1. Linux

```
Ctrl+c：终止程序
Ctrl+z：挂起程序
Ctrl+d：发送EOF结束输入
```

1.1. 文件管理

1.1.1. 常用文件操作

```
ls # 列出目录
-a：all。显示隐藏文件
-A：-a除了当前目录.和上级目录..
-l：显示详细信息
-d：directory。只显示目录本身，而不是目录里的数据
-F：在最后面显示文件类别

cd # 切换目录
~：用户home目录
..：上级目录

pwd # 显示当前绝对路径
-P：显示链接指向的真实路径

mkdir # 创建目录
-p：parent。创建多级目录，如a/b/c
-m：mode。指定新目录的权限，而不是使用默认的权限

cp # 复制文件/目录
-r：recursive。递归复制目录
-i：interactive。当目标存在时提示是否覆盖
-p：preserve。保留文件的属性一起复制过去(备份时常用)

mv # 移动文件/目录
-i：当目标存在时提示是否覆盖

rm # 删除文件/目录
-r：递归删除目录
-f：force。文件不存在时不提示
-i：删除确认
```

1.1.2. touch

```
touch file # 修改file的时间属性，若file不存在则新建文件
-c：若file不存在，不新建文件
```

1.1.3. ln

```
ln source link # 给source创建硬链接link
-s：soft。创建软链接

unlink link：删除链接
```

1.1.4. chmod

```
-R：递归处理子目录

# 更改文件所属组，但该组必须是用户所在组
chgrp group file

# 更改文件的所有者及所属组，需要root权限
chown user:group file
chown user file # 只改变所有者
chown :group file # 只改变所属组

# 更改文件权限
chmod mode file
# [ugoa][+-=][rwx]
chmod u+w,g-x,o= file # 符号模式，a可省略
chmod 777 file # 数字模式
```

1.1.5. find

```
# 基本用法：find path -size +2k -a -type f -ok rm -rf {} \;
path：查找的路径，默认为当前目录

-name name # 根据文件名查找
-iname # 忽略大小写

-size +2k # 根据文件大小查找
2k：大小为2KB
-：比2KB小
```

```
+: 比2KB大
b: 默认单位，512B
c: Byte
k: KB
M: MB
G: GB

-type f # 根据文件类型查找
d: 目录
f: 文件
l: 软链接

-atime 10 # 访问时间，单位天
-mtime # 数据修改时间
-ctime # 状态修改时间
# 还有对应的-amin、-mmin、-cmin单位是分钟
10: 第10~11天之间
-10: 第10天以内
+10: 第11天之前

-perm 642 # 根据权限查找
642: 权限为642
-642: 权限完全包含642
+642: 三组权限任意一组包含即可，如711、142都满足

-user # 根据用户名查找
-uid # 根据用户ID查找
-group # 根据用户组查找
-gid # 根据用户组ID查找
-nouser # 查找没有所有者的文件，比如U盘里面的文件或网络上下载的文件
# 一般-nouser可以用来查找垃圾文件

# 逻辑运算
-a: and。与
-o: or。或
-not: 非
!: 非
find ! -name test # 注意!的前后都要有空格

# find查找到目标后的默认动作是-print，即把目标打印出来
```

```
-exec cmd {} \; # 执行cmd命令，将目标以{}作为参数传递。注意一定要以\;结尾
-ok cmd {} \; # 与-exec不同的是-ok在对每个目标执行操作前都会进行确认
```

1.1.6. gzip / zip / unzip

```
gzip file # 压缩成.gz，原文件会被删除
-r: 递归压缩目录
-d: decompress。解压缩.gz文件
-l: 查看被压缩文件的相关信息

zip target source # 压缩成.zip，原文件不会被删除
-r: 递归压缩目录
-d: delete。从.zip文件中移除文件
zip target -d source # 注意.zip文件要放在前面

unzip file # 解压缩.zip文件
```

1.1.7. tar

```
tar -zxvf target source # 打包压缩成.tar.gz格式
tar -zxvf file # 解压缩.tar.gz文件
tar -ztvf file # 查看压缩文件里的内容
-z: 使用gzip、zip进行压缩
-v: 显示执行过程
-f: file。要压缩/解压的文件
-c: create。创建压缩文件
-x: extract。解压文件
-t: 列出压缩文件内容
```

1.1.8. rcp / scp

```
rcp user@host:source user@host:target # 远程复制文件
-r: recursion。递归目录
-p: preserve。保留原文件的属性

scp user@host:source user@host:target # scp是rcp的加强版，传输是加密的
-P: port。指定scp所开放的端口(注意是大写的P，小写的p是保留原文件的属性)
```

1.1.9. rsync

```
# rsync用来进行远程同步，可以替代cp和mv命令
# rsync仅传输差异部分，因此效率比scp要高
rsync source target # 将source同步到target目录里，形成target/source
rsync source/ target # 将source里的内容同步到target目录里，不会在target里创建source目录
-r：递归同步子目录
-a：archive。同步一些属性，一般用-a而不是-r
-v：verbose。显示变动情况
--delete：删除target中存在而source中不存在的内容，形成source的一个镜像
--exclude：排除部分文件不同步
--exclude='.*' # 排除以.开头的隐藏文件
--exclude={'*.txt','*.jpg'} # 排除多种模式
--exclude-from='exclude-file' # 排除文件里的每一行模式
--include：与--exclude一起使用，将--exclude排除的一些文件重新包含进来进行同步
--link-dest：指定基准目录，当文件存在基准目录中时在target创建其硬链接，否则才进行同步
```

1.2. 文件内容管理

1.2.1. 查看文件内容

```
cat # 从第一行开始查看文件所有内容
-b：列出非空行的行号
-n：列出所有行的行号
-E：显示行尾$
-T：显示tab符^I
-A：显示所有字符

tac # 从最后一行开始倒序显示(行内不是倒序)，用法和cat一样

nl # 等价于cat -b
-w：行号所占用的字符宽度

more # 按页查看文件
<enter>：向下翻一行
<space>：向下翻一页
b：向上翻一页
```

```
q: 退出
/word: 查找字符串
:f: 显示当前行号

less # 用法包含more的功能
?word: 向上查找字符串
n: 重复搜索下一个(取决于/向下或?向上)
N: 重复搜索上一个

# less和more的区别
# 1. less可以通过方向键向上或向下翻一行而more不能
# 2. less退出后不会再终端显示内容而more会
# 3. less不必读取整个文件，加载速度会更快

head # 查看开头的几行内容
-n: 设置查看的行数

tail # 查看结尾的几行内容
-n: 设置查看的行数
-f: 动态监测文件末尾的内容(查看日志时常用)
```

1.2.2. grep

```
# 基本语法: grep [option] exp files
--color: 高亮显示
-E: Extended。使用扩展正则表达式
-i: ignore。忽略大小写
-n: number。显示行号
-c: count。统计出现的行数(不是次数)
-B1: Before。显示匹配行的同时显示前面一行
-A1: After。显示匹配行的同时显示后面一行
-C1: Context。显示匹配行的同时显示前面和后面一行
-o: only-matching。只打印匹配到的关键字，若同一行有多个匹配，则会分别打印多行
-w: word。匹配整个单词
-v: invert。反向查找，即查找不存在匹配的行

# 只想知道是否有匹配，而不关心在哪匹配
grep -q exp file
echo $?
# 当有匹配时返回0，没有匹配时返回1
```

1.2.3. wc

```
wc file # 统计file的行数、单词数、字节数
-l: line。只显示行数
-w: word。只显示单词数
-c: 只显示字节数
```

1.2.4. diff

```
diff file1 file2 # 比较两个文件的差异。若是目录，则比较目录下同名文件的差异，默认不会递归子目录
-i: ignore。忽略大小写
-r: 递归比较子目录中的文件
-y: 并排显示两个文件的异同
-W: width。在-y时指定每一栏的宽度
--left-column: 在-y时内容相同的行只显示左边一栏
--suppress-common-lines: 在-y时不显示相同的行

diff <(cmd1) <(cmd2) # 比较两个命令输出的差异，<是重定向符
```

1.2.5. sed

```
# 基本语法: sed [option] [-e script] [-f script_file] files
# 其中在命令中直接输入script时-e选项可以省略
# 默认会输出处理后的所有行，如果不想要默认输出则使用-n选项
a: 在指定行后面追加一行
i: 在指定行前面插入一行
d: 删除指定行
c: 替换指定行
s: 替换指定字符串
p: 打印指定行。一般与-n一起使用，否则除了打印指定行之外还会默认打印所有行

sed '4a newline' file # 在第4行后面添加一行，内容为newline
sed '4a \ ' file # 添加只有一个空格的行
sed '4a \\' file # 添加一个空白行
sed '3,5d' file # 删除第3-5行
sed '2,$c newlines' file # 将第2行到最后一行替换为newlines一行
sed 's/ev/op/g' file # 将ev替换为op，如果最后面没有g，则只替换每一行的第一个匹配项
sed -n '/^ev/p' file # 打印以ev开头的行
```

```
sed -n '/ev/{s/ev/op/g;p}' file # 将ev替换为op并输出修改的行

-i：inplace。sed默认只输出修改后的内容而不会修改原文件，加上该选项后可修改原文件
sed -e '2d' -e '5a newline' file # 依次执行多条命令
```

1.2.6. tee

```
tee file # 将标准输入的内容写入文件，同时输出到标准输出中
-a：append。追加模式
```

1.3. 常用命令

1.3.1. echo

```
-n：newline。不输出换行，默认是会换行的
-e：escapes。解析转义字符和彩色输出，否则"\n"之类的特殊字符会直接按字面输出
echo "`ll`" # 加上""按原命的输出格式进行输出

# 双引号""和单引号''的区别
echo "test $a" # 可以解析变量
echo 'test $a' # $a会按字面输出

# 转义字符
\n：换行
\r：回车到行首进行覆盖输出
\t：制表符
\v：垂直制表符，换行并接着上一行末尾处输出
\f：同\v
\b：退格。当后面存在字符的情况下会删除\b之前的那个字符，可连续使用多个\b删除前面的多个字符
\c：忽略后面的字符，且前面的字符不换行输出
\\：输出\本身

# 彩色代码
\033[xx;yymstring\033[0m # xx和yy是任意表示前景色(字体颜色)或背景色的编码
前景色：30-黑色；31-红色；32-绿色；33-黄色；34-蓝色；35-紫色；36-浅蓝色；37-灰色
背景色：40-黑色；41-红色；42-绿色；44-黄色；44-蓝色；45-紫色；46-浅蓝色；47-灰色
```

```
其它：1-高亮；3-斜体；4-下划线；9-删除线
      0-复位(0可省略)；5-闪烁；7-反色(前背景互换)；8-隐身(字体颜色设为背景色)
```

1.3.2. printf

```
# 基本语法：print format input
# printf默认不会换行

# 格式替换字符
%d：十进制整数
%f：浮点数
%s：字符串
%c：单个字符
%o：无符号八进制整数
%u：无符号十进制整数
%x：无符号十六进制整数，a-f表示10-15
%X：同上，A-F表示10-15
%%：%本身

%5s：指定字符串宽度为5，默认右对齐
%-5s：加上符号表示左对齐
%+d：当后面的数是整数时显示正号
%.3f：浮点数保留3位小数
%12.5d：表示占12位宽，整数长度为5，不足时前面补0
```

1.3.3. seq

```
seq end # 输出1到end之间的整数
seq start end # 输出start到end之间的整数
seq start incr end # 输出从start到end之间，从start开始增量为incr的所有整数
-s：separator。指定换行符
-f：format。使用printf中浮点数的格式化符号
-w：width。在前面添0使得所有的输出等宽
```

1.3.4. xargs

```
# xargs捕获命令的输出，传递给另一个命令。所有的换行和空白字符经过xargs都会变成<space>
# 之所以用xargs，是因为有很多命令不支持用管道来传参数，而用xargs就可以完成这个任务
cmd | xargs cmd # xargs后面默认的命令是echo
-n：指定一次使用多少个参数，默认是所有的参数
-d：指定分隔符，xargs解析输入所使用的分隔符
-I {}：指定用{}来替代参数，每次传递一个参数
```

1.3.5. & / jobs / bg / fg / nohup

```
cmd & # 将程序放在后台运行
Ctrl+z # 将程序挂起
jobs # 查看后台运行的程序，每个后台程序都有一个编号
jobs -l # 额外列出后台程序的PID
bg 1 # 将挂起的1号后台程序放在后台继续执行
fg 1 # 将1号后台程序放到前台执行

nohup cmd & # 将程序放在后台执行，但不挂起
# 默认会在当前文件夹创建一个nohup.out文件保存输出，可重定向到其他文件
# 若要结束程序运行，用ps指令找到PID再kill
```

1.4. 系统管理

1.4.1. ps

```
ps # process status。显示进程的状态信息。ps命令参数很多，只记录几个常用命令
ps -ef | grep python # 查看在运行的Python进程
ps -aux # 和-ef是两种不同的风格，都是查看进程的详细信息
ps -u user # 查看某个用户的进程

# 可以根据ps第一行显示的字段名进行排序
ps aux --sort=-%cpu # 根据CPU使用量降序排列，-改为+就是升序排列
ps aux --sort=+rss # 根据内存升序排列
```

1.4.2. top

```
top # 实时显示进程的动态
-c：显示完整的程序路径
-d：设置更新的周期，单位秒
-p：PID。指定显示某个进程

# 以下为top显示界面输入的指令
u：输入用户名显示指定用户的进程
```

1.4.3. free

```
free # 显示内存使用情况
-h：以合适的单位显示 # 类似的还有-b -k -m等
-s：以该周期持续监控
```

1.4.4. df / du

```
df # disk free。列出磁盘空间使用情况
-h：以易读的单位显示
-k：以KB单位显示
-m：以MB单位显示

du file # disk used。列出文件/目录占用空间的情况
-a：all。将文件的占用空间也显示出来，默认只显示目录的
-s：summary。只显示使用总量，而不显示各个子目录的使用量
-d：--max-depth。显示最大子目录深度，最顶级目录深度为0
```

1.4.5. lsof

```
lsof # list opened files。列出系统当前打开的文件，包括特殊文件、网络文件等
lsof file # 查看打开file的进程
-p：PID。指定进程号，查看指定进程打开的文件
-u：user。指定用户名，查看指定用户打开的文件
-c：cmd。查找运行命令包含cmd的项
-i：指定使用的协议、端口等信息
lsof -i:8080 # 查看占用8080端口的进程
```

1.4.6. 用户(组)管理

```
useradd user # 添加账号
-d: directory。指定主目录。与-m配合使用创建主目录
-g: group。指定所属主组
-G: 指定所属附加组
-s: shell。指定登录的shell
-u: UID。指定用户ID

userdel user # 删除用户
-r: remove。同时删除用户的主目录

usermod user # 修改用户，用法和useradd一样
-l: login。修改用户名

passwd user # 修改用户密码，user省略时为当前用户，普通用户只可修改自己的密码
-l: lock。锁定密码
-u: unlock。解锁密码
-d: delete。删除密码
-e: expire。强制用户下次登录时修改密码

groupadd group # 创建用户组
-g: GID。指定组ID

groupdel group # 删除用户组

groupmod group # 修改用户组，用法同groupadd
-n: new。修改用户组名

newgrp group # 用户登录后默认在主组中，次命令切换到附加组
```

1.4.7. shutdown

```
# 关机、重启
shutdown -h now/10/22:05 # 立即/10分钟后/在22:05关机
shutdown -r now/10/22:05 # 立即/10分钟后/在22:05重启
poweroff # 关机
reboot # 重启
init 0 # 关机
init 6 # 重启
shutdown -c # 取消定时关机/重启
```

1.5. 系统工具

1.5.1. crontab

```
crontab # 定时任务管理工具
-e: edit。编辑定时任务
# 添加环境变量 export VISUAL=vim 将使用vim编辑
-l: list。查看定时任务
-r: remove。删除定时任务

# 定时任务书写格式
min hour day mon week cmd # 分钟 小时 天 月 星期。注意星期是0-6
*: 表示每分钟(其它单位类似)
*/n: 每n分钟
a-b: 第a-b分钟
a,b,c: 第a,b,c分钟
a-b/n: 第a-b分钟之间每n分钟
```

1.5.2. alias

```
alias # 查看所有的别名
alias newcmd="cmd" # 为cmd起个别名

unalias newcmd # 删除别名
unalias -a # 删除所有别名
```

1.5.3. date

```
date format # 格式化输出日期和时间
%Y-年 %m-月 %d-日 %H-时 %M-分 %S-秒
%a(A): 星期的缩写(全称)
%b(B): 月份的缩放(全称)
%s: 自 1970-01-01 00:00:00 UTC 到现在的秒数
```

```
%n: 换行符
%%: %本身
date +"%Y/%m/%d %H:%M:%S" # 若要省略前缀0，则在相应的字段前面加一杠'-'
# 格式前面加一个加号+表示显示时间，否则为设定时间，设定时间的格式为mmddHHMM[[CC]yy][.SS]。注意设置时间后要使用clock -w写入，这样下次开机后才会保持这次修改

-d: 显示字符串表示的时间
date -d '2022/11/11' +"%Y-%m-%d" # 时间格式转化
date -d@1234567890 +"%Y-%m-%d" # 自 1970-01-01 00:00:00 UTC 到现在的秒数所表示的时间
date -d "1 day ago" +"%Y-%m-%d" # 1天前
date -d "-1 year" +"%Y-%m-%d" # 1年前
date -d "1 month" +"%Y-%m-%d" # 1个月后
```

1.5.4. sleep

```
sleep 5s # 睡眠5秒
s-秒 m-分 h-时 d-天
```

1.5.5. which / whereis

```
which cmd # 在$PATH中查找命令的位置，有多条结果时输出第一条
-a: all。有多条结果时输出所有

whereis cmd # 查找命令的二进制文件、源代码文件、帮助文件
-b: binary。只查找二进制文件
-s: source。只查找源代码文件
-m: manuscript。只查找帮助文件
```

1.5.6. who / whoami

```
who # 查看有哪些终端在使用该Linux系统
-H: heading。显示标题栏，即每个字段的含义
-m: 显示当前终端信息，与who am i等价

whoami # 显示当前用户的名称
```

1.5.7. su

```
su # switch user。切换用户，不带参数则切换为root
su - user # 切换用户并切换目录到新用户的主目录
```

1.6. vim

1.6.1. 命令模式

```
# 移动光标
h：向左移动一个字符
j：向下移动一个字符
k：向上移动一个字符
l：向右移动一个字符
3l：向右移动3个字符
3<space>：同上，往后移动3个字符
b：移动到上一个单词开头
0：移动到行首
$：移动到行尾

G：移动到文档最后一行
3G：移动到第3行
gg：移动到第一行，等价于1G
3<enter>：往下移动3行
H：high。移动到屏幕中第一行
M：middle。移动到屏幕中间一行
L：low。移动到屏幕中最后一行

Ctrl+b：backward。向上翻一页
Ctrl+f：forward。向下翻一页
Ctrl+u：up。向上翻半页
Ctrl+d：down。向下翻半页

# 查找替换
/word：向下查找word
?word：向上查找word
n：继续下一个(向下/向上)查找
N：反向继续查找，即如果是/word则向上继续查找

:3,5s/old/new/g：将第3行与第5行之间的old替换为new
:3,$s/old/new/g：$表示最后一行
:%s/old/new/g：替换所有行，'%'相当于'1,$'
```

```
:3,5s/old/new/gc：confirm。每个替换都进行确认
# 确认时有几个选择：y-yes，n-no，a-all，q-quit，l-last(替换完当前这个再退出)

# 复制、粘贴、删除(单个动作前面可以加上数字表示翻倍)
x：剪切光标处的字符
X：剪切光标前面那个字符
dd：删除当前行
d1G：删除当前行及前面的所有行
dG：删除当前行及后面的所有行
d0：删除光标处(不包含)到行首
d$：删除光标处(包含)到行尾
dw：删除光标处到单词末尾(包含空格)，配合b可以删除一个单词
yy：复制当前行
# y1G、yG、y0、y$、yw 类似 d 对应的操作
p：粘贴到光标后面或下一行(复制一行时)
P：粘贴到光标前面或上一行(复制一行时)

J：将下一行移动到当前行尾，用空格分开
u：撤销操作
Ctrl+r：重做
.：重复上一个操作
```

1.6.2. 编辑模式

```
# 进入编辑模式
i：在光标位置编辑
I：在当前行行首编辑
a：在光标后一个位置编辑
A：在当前行行尾编辑
o：在下一行插入一行
O：在上一行插入一行
r：替换光标位置字符(替换一个字符后就会退出编辑模式)
R：进入替换模式(会一直往后替换)
```

1.6.3. 底线模式

```
:w：保存
:q：退出
:q!：强制退出
:w filename：另存为文件
:3,5 w filename：将第3到第5行的内容保存到文件
:r filename：在下一行插入另一个文件的内容
:set nu：设置行号
:set nonu：取消行号
```

1.6.4. 一些技巧

```
Ctrl+v：进入可视模式
# 可视模式下移动光标可以选中多行，再按i进入编辑模式，进行编辑后按两下ESC，可以将操作应用到之前选中的所有行

Ctrl+n：补全提示

:5,12s/^/#/g：注释第5到第12行
:5,12s/#//g：取消注释
```

1.7. shell

1.7.1. 变量

```
# 全局变量
var=value # 创建变量
$var：使用变量
${var}str：当变量后面接其它字符时，需要用{}将变量名包裹
unset var # 撤销变量
unset -f func # 撤销函数

# 如果要定义局部变量，使用local关键字
local var=value

# 使用export关键字定义环境变量，环境变量可在子进程中使用
export var=value
export var # 或直接导出已有的普通变量为环境变量

# 只读变量，无法修改和撤销
readonly var=value
# 只读变量也分普通变量和环境变量
export readonly var=value
```

```
19
20 # 特殊变量
21 $0：当前脚本名称
22 $2：获得传入的第2个参数
23 ${11}：如果参数位置超过一位数，需要将下标用大括号{}包裹
24 shift 2 # 剔除前2个参数，原来的第3个参数就变成了第1个参数
25 $?：保存着上一个命令的执行状态，范围为0-255，0表示正确执行，1、2、127为系统预留的状态码
26 $#：脚本中传入参数的个数
27 $@：当不加引号时，$@和$*等价，也与"$@"等价，都是以数组的形式传递参数列表
28 $*：当加上引号时，"$*"传递的是由所有参数组成的字符串
29 ${@:2}：获取第2个参数后面的所有参数，@可以换成*
30 ${@:2:3}：获取第2个参数后面的3个参数
31 $$：脚本运行的进程ID
32 $!：后台运行的最后一个进程的ID
33
34 # 使用declare声明变量
35 declare var=value
36 declare -i var=value # 指定变量为一个整型
37 declare -x var=value # 声明环境变量
38 declare -r var=value # 声明只读变量
39 declare -xr var=value # 声明一个只读环境变量
```

1.7.2. 字符串

```
1 # 定义字符串
2 var='a$int\str' # 单引号里的内容会原样输出，因此无法使用转义字符，也无法引用变量
3 var="\"$arr\"\n" # 双引号中可以使用转义字符，也可以引用变量
4
5 # 获取字符串长度
6 ${#var}
7 # 指定位置开始截取子串
8 ${var:3}：从第3个字符开始截取到末尾
9 ${var:0-3}：从倒数第3个字符开始截取到末尾
10 ${var: -3}：同上，注意-3前面有个空格
11 # 指定位置开始截取指定长度子串
12 ${var:3:5}：从第3个字符开始截取5个字符
13 ${var:3:-2}：从第3个字符开始截取完之后，删除末尾的2个字符
14 # 删除指定字符及前面的所有字符
```

```
${var#*A}：删除从左往右第一个字符A及之前的所有字符
${var##*A}：删除从左往右最后一个字符A及之前的所有字符
# 删除指定字符及后面的所有字符
${var%A*}：删除从右往左第一个字符A及之后的所有字符
${var%%A*}：删除从右往左最后一个字符A及之后的所有字符
# 替换子串
${var/old/new}：将首次出现的old替换为new
${var//old/new}：将所有的old替换为new
${var/#old/new}：将行首的old替换为new
${var/%old/new}：将行尾的old替换为new
# 删除子串，和替换子串的语法类似，只是将new省略了(即替换为空串)
${var/str}：删除首次出现的子串str
# 大小写转换
${var^^}：将所有字母转换为大写
${var,,}：将所有字母转换为小写
# 字符串为空时的操作
${var:=value}：当var为空时返回value，并将var赋值为value
${var:-value}：当var为空时返回value，但不进行赋值操作
${var:+value}：当var不为空时返回value，也不进行赋值操作
${var:?value}：输出bash错误，其中value为错误提示信息
```

1.7.3. 数组

```
var=(value1 value2 ...) # 定义数组
var[10]=ten # 可以单独为某个元素赋值，下标没有长度限制
${var[2]}：根据下标获取元素
${var[@]}：获取所有元素，只写var默认获取的是第1个元素
${var[*]}：同上
${#var[@]}：获取数组长度
${#var[*]}：同上

# 使用-A选项定义关联数组，即支持字符串作为下标，相当于是一个字典
declare -A var=([key1]=value1 [key2]=value2)
${!var[@]}：获取所有的key
${!var[*]}：同上
```

1.7.4. 算术运算

```
# 使用let命令，支持++、--、+=、-=
let var=1+2; $var=3
# let只支持整数运算
let var=5/2 # $var=2，只保留结果的整数部分
```

```
let var=3.6*2 # 语法错误，不支持浮点数

# 使用expr命令，只支持整数运算。注意数字和运算符之间需要用空格隔开，且乘号*需要进行转义
expr 1 + 2 # 输出3
# 如果需要将运算结果赋值给变量，使用命令引用
var=`expr 5 \* 2` # $var=2

# 使用bc命令，支持浮点数运算
echo '2.3+4' | bc # 输出6.3
# 除法运算时需要指定精度，否则只会保留整数部分
echo 'scale=3;8/3' | bc # 输出2.666
# 不使用管道，直接使用bc命令
bc <<< '2.3+4'

# 使用算术运算语法，只支持整数运算，[]或(())里面引用变量不用加$
var=$[3+4] # $var=7
var=$((8/3)) # $var=2

# 直接声明变量为整型变量
declare -i var=1+3 # $var=4
```

1.7.5. 逻辑关系运算符

```
# 非：!
# 与：-a或&&，&&有短路功能
# 或：-o或||，||有短路功能
# 除了是否有短路功能外，两者在语法上也有差别
[ $var1 -a $var2 ]
[ $var1 ] && [ $var2 ]
[[ $var1 && $var2 ]]
[[ $var1 ]] && [[ $var2 ]]
# &&和||也可以用来控制多条命令的执行
cmd1 && cmd2 # 当cmd1执行成功后执行cmd2
cmd1 || cmd2 # 当cmd1执行失败后执行cmd2

# 大于>和小于<，支持整数和字符串的比较。=和==都可以用来判等
[ 2 \> 1 ] # 当使用单括号时需要进行转义
[[ 'a' < 'b' ]]
# 当比较数字大小时也可以使用-gt和-lt，不支持字符串比较
[ 1 -lt 2 ]
```

```
[[ 2 -gt 1 ]]
# 一共6个整数关系运算符：-gt -ge -lt -le -eq -ne

# 匹配正则表达式
[[ $var =~ /exp/]]
```

1.7.6. 文件测试运算符

```
# file是文件路径
[ -d $file ]：判断是否目录
[ -f $file ]：判断是否文件
[ -r $file ]：判断文件是否可读
[ -w $file ]：判断文件是否可写
[ -x $file ]：判断文件是否可执行
[ -e $file ]：判断文件是否存在
```

1.7.7. 控制流程

```
# 条件控制语句
if condition
then
    command
elif condition
then
    command
else
    command
fi
# 在终端写成一行
if condition; then command; elif condition; then command;
else command; fi

# for循环语句
for var in value1 value2
do
    command
done
# 在终端写成一行
for var in value1 value2; do command; done

# while循环语句
while condition
```

```
do
    command
done

# until循环
until condition
do
    command
done

# 无限循环
for (( ; ; ))
while true
while :
# 可以使用break和continue

# case语句
case var in
1|2|3)
    command
    ;; # break
*) # *表示默认选项
    command
    ;;
esac
```

1.7.8. 函数

```
function func() { # function关键字和()都可以省略
    command; # 每行末尾需要加上分号;
    return x; # 返回0-255中的状态码，如果没有return语句则返回最后一条命令的状态码
}
func # 直接调用函数
func 1 2 3: # 带参数调用，函数内部可以使用$1之类的变量获取参数
```

1.7.9. 重定向

```
cmd > file # 输出重定向，覆盖原文件
cmd >> file # 输出重定向，往原文件末尾追加
cmd < file # 输入重定向，将file的内容作为cmd的输入
# shell默认打开的3个文件：0-stdin; 1-stdout; 2-stderr
cmd 2>file # 将stderr重定向
cmd > file 2>&1 # 将输出和错误都重定向
cmd < file1 > file2 # 将输入和输出都重定向
# /dev/null是一个特殊的文件，写入其中的内容都会被丢弃
cmd > /dev/null 2>&1 # 屏蔽输出和错误
```

1.7.10. 通配符

```
?：匹配任意单个字符
*：任意数量的字符，包括空
[abc]：匹配[]中的任意字符
[a-z]：匹配范围内的任意字符
[a-9]：如果不是合法范围，则只会当成"[a-9]"这个字符串本身
[^abc]：^表示取反，即匹配[]中以外的任意字符
[!0-9]：!同^表示取反
{abc,123}：匹配abc或者123之一
ab{,c}：匹配ab或abc
{8..11}：8 9 10 11
{1..10..2}：步长为2输出 1 3 5 7 9
{008..11}：前面补0对齐 008 009 010 011
{e..a}：倒序输出 e d c b a
{a1..3c}：无法扩展，输出{a1..3c}本身

# 注意：*和?不能匹配路径分隔符/
ls */*.jpg
```

1.7.11. 注意事项

```
# 反引号用于命令替换，等价于$()
echo `ll` # 将ll执行的结果打印出来。但结果不会换行
echo $(ll) # 推荐使用$()
echo "$(ll)" # 和ll直接执行的结果一样会换行

# shell脚本的运行
./run.sh # 开启子进程执行脚本，无法使用当前shell的普通变量
source ./run.sh # 不会产生子进程在当前shell中执行脚本，就可以使用当前shell的普通变量了
```

```
. ./run.sh # 第一个'.'的作用和source一样
# 在脚本文件中使用source或"."相当于是导入了另一个文件，可以使用其中的变量

# 将多个命令组成成一个整体
(cmd1;cmd2)
{ cmd1;cmd2; } # 最后一个命令后面也要有';'，且'{}'与命令之间需要用空格隔开
# ()中的命令会在子shell中执行，而{}会在当前shell中执行

# []与[[]]的区别，总的来说使用[[]]保险一些
# 1. 判断字符串是否为空
[ $var ]：判断变量是否为空
[[ $var ]]：也可以直接使用[[]]
[ -z "$var" ]：-z选项判断变量是否为空，这时变量需要加上引号，等价于test -n "$var"
[[ -n $var ]]：-n选项判断变量是否非空，使用[[]]时则不需要加引号
# 2. 逻辑运算符(前面的例子)
[ $var1 -a $var2 ]：-a和-o只能用在[]里
[ $var1 ] && [ $var2 ]：&&和||使用[]时必须放在[]外面
[[ $var1 && $var2 ]]：&&和||使用[[]]时既可放在里面也可放在外面
[[ $var1 ]] && [[ $var2 ]]
# 3. 关系运算符
[[ $var =~ /exp/]]：正则表达式=~只能用[[]]
[ 2 \> 1 ]：大小关系符><使用[]时需要进行转义
[[ 'a' < 'b' ]]

# [[]]和(())的区别(没完全理清，只记录一些场景)
# [[]]是增强[]，主要用于-z等条件测试、<、&&等运算符
# (())是增强()，常用于算术运算，变量可以不使用$
if (($i<5))
for ((i=0;i<5;i++))
# [[]]与表达式直接要用空格分开，(())可以不用
# $((exp))和`expr exp`等价
```

1.7.12. 内置命令read

```
read var # 将读取的内容赋值给var
-p：prompt。提示信息
-t：timeout。设置超时时间，单位秒
-n：nchars。最大字符数量，当输入到指定数量的字符时直接结束输入
-s：silent。不显示输入，比如输入密码时
-e：使用命令补全功能
-a：array。以空格为分隔符输入数组
-r：raw。不将\作为转义字符，直接读取字符串本身的内容
```

1.7.13. 内置命令declare

```
-：设置属性
+：取消设置的属性

i：整数
a：数组
A：关联数组，即key可以是字符串
r：只读变量
x：环境变量

declare -i var=1 # 定义整数
var=string # $var=0，因为var已经被定义为整数了
declare +i var # 取消var的整数属性
var=string # $var=string
```

1.8. awk

```
# 基本语法：awk [option] 'pattern {action}' files
# 默认的action是打印整行，即print $0

# 一个复杂一点的例子
awk -v name='hb' -F; 'BEGIN{var1=val1;var2=val2}{print
$1,$2}END{printf "%s\n","end"}' files

# 打印奇偶行简洁写法
awk 'i=!i' files # 奇数行
awk '!(i=!i)' fiels # 偶数行
```

1.8.1. 变量

```
# 内置变量
$0：表示整行
$x：x是一个整数，表示第几列
NF：表示列数
$NF：表示最后一列

NR：表示当前行号，多个文件时行号会延续
FNR：多个文件时各个文件分别记录行号
FS：输入分隔符
OFS：输出分隔符
RS：输入换行符
ORS：输出换行符
FILENAME：当前文件名
ARGC：命令行参数的个数
ARGV：命令行参数数组，第一个参数就是awk本身，后面的参数是各个操作的文件名

# 自定义变量有两种方式，可以通过-v选项定义，也可以在action中直接定义和使用
awk 'BEGIN{var=4;print var}'
# 数组也可以直接使用，awk中的数组其实是一个字典，它的键可以是字符串
awk 'BEGIN{arr[3]=8;print arr[3]}'
# 对于没有定义的变量或数组元素，默认值是空字符串，即""
key in arr：判断数组中是否存在某个元素
delete arr[key]：删除数组中的元素
delete arr：删除整个数组
# 注意用for遍历数组时，遍历的是key而不是value，且是无序的

# 对字符串变量进行加法运算时，变量会被当做0
awk 'BEGIN{var="test";print var+1}'
```

1.8.2. option

```
# 分隔符分输入分隔符和输出分隔符，默认都是空格
-F;：指定分号';'为输入分隔符
-v FS=';'：通过内置变量指定输入分隔符
-v OFS='#'：指定输出分隔符为'#'
-v VAR=val：自定义变量
```

1.8.3. pattern

```
# 当pattern满足条件时，后面的action才会执行
BEGIN：指定在处理文本之前执行的操作
END：指定在处理文本之后执行的操作
空模式：任何一行都满足条件

# 关系表达式，如== != < <= > >=
NR>=5 && NR<=10 # 输出第5行到第10行

# 正则表达式
~：表示匹配正则表达式时为真
!~：表示不匹配正则表达式时为真
# 直接使用正则表达式而不说明哪个变量需要满足条件时，默认为整行，即$0~/exp/
# 当使用匹配次数的正则表达式时，需要加上--posix或--re-interval选项
awk --posix '/a{2,4}bc/{print}' files

# 匹配两个正则表达式之间的行
'/exp1/,/exp2/' # 匹配exp1第一次出现的行与exp2第一次出现的行之间的所有行
```

1.8.4. action

```
print：打印并换行。打印多个数据时如果用逗号','连接，则输出数据会以输出分隔符分隔，否则多个数据会紧连
printf：用法和printf命令基本一样，只不过格式与字符串之间要用逗号','隔开

# 条件控制语句，和c++的语法一样
if () {} else if () {} else {}
condition?action1:action2 # 三目运算符
# 循环语句
for (i=0;i<10;++i) {}
for (key in arr) {}
while () {}
do {} while ()
# 循环语句中同样有break和continue语句

next：结束当前行，继续下一行，类似于循环语句中的continue
```

```
exit：直接跳到END模式执行的动作，如果没有END模式则直接结束awk
```

1.8.5. 内置函数

```
# 算术函数
srand()：生成随机数种子
rand()：生成0-1之间的随机数
int()：转化为整数

# 字符串函数
length($0)：返回字符串长度，不指定参数时默认为$0
index($0,'str')：查找子串出现的位置，不存在时返回0
gsub('old','new',$0)：字符串替换，第3个参数不指定时默认为$0。支持正则表达式
sub('old','new',$0)：只替换第一个匹配项
split(str,arr,sep)：将str按sep切割，存到arr数组中，下标从1开始。返回的是数组的长度

# 其它函数
asort(arr)：将数组元素升序排列，但排序后的数组下标会从1开始重置。返回数组的长度
asort(arr,res)：将排序后的结果存到res，不改变原来的数组
asorti(arr,res)：将数组的key排序后存到res中。返回数组的长度
```

2. 通用知识

2.1. 正则表达式

2.1.1. 普通字符

```
[abc]：匹配abc里的任意字符
[^abc]：匹配非abc里的任意字符
[a-z]：匹配ASCII码在a和z之间的任意字符
[!-a]：匹配ASCII码小于等于a的任意字符
[a-~]：匹配ASCII码大于等于a的任意字符
.：匹配除换行符之外的任意字符，等价于[^\r\n]
ab|12：匹配ab或12
\d：匹配数字，等价于[0-9]
\D：匹配非数字
\w：匹配数字、字母、下划线，等价于[0-9A-Za-z_]
\W：匹配非数字、字母、下划线
```

2.1.2. 非打印字符

```
\r：匹配回车符
\n：匹配换行符
\t：匹配制表符
\v：匹配垂直制表符
\f：匹配换页符
\s：匹配所有空白字符，等价于[ \r\n\t\v\f]
\S：匹配所有非空白字符，等价于[^ \r\n\t\v\f]
```

2.1.3. 定界符

```
^：锚定行首，即匹配以exp开头的行
$：锚定行尾
\<：锚定词首，即匹配以exp开头的单词
\>：锚定词尾
\b：既可锚定词首又可锚定词尾，即可替换\<或\>
\B：匹配非单词边界，即匹配不以exp开头但包含exp的单词
```

2.1.4. 限定符

```
*：匹配任意次数
+：匹配1次或多次
# *和+默认匹配是贪婪的，即它会尽可能多地去匹配，在其后面加上?可以实现非贪婪匹配
# 比如对于字符串<h1> title </h1>，正则表达式/<.*>/会匹配整个字符串，而/<.*?>/则只会匹配到<h1>

?：匹配0次或1次
{n}：匹配n次
{n,}：匹配至少n次
{n,m}：匹配至少n次、至多m次
```

2.1.5. 分组与引用

```
(exp)：匹配exp并缓存，可用\x(x是一个数字)代替第x个分组匹配的结果
# /\b([a-z]+) \1\b/可以用来匹配两个相同相邻的单词，\1表示前面匹配到的那个单词

(?:exp)：匹配exp但不缓存
exp2(?=exp1)：查找exp1前面的exp2，同样exp1不会被缓存
(?<=exp1)exp2：查找exp1后面的exp2
exp2(?!exp1)：查找后面不是exp1的exp2
(?<!exp1)exp2：查找前面不是exp1的exp2
```

2.1.6. 修饰符

```
/abc/i：不区分大小写
/abc/g：查找所有的匹配项
/^abc/m：多行匹配，使得^和$匹配每一行的开头和结尾，而不是匹配整个字符串的
/ab./s：使得.可以匹配换行符，默认情况下.是匹配换行符以外的任意字符
```

2.1.7. 内置字符簇

```
[[:digit:]]：数字
[[:alpha:]]：字母
[[:alnum:]]：数字和字母
[[:upper:]]：大写字母
[[:lower:]]：小写字母
[[:space:]]：空白字符
[[:punct:]]：任何标点符号
[[:xdigit:]]：16进制数字，等价于[0-9A-Fa-f]
```