不具可交互性的。为了满足交互性、及时性和多用户同时访问的需要，人们希望研制分时操作系统——通过时间片轮转为很多用户同时提供交互式服务。早期的尝试能支持的用户量很有限（MIT推动的MAC计划只能让30名左右的使用者同时共享IBM的大型计算机，到了1965年还最终不堪负荷），于是MIT决定研制更大型的分时计算机系统——MULTICS——其中通用电子公司参与了硬件开发，贝尔实验室承包了软件也就是操作系统。但是这一由众多大名鼎鼎的机构联合执行的计划最终失败了，MULTICS被传为笑柄。

但是，当时在贝尔实验室参与分时操作系统开发的成员在此项目中获得了宝贵的经验。贝尔实验室虽然退出了MULTICS项目，但为创建交互式计算环境的努力从未停止，后来汤姆森(Ken Thompson) 与里奇(Dennis Ritchie)在一些机缘巧合之下开发了Uniplexed Information and Computing System，被简称Unix。历经很多版本的改进（其中重大的改进如将内核从由汇编语言写改成由高级语言写，他们尝试过世界上第一个被正式推广使用的高级语言Fortran，后来又产生出了B语言，但是都不理想，最后改良了B语言，开发出了C语言，因而里奇被称作C语言之父。这一改进大大增强了Unix的多平台可移植性），人们发现Unix相比于批处理系统有很多优势，应用规模逐渐扩大，最终贝尔实验室的所有者AT&T公司宣布要将Unix商业化。由于众多衍生版的出现， 在随后的几十年中，Unix的发展经常伴随着产权纠纷。 目前它的商标权由国际开放标准组织所拥有。

20世纪80年代，随着硬件计算能力的提升，计算机市场不断扩大。可供选用的操作系统中，Unix是商业软件，AT&T制定的价格昂贵；DOS系统是单用户单任务的，且源代码被微软当作商业秘密； MacOS系统只能用于苹果电脑。这在教学上为老师提了难题，因为找不到开源的例子。一名教授就编写了一个操作系统Minix来讲解操作系统内部工作原理。虽然Minix比较简单，但好处是完全开源的。

全世界学计算机的学生都通过钻研Minix的源代码来学习操作系统，芬兰赫尔辛基大学大学二年级的学生Linus Torvalds就是其中一个。借鉴了Minix和Unix的思想，Linus在1991年写出了Linux操作系统。他把Linux的源码放到网上，所有人都可以免费获取，甚至可以进行商业应用。这就使得Linux迅速得到了全世界计算机企业和爱好者的支持，出现了很多发行版，用户可以根据自己的需要自由增添、剪裁Linux系统。这样以来，一个庞大的社区来不断更新和维护使Linux具有了长久的生命力。

1.2 Linux的主要特点

（1）Linux是开源的，遵循GNU通用公共许可证（GPL）。这意味着任何人都可以基于Linux底层代码创建一个符合自己需求的Linux 发行版。

（2）多用户、多任务。同Unix一样，Linux是分时操作系统，各个用户可以同时登陆，由操作系统来保证各用户对于各类文件有合适的权限。Linux系统可以同时连接多个终端并且每隔一段时间重新扫描进程，重新分配进程的优先级，动态分配系统资源，从而在多进程同时运行时保持良好性能。

（3）可操作性。Linux具有很强的可操作性，它将所有操作权都交给了用户，向用户暴露所有的细节。普通用户刚入手时可能对这些操作有些摸不着头脑，可能一个指令就把操作系统弄崩溃了。但对于开发者而言，Linux的开放使我们能清楚地看到程序是如何运行的，运行报错也会有友好的提示，根据报错指引往往能将问题解决。

（4）安全稳定高效。Linux用户权限分级严谨，不容易受到病毒和恶意软件的侵害，在长时间运行下稳定性和可靠性都非常高。

（5）多平台支持：Linux可以安装在各种硬件平台上。

1.3 Shell是什么?

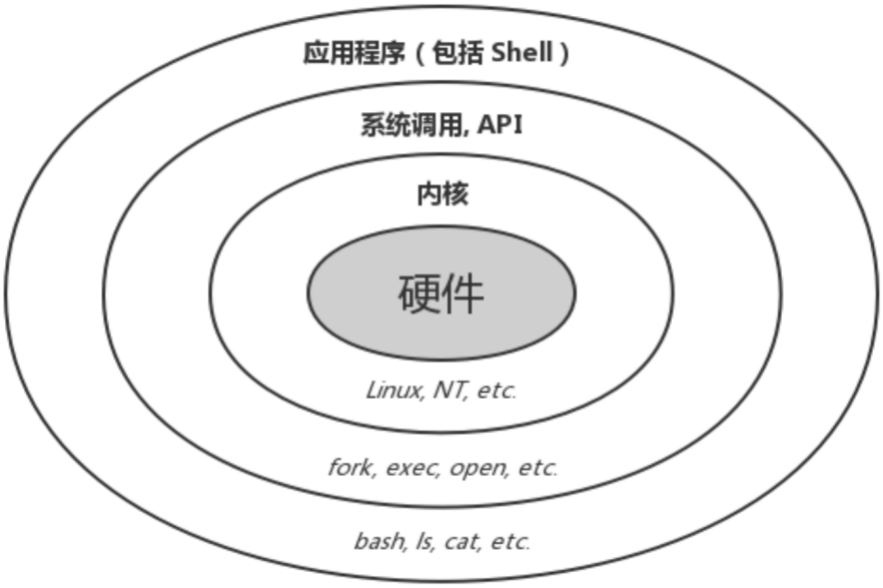
操作系统主要负责系统内存管理、软件程序管理、硬件设备管理和文件系统管理。在用户与操作系统之间需要应用程序作为桥梁，帮助用户与操作系统进行交互，实现各种功能。

应用程序与用户的交互有两种表现形式，一种是我们常见的 GUI（Graphical User Interface），即图形用户界面，可以用鼠标点击操作；另一种是 CLI（Command Line Interface），即命令行界面。

Terminal，终端，是人与计算机进行交互的接口。一台大型机可以连接很多终端机，用户在这些终端机上跟计算机进行交互，这些早期终端一般是电传打字机（TTY）。随着个人计算机的普及和进化，大型机时代的专门的终端硬件早已消失，我们只用自己的键盘、鼠标来控制计算机。但是终端的思想保留了下来，变成了操作系统中的软件——终端模拟器（如Win32控制台）。终端模拟器可以捕获你的键盘输入，发送至命令行程序，拿到输出结果后再调用图形接口，将输出渲染至屏幕上。由于人们除了在博物馆再也见不到TTY这样的终端硬件了，于是就直接称终端模拟器为终端。

Shell，即“壳层”，是帮助用户与内核交互的一类程序。操作系统内核直接负责管理计算机硬件，处于操作系统最底层，由于其重要性是不能轻易由用户直接操作的，因此需要有Shell程序来作为用户操作系统的入口。Shell是一类程序，可以有CLI形式的（如sh、bash、zsh；cmd、powershell等），也可以有GUI形式的（如Windows资源管理器explorer.exe等）。

用户所看见的界面通常是终端，在用户用键盘和鼠标输入后，终端（Terminal）将这些输入发送给你选用的壳层（Shell），Shell解析你的命令发送给操作系统内核去执行，并把执行结果返回给终端，终端调用图形接口将结果显示到屏幕上。



二、Linux文件操作

在windows中，你会看到这样的文件路径：

```
C:\Users\eesast\Documents\test.doc
```

这种路径表明了 `test.doc` 文件位于C盘，在目录 `\Users\eesast\Documents\` 下。

Linux的文件系统与windows的实现方式并不一样。Linux的文件结构为树状结构，它的文件路径中没有驱动器盘符（如 `C:` 或 `D:`），Linux采用虚拟目录的方式，将计算机所有存储设备的文件路径都纳入统一的单目录下。所有的文件与目录都被放在根目录 `/` 下。在Linux中你会看到这样的路径：

```
/home/ubuntu/Documents/test.doc
```

注：Linux使用正斜线(/)来分隔文件路径中的目录，而反斜线(\)用作转义字符。

2.1 查看与切换目录

首先说明一下 **绝对路径与相对路径** 的概念：

- 绝对路径

  绝对路径从根目录写起，以 `/` 为起始，相当于目录的全名，如 `/usr/bin`

- 相对路径

  相对路径从当前目录写起，以当前目录的子目录名或 `.` ， `..` 起始，单点号表示当前目录，双点号表示父目录。按照之前的例子，如果当前处在 `/home/ubuntu` 下，则 `./Documents` 或 `Documents` 与绝对路径 `/home/ubuntu/Documents` 等价。

另外，`~` 表示用户目录，它的绝对路径取决于你当前所使用的账户。

`pwd` 命令可打印当前目录的绝对路径。

列表命令 ls

`ls` 会列出当前目录下的文件和目录，按列以字母排序。

`ls -a` 会显示隐藏文件，`ls -l` 会显示文件的详细信息（大小、修改日期、权限等）。

```
ls [options] [directory name]
options -a 显示所有文件及目录(“.”开头的隐藏文件也会列出)
        -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
directory name可省略，表明显示该目录下的文件及目录
```

tips：多个命令选项可合并写，如 `ls -al`；使用 `man ls` 可查看命令的详细参数。

tips2：`tree` 命令可以以树状图（实际上是阶梯形）形式显示文件结构，系统可能未安装该命令，需按提示安装。

目录切换命令 cd

```
cd [destination]
```

目录切换命令 `cd destination` 用来切换到 `destination` 指定的目录。可以使用绝对路径或相对路径。不指定参数时，将切换到用户目录 `~`。

tips：按 `TAB` 键可以根据输入文件或目录名称前缀自动补全。

2.2 管理目录

创建目录 mkdir

```
mkdir [-p] directory_name
option -p 根据需要创建缺失的父目录
```

例:

```
ubuntu@VM-17-6-ubuntu:~$ mkdir test1/test2
mkdir: cannot create directory ‘test1/test2’: No such file or directory # 创建失败，test1不存在
ubuntu@VM-17-6-ubuntu:~$ mkdir -p test1/test2
ubuntu@VM-17-6-ubuntu:~$ tree test1
test1
└── test2
```

删除目录 rmdir

```
rmdir [-p] directory_name
```

为避免误删除， `rmdir` 只能删除空目录。要删除带文件的目录需要 `rm` 命令。

2.3 处理文件

创建文件

通常使用 `touch` 命令创建文件。

```
touch file_name
```

如果文件不存在，则 `touch` 命令会创建一个新的空文件。如果已存在，则该命令仅会将文件的修改时间改为当前时间。

也可以使用重定向符将空内容输入到指定文件的方式创建文件，关于重定向的内容稍后详述。

复制与移动文件 cp&mv

复制文件或目录使用 `cp` 命令，通常带两个参数 `source` 和 `destination`。会将源文件复制成一个新文件，并以 `destination` 命名。

```
cp [options] source destination
options
    -i 在覆盖已有文件之前询问是否覆盖
    -r 若给出的源文件是一个目录文件，将复制该目录下所有的子目录和文件
```

例:

```
cp file1 file2
cp -r dir1 dir2 # dir2不存在，此时会将dir1重命名
cp -r dir1 existdir # existdir存在，此时会将dir1放到existdir下
cp file1 existdir/[filename]
```

移动（或者说剪切）文件使用 `mv` 命令。

```
mv [options] source destination
options
    -i 执行覆盖前询问
    -n 不执行覆盖
    -u 当源文件比目标文件修改时间更晚或目标不存在时，才执行移动
```

删除文件或目录 rm

```
rm [options] [file or directory]
options
    -i 删除前询问
    -r 递归地删除所有子目录和文件
    -f 强制删除，不给出任何提示
    -v 显示详细的删除过程
```

shell没有回收站这样的东西，文件一旦删除无法恢复，所以 `rm -rf` 要谨慎使用。

2.4 文件显示与编辑

连接并显示文件内容 cat

```
cat [options] file...
options
    -n 加上行号
    -s 将连续两行以上的空白行替换为一行
如果file不止一个文件，则会将它们连接起来
```

cat命令会一次性显示文件的所有内容，如果想一次只看一页，可以使用 `more` 或 `less` 命令。

过滤文件内容 grep

grep命令可以查找拥有与给定正则表达式相匹配的内容的文件，如果发现匹配成功的文件，grep命令默认会把含有匹配字符串的那一行显示出来。如果没有指定文件，则grep命令会从标准输入设备（键盘）读取数据。

```
grep [options] [想要匹配的正则表达式] [文件或目录...]
options
    -A [n] 除了显示匹配的那一行之外，还显示该行之后的n行（after）
    -B [n] 除了显示匹配的那一行之外，还显示该行之前的n行（before）
    -C [n] 除了显示匹配的那一行之外，还显示该行之前和之后的n行（contest）
    -c 计算总匹配的行数
    -r 查找目录而非文件时，必须加上这一参数，否则会报错
    -e 使用正则表达式进行匹配，默认也是这样
    -E 使用拓展的正则表达式进行匹配
    -i 忽略字符大小写的差别
    -l 列出含有匹配成功内容的文件名
    -n 在显示匹配的行之前，标示出该行的行号
    -v 显示不包含匹配表达式的所有行，相当于反向选择
```

为了避免shell的保留关键字对正则表达式的影响，建议将正则表达式用单引号括起来，但不要用双引号括起来。

补充：正则表达式

表达式	含义
`^word`	搜寻以word开头的行
`word$`	搜寻以word结束的行
`.`	匹配任意一个字符
`\c`	转义\后面的特殊字符c，在正则表达式中有特殊含义的字符必须要先转义才能使用
`c*`	表示*前面的字符c可以重复0次到多次
`[list]`	匹配一系列字符中的一个
`[n1-n2]`	匹配一个字符范围，如[0-9]、[a-z]中的一个字符
`[^list]`	匹配一系列字符以外的字符
`\<word`	匹配以word开头的单词
`word\>`	匹配以word结尾的单词

拓展正则表达式的几个例子：

表达式	含义
c?	匹配0个或1个字符c，即c既可以出现也可以不出现
c+	表示+前面的字符c可以重复1次到多次，注意跟*有区别
|	表示“或”，匹配一组可选的字符或字符串

补充2 通配符

通配符	含义
`*`	匹配任意数量的任意字符
`?`	匹配 1 个 任意字符
`[□■]`	匹配 `□` 或 `■` （字符集中的任意一个字符）
`[^□■]`	匹配除 `□` 和 `■` （字符集中的任意一个字符）以外的任意一个字符
`[A-Z]`	匹配一个大写字母（小写字母、数字同理，也可取反）
`{□,■}`	匹配括号内 `□` 和 `■` (两个或多个字符串)

在 linux 许多与文件名相关的指令中，都可以使用通配符，大家可以自行尝试。

文档编辑器 vim

Linux系统中都有vim文本编辑器，其功能十分丰富，可以作为程序开发工具使用，虽然比较难以掌握。一般我们只要掌握其最基本的用法就够了。

vim分为三种模式：命令模式（Command mode），输入模式（Insert mode）和底线命令模式（Last line mode）。当我们用vim [file]打开一个文件时，就进入了命令模式，在这一模式下我们可以输入命令，比如摁下 `i`（代表insert）可以切换至输入模式，摁下 `:` 可切换至底线命令模式（可以输入更多命令）。

在输入模式当中，我们可以上下左右移动光标，用键盘在光标处进行输入等，基本上跟我们习惯的输入方式一样。如想退出，或者执行其他指令，要先摁 `ESC` 键退出输入模式到命令模式，再摁 `:` 键切换至底线命令模式。在底线命令模式中，最常用的命令是 `q`（退出）和 `wq`（保存并退出）。vim还可以进行快捷查找，如果对更多vim的使用方式感兴趣，可以上网查阅更多资料。

三、Linux用户管理

Linux是一个多用户系统，在多用户同时使用服务器时这一点体现尤为明显。每个用户账号都拥有一个唯一的用户名和密码。用户在登录时键入正确的用户名和密码后，会进入自己的主目录（通常是 `/home/用户名`，缩写为 `~`）。

用户对系统中各种文件的访问权限取决于所登录的账户，同时为了管理涉及共享资源的一组用户，采用了"用户组"的概念。每个用户有唯一的UID，每个组有唯一的GID。

3.1 用户账号的新建、修改、查看与删除

新建用户 useradd

```
sudo useradd [options] username
options
    -m 创建用户的主目录
    -d 指定用户的主目录（如果不使用默认的话）
    -g 指定用户登录组的GID或组名
    -p 指定默认密码
    -s 指定用户登录的shell
```

这一命令需要超级用户权限执行。Linux中，命令前加sudo表明用超级用户权限执行该命令，但要保证当前用户拥有超级用户权限。如果没有的话，可以用su命令切换至root用户操作。

例如，我们可以执行：

```
sudo useradd -m eesast
```

使用 `useradd -D` 可以查看不指定选项时的默认设置。

切换用户 su

```
su [username]
```

如果不加用户名，则默认切换到超级用户 root。当我们尝试切换到刚创建的用户时，会发现要求输入密码，而无论输入什么都是验证失败，因为我们还没有设密码，账户处于锁定状态。

修改密码 passwd

```
passwd [options] # 修改自己的密码，需要知道自己的现有密码
sudo passwd [options] username # 修改别人的密码，需要超级用户权限
```

通过options可以锁定密码（-l）、解锁密码（-u）、使账号无密码（无法登录，-d）、 强迫用户下次登录时修改密码（-f）等等，小型开发不常用。当我们为刚才新建的用户指定一个密码后，再切换用户就可以输入密码登录了。

修改用户属性 usermod

```
sudo usermod [options] username
```

修改已建立用户的属性，其选项同新建用户时一样。例如：

```
usermod -s /bin/bash eesast
```

此时再登录eesast时，我们会发现默认shell从sh切换到了bash。

查看用户 /etc/passwd

可以通过查看/etc/passwd文件来查看当前系统中的全部用户和属性：

```
cat /etc/passwd
```

每行用户信息都以 ":" 作为分隔符，划分为7个字段，每个字段所表示的含义如下：`用户名：密码（被隐藏）：UID（用户ID）：GID（组ID）：备注（在创建用户时或修改用户时添加）：主目录：默认Shell`。例如：

```
eesast:x:1001:1001::/home/eesast:/bin/bash
```

赋予用户超级用户权限

eesast用户此时还没有权限使用sudo命令，我们可以切换回root通过修改 `/etc/sudoers` 文件来设置用户权限。注意，该文件如果修改错误会产生严重的影响，因此不要直接用文本编辑器修改，而是用visudo工具进行修改。

```
sudo visudo
```

权限配置的格式如下：

```
root    ALL          = (ALL           :  ALL          )    ALL
用户名   允许登录的主机    可使用的用户身份    可使用的组          可使用的命令
%sudo   ALL=(ALL:ALL) ALL
```

你可以见到 `%admin` 和 `%sudo` 这样的名字，它们是组名。所以要给一个用户赋予sudo权限，可以直接将其添加到 sudo 组中。

```
sudo usermod -aG sudo eesast
```

/etc/sudoers中还有一些其他有趣的配置。例如，添加一行如下代码可以让下次输入sudo密码时不是什么都不显示，而是显示星号：

```
Defaults pwfeedback
```

删除用户 userdel

```
sudo userdel [options] username
options
    -r 同时删除用户主目录
```

3.2 用户组

一个组中可以有多个用户，一个用户也可以在多个组中。 当一个用户同时是多个组中的成员时，在/etc/passwd文件中记录的是用户所属的主组，也就是登录时所属的默认组，而其他组存储在/etc/group中，称为附加组。

查看用户组 /etc/group

```
cat /etc/group
```

每行用户组信息都以 ":" 作为分隔符，划分为4个字段，每个字段所表示的含义如下：

组名 : 用户组密码（一般没有密码） : GID : 组内用户列表（如果某用户的主组是该组，该用户不会在这里显示）。

用户组操作

- 新建用户组：`groupadd`
- 修改用户组：`groupmod`
- 删除用户组：`groupdel`
- 切换用户组：`newgrp`
- 查看用户所属的组：`groups username`

如果想将某个用户添加到某个组，可以使用 `usermod -aG groupname username`；想将某个用户从指定的组中删除，可以使用 `gpasswd -d username groupname`。

3.3 用户/用户组权限

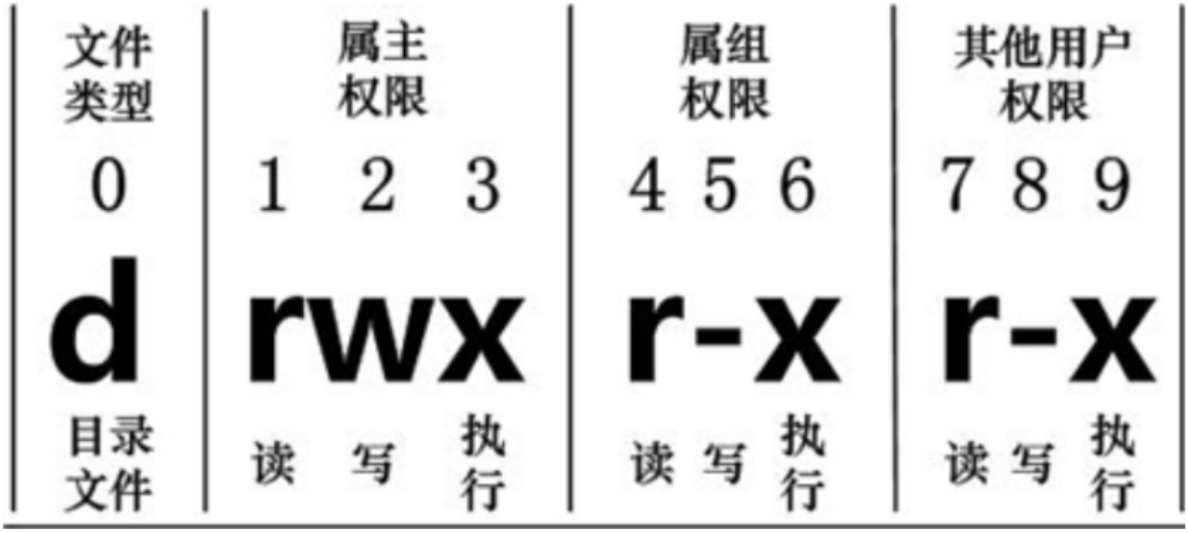
查看文件权限

```
ll (约等于 ls -al) 或 ls -l
```

会得到若干条这样的记录：

```
drwxr-xr-x  3 root   root   4096 May 18  2022 ../
```

第一段的 `drwxr-xr-x` 表示文件类型及权限：



文件类型常见的有d（目录）、-（文件）、l（链接）。接下来以三个字母为一组表明了文件权限：属主权限指的是文件所有者对该文件的权限， 属组权限指的是文件所有者的同组用户对该文件的权限，其他用户权限顾名思义。r、w、x分别代表读、写和执行的权限，如果没有此项权限则会出现减号。

第二段的 `3` 表示有三个硬链接指向该文件对应的节点。关于软硬链接的内容这里略去，感兴趣的同学可以参见该教程的笔记部分：Linux 文件与目录管理 | 菜鸟教程 (runoob.com)

第三段的 root 表示文件属主，第四段的 root 表示文件属组，4096为文件大小，其后是修改日期，和文件名

修改文件权限 chmod

```
chmod [-R] [u/g/o/a][+/-/=][r/w/x] [file]
```

其中u（user）代表改变属主的权限，g（group）代表改变属组的权限，o（owner）代表改变其他用户的权限，a（all）代表改变所有用户的权限。+代表增加权限，-代表删除已有权限，=代表设定权限。r/w/x代表读、写、执行。如果加上-R，则表明要递归修改权限，如果文件参数是一个目录，就会将该目录下的所有文件都修改。这些参数可以灵活地组合使用，下面举两个例子。

(1) 为所有用户赋予1.txt的执行权限

```
chmod a+x 1.txt
```

(2) 为属主赋予读、写、执行权限，删除属组和其他用户的执行权限

```
chmod u=rwx,g-x,o-x 1.txt
```

除了通过这种符号方法，还可以根据编码来决定如何修改权限。编码方法可以更简洁地修改文件权限。表示方法是为每个权限赋予权重，读的权重为4，写的权重为2，执行的权重为1。在表示每类用户的权限时，把该类用户所拥有的全部权限的权重相加，得到一个0-7的数，再按属主、属组、其他用户的顺序排列起来，得到一个三位数，表示该文件的权限。数字方法只能直接设定文件权限，不能增添或者删除权限。

例如，为所有用户赋予所有权限，用编码方法表示为：

```
chmod 777 1.txt
```

修改文件属主和属组 chown

```
sudo chown [-R] [owner] [file]
sudo chown [-R] [owner]:[group] [file]
```

-R表示对目录递归地进行修改。这一命令只能用超级用户权限执行，哪怕你是该文件的所有者且拥有全部权限，也不能作为普通用户修改文件的属主和属组。

也可以通过 `chgrp` 来单独修改属组。

四、Linux系统与磁盘管理

4.1 磁盘管理

常用的命令有三个：

- `df` （disk free）：列出文件系统的整体磁盘使用量
- `du` （disk used）：列出文件或目录的磁盘空间使用量
- `fdisk`：用于磁盘分区

```
df [options] [file or directory path]
options
    -h 以人们较易阅读的 GB, MB, KB 等格式自行显示
    -a 列出所有的文件系统
    -T 显示文件系统格式
df命令会把file or directory path所在的文件系统的磁盘用量展示出来
```

```
du [options] [file or directory path]
options
    -h 以人们较易读的容量格式 (G/M) 显示
    -a 列出所有的文件与目录容量，因为不指定文件时默认仅统计目录的容量
    -s 列出总量，而不列出每个目录占用容量
du命令会把file or directory path自身占用的磁盘空间大小展示出来，不指定时默认为当前目录
```

```
sudo fdisk [option] [device name]
option -l 可以把整个系统内能够搜寻到的装置的分区均列出来
```

一般使用时，先通过df命令找出某个文件\目录所在的磁盘名称，然后再使用fdisk [找到的磁盘名称]对该磁盘进行分区管理。操作时，fdisk工具会有一步步的提示，只需跟着提示去做就可以了。离开时，按q不会保存修改；而按w会保存修改。

磁盘格式化

```
mkfs [option] [device name]
option: -t [filesystem] 将指定的磁盘格式化为指定文件系统格式
```

磁盘挂载与卸载

```
mount [device name] [directory] # 将文件系统挂载到某个目录下
umount [device name or directory] # 将某个文件系统或者某个目录挂载的文件系统卸载
```

4.2 系统管理

查看CPU和内存占用 top

使用 `top` 命令，可以进入一个定时刷新的动态显示界面，刷新频率可以由参数指定。

查看进程 ps (process status)

```
ps [options]
options -a 列出所有的进程（包括其他用户的）
        -u 显示较详细的信息
        -x 也显示没有控制终端的进程
```

终止进程 kill

kill指令用于向进程发送信号，最终如何响应由该进程决定。一般用作“杀死”一个进程，但无法“杀死”系统进程和守护进程。

```
kill -l # 列出全部可以发送的信号名称和编号
```

```
kill [options] [PID]
options -s [要发送的信号名称或编号] # 如不指定该项参数，默认发SIGTERM(15)，一般可终止进程
        -u [用户名] 向指定用户的所有进程发送信号
```

一般在终止一个进程时，先用ps查询该进程的PID，然后用kill [PID]即可；如发现该进程不能正常终止，可以用 `kill -s 9 [PID]` 或 `kill -s KILL [PID]` 或者省略s，用kill -9/KILL强制终止进程。但要注意，所谓“强制”只是向进程发送了一个不同的信号，并不代表一定能终止进程，一般都是可以的。

4.3 软件安装

在Debian和Ubuntu发行版当中，apt是一个软件包管理工具。我们可以通过它方便地实现软件包的安装、更新、删除等操作。运行apt需要超级用户权限。

软件安装

```
sudo apt install [package name...] # 可以一次性安装多个软件包
```

软件更新

```
sudo apt update # 列出所有可更新的软件清单
sudo apt upgrade # 升级全部可升级的软件包
sudo apt upgrade [package name...] # 指定升级软件包
sudo apt full-upgrade # 升级全部可升级的软件包，并且在升级前将旧的包先卸载
```

软件查询

```
apt list --installed # 列出所有已安装的软件包
apt list --all-versions # 列出所有已安装的软件包的版本信息
sudo apt show [package name] # 显示指定软件包的具体信息
sudo apt search [keyword] # 查找可供安装的软件包
```

软件卸载

```
sudo apt remove [package name] # 卸载软件包
sudo apt autoremove # 清理不再使用的依赖和库文件
```

除了apt工具外，还可以使用snap工具，从源代码安装等方式。

五、其他常用Shell命令

5.1 查找文件 find

```
find [path] [options]
path是要查找的目录路径，可以是多个，未指定则默认为当前目录
options
    -name [name]  按文件名查找，支持通配符，-iname会忽略大小写
    -type [type]  按文件类型查找，f(普通文件),d(目录)等
    -path [p] 路径符合 p 的文件
    -amin [-n] 在过去 n 分钟内被读取过的文件
    -size [+nM] 大小大于 n M的文件
```

5.2 输入输出重定向

输入重定向

格式	作用
`command < file`	将指定的文件内容作为命令的输入
`command << string`	从标准输入设备（键盘）中读入，直到遇到string才停止

输出重定向

格式	作用
`command > file`	将命令执行的标准输出结果(stdout)重定向输出到指定的文件中，如果该文件已包含数据，会清空原有数据，再写入新数据。
`command 2> file`	将命令执行的错误输出结果(stderr)重定向到指定的文件中，如果该文件中已包含数据，会清空原有数据，再写入新数据。
`command >> file`	将命令执行的标准输出结果重定向输出到指定的文件中，如果该文件已包含数据，新数据将写入到原有内容的后面。
`command 2>> file`	将命令执行的错误输出结果重定向到指定的文件中，如果该文件中已包含数据，新数据将写入到原有内容的后面。
`command &> file`	`&`表示将stderr和stdout同时重定向
`command 2>&1`	`>&1` 表示将前面的输出重定向到文件描述符1所指向的文件，这里 `2>&1` 表示将stderr重定向到stdout

```
ubuntu@VM-17-6-ubuntu:~$ cd redirect/
ubuntu@VM-17-6-ubuntu:~/redirect$ touch 1.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ touch 2.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ touch 3.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ ls > 1.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ cat 1.txt
1.txt
2.txt
3.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ ls >> 1.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ cat 1.txt
1.txt
2.txt
3.txt
1.txt
2.txt
3.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ ls abs 2> 1.txt
ubuntu@VM-17-6-ubuntu:~/redirect$ cat 1.txt
ls: cannot access 'abs': No such file or directory
```

```
ubuntu@VM-17-6-ubuntu:~/redirect$ ls abc >> 1.txt 2>&1
ubuntu@VM-17-6-ubuntu:~/redirect$ cat 1.txt
ls: cannot access 'abs': No such file or directory
ls: cannot access 'abc': No such file or directory
```

5.3 管道

管道 (pipe)，就是将两个或者多个命令连接到一起，把一个命令的输出作为下一个命令的输入。 Linux 管道使用竖线 `|` ，即管道符连接多个命令。 语法格式如下：

```
[command1] | [command2] | [command3] ...
```

此时，管道符 `|` 左边命令的正确输出就变成了右边命令的输入，后续命令无法处理之前命令的错误输出。

practice：查看某文件是否处于当前目录下

之前讲过的find命令可以完成：`find . -name filename`

如果使用管道，可以利用 `ls` 命令的输出结果，使用 `grep` 过滤内容：

```
ls | grep filename
```

我们把ls命令的输出，也就是当前目录下的文件列表（不包含隐藏文件），送入“管道”作为grep命令的输入，之后grep命令匹配某文件名是否存在于文件列表当中，这样就达到了我们的目的。

一句话计算 1+2+...+100

```
echo {1..100} | tr ' ' '+' | bc
```

六、Shell脚本

Shell脚本是一种在Shell中运行的脚本程序，由Shell作为脚本解释器，Shell脚本的一行就相当于一个命令，也可以用分号分隔一行之中的不同命令。

6.1 shell脚本基本语法

注释

```
# hello!
```

变量

定义变量时，注意**变量名和等号间不允许有空格。**变量名也有限制，如只能包含英文字母、数字和下划线且不能以数字开头等。如：

```
a=1
```

引用变量时，在变量名前加 `$` 符号，如：

```
echo $a
```

查看环境变量：

```
echo $HOME
```

删除变量用 `unset` 命令：

```
unset a
```

数组

定义数组

```
array=(value0 value1 value2 value3)
```

读取数组元素：

```
echo ${array[0]}
```

使用 @ 符号可以获取数组中的所有元素：

```
echo ${array[@]}
```

分支

```
if condition1 then
    command1
elif condition2 then
    command2
else
    commandN
fi
```

由于shell的语法解析不太鲁棒，所以condition的表达式比较特殊，具体可以自行搜索或参考 Linux中if语句用法总结_linux if_hyfstyle的博客

循环

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

仿 C 风格的 for 循环：

```
for (( i = 1; i <= 10; i++ )) # 需要使用双括号，变量定义可以有空格
```

```
while condition
do
    command
done
```

在循环中，也可以跟C语言一样使用break和continue。

6.2 Shell函数

```
function test(){
    echo "参数1： ${1} !" # 通过${n}来获取传入函数的第n个参数
    echo "参数10： ${10} !" # 双引号括起来的字符串中可以引用变量
    n=$((${1}+${10}))
    return $n # 如果不加return，则返回值为最后一条命令的结果
}
test 1 2 3 4 5 6 7 8 9 10 # 调用函数时依序传递参数
echo $? # 可以通过变量$?来获取函数的返回值或者上一条命令的结果
```

6.3 运行Shell脚本

在确保脚本具有可执行权限的前提下，可以直接执行脚本。

```
./test.sh # 注意不能写成test.sh，否则会去环境变量中查找
```

在运行shell脚本时可以跟函数调用一样传递参数。

作为解释器参数运行shell脚本：

```
bash test.sh # 参数是脚本路径
```

七、Shell环境变量

环境变量是存储有关 shell 的工作环境的数据，以便程序或 shell 中运行的脚本能够访问到它们。系统会创建好一些环境变量，我们在安装新程序等操作时也要留意是否需要设置环境变量，否则会出现“command not found”的错误。

7.1 查看环境变量

```
env # 查看全部全局变量
printenv HOME # 查看个别全局变量（HOME）
echo $HOME # 查看个别全局变量（HOME）
```

常用环境变量

- `HOME`：当前用户的主目录
- `PATH`： shell 查找命令的目录列表，由冒号分隔
- `PWD`：当前工作目录

7.2 设置环境变量

可通过将局部变量导出到环境当中来设置环境变量，在该进程及其所创建的子进程中，该环境变量都是可见的，且子进程中的修改不会作用到父进程的环境变量。但是，这种设置是临时的，一旦退出当前shell就消失了。

```
variable="test"
export variable # 导出环境变量
unset variable # 删除环境变量
```

如想设置持久的环境变量，就需要修改系统启动时用于加载环境变量的文件。这些文件各自有不同功能，首先，/etc目录下的文件是系统级的，所有用户的终端都会加载；而~/也就是用户主目录下的文件是用户级的，只有该用户的终端会加载。其次，profile文件在交互式登录shell当中加载，而bashrc文件在交互式非登录shell当中加载，不过由于profile文件当中引用了bashrc文件中的环境变量，因此也可认为在登录shell当中将二者都加载了。所谓登录就是指“用户登录到系统的shell”，而其他的都是非登录shell。

因此，一般我们修改的文件有以下四个（其实绝大多数情况下修改~/.bashrc就够了）：

```
/etc/profile # 系统级，登录shell当中加载
/etc/bashrc # 系统级，可认为都会被加载
~/.profile # 用户级，登录shell当中加载
~/.bashrc # 用户级，可认为都会被加载
```

修改PATH环境变量时，一般采用追加导出的方式，即在文件的最后一行加上：

```
export PATH=/usr/local/src/mongodb/bin:$PATH
```

如果想不重启终端使设置的环境变量立即生效，可以用source命令。source命令本意是在当前shell中依次执行一个文件中的每行命令，而不像sh file name与./file name 那样会建立一个子shell。通过执行profile或者bashrc中的全部命令，就重置了当前shell的全部环境变量。

```
source [file name]
```

八、SSH超简介

SSH = Secure Shell，即安全外壳协议，是为远程登录会话提供安全性的协议，一般用作远程登录到服务器。有关SSH的通信过程和加密原理在此就不做介绍了，感兴趣的同学可以查找网络资料。

```
ssh user@hostname # 主机名可以是IP地址或者域名
```

Reference

Linux & Shell | EESΛST Docs (eesast.com)

《Linux命令行与shell脚本编程大全》