第五章：文件读写

生命科学学院

一、打开、关闭文件

- 语法为open (filevar, filename)，其中filevar 为文件句柄，或者说是程序中用来代表某文件的代号，filename为文件名，其路径可为相对路径， 亦可为绝对路径
  open(FILE1,"file1");
  open(FILE1, "/u/jqpublic/file1");

  打开文件时必须决定访问模式， 在PERL 中有三种访问模式：读、写和添加。后两种模式的区别在于写模式将原文件覆盖，原有内容丢失， 形式为：
  open(outfile,">outfile");

- 而添加模式则在原文件的末尾处继续添加内容， 形式为：open(appendfile,">>appendfile")。
- 要注意的是，不能对文件同时进行读和写/添加操作。

一、打开、关闭文件

- open 的返回值用来确定打开文件的操作是否成功，当其成功时返回非零值， 失败时返回零， 因此可以如下判断：

```
if (open(MYFILE, "myfile")) {
#heres what to do if the fileopened successfully
}
```

- 当文件打开失败时结束程序：

```
unless (open (MYFILE, "file1")) {
  die("can not open input file file1\n");
}
```

- 亦可用逻辑或操作符表示如下：

```
open (MYFILE, "file1") || die ("Could not open file");
```

当文件操作完毕后， 用close(MYFILE); 关闭文件。

二、读文件

- 语句$line = <MYFILE>;从文件中读取一行数据存储到简单变量$line 中并把文件指针向后移动一行。<STDIN>为标准输入文件， 通常为键盘输入， 不需要打开。语句@array = <MYFILE>;把文件的全部内容读入数组@array，文件的每一行(含回车符）为@array的一个元素。

三、写文件

- 形式为：
  open(OUTFILE, ">outfile");
  print OUTFILE ("Here is an output line.\n") ;
  注： STDOUT、STDERR 为标准输出和标准错误文件，通常为屏幕， 且不需要打开。

四、判断文件状态

- 1、文件测试操作符
  语法为： -op expr， 如:

```
if ( - e "/path/file1" ) {
  print STDERR ("File file1 exists.\n" );
}
```

四、判断文件状态

操作符	描述
-b	是否为块设备
-c	是否为字符设备
-d	是否为目录
-e	是否存在
-f	是否为普通文件
-g	是否设置了setgid位
-k	是否设置了sticky位
-l	是否为符号链接
-o	是否拥有该文件
-p	是否为管道
-r	是否可读
--s	是否非空
-t	是否表示终端

四、判断文件状态

-u	是否设置了setuid位
-w	是否可写
-x	是否可执行
-z	是否为空文件
-A	距上次访问多长时间
-B	是否为二进制文件
-C	距上次访问文件的inode多长时间
-M	距上次修改多长时间
-O	是否只为“真正的用户”所拥有
-R	是否只有“真正的用户”可读
-S	是否为socket
-T	是否为文本文件
-W	是否只有“真正的用户”可写
-X	是否只有“真正的用户”可执行

注：“真正的用户”指登录时指定的userid，与当前进程用户ID 相对， 命令suid可以改变有效用户ID。

四、判断文件状态

- 例:

```
unless (open(INFILE, "infile")) {
die ( "Input file infile cannot be opened.\n");
}
if( -e "outfile") {
die("Output file out file already exists .\n" ) ;
}
unless (open(OUTFILE, ">outfile")) {
die ("Output file outfile cannot be opened.\n");
}
```

等价于

```
open(INFILE, "infile") && !( -e "outfi l e " ) &&
open(OUTFILE, ">outfile") || die("Cannot ope
files\n");
```

五、命令行参数

- 象C 一样，PERL 也有存储命令行参数的数组@ARGV，可以用来分别处理各个命令行参数； 与C 不同的是，$ARGV[0]是第一个参数， 而不是程序名本身。

```
$var = $ARGV[0]; # 第一个参数
$numargs = @ARGV; # 参数的个数
```

- PERL 中，<>操作符实际上是对数组@ARGV 的隐含的引用，其工作原理为：
  1、当PERL 解释器第一次看到<>时，打开以$ARGV[0]为文件名的文件；

五、命令行参数

- 2、执行动作s h i ft(@ARGV)；即把数组@ARGV 的元素向前移动一个， 其元素数量即减少了一个。
- 3 、<>操作符读取在第一步打开的文件中的所有行。
- 4、读完后， 解释器回到第一步重复。

例:

```
@ARGV = ("myfile1", "myfile2"); #实际上由命令行参数赋值
while($line = <> ) {
    print ($line) ;
}
```

将把文件myfile1 和myfile2 的内容打印出来。

六、打开管道

- 用程序的形式也可以象命令行一样打开和使用管道(ex : ls >tempfile)。如语句
  open(MYPIPE ,"|cat>hello"); 打开一个管道，发送到MYPIPE 的输出成为命令"cat >hello"的输入。由于cat 命令将显示输入文件的内容， 故该语句等价于
  open(MYPIPE,">hello"); 用管道发送邮件如下:

```
open (MESSAGE, "|maildave" ) ;
print MESSAGE ("Hi, Dave! Your Perl program
sent this!\n " ) ;
close (MESSAGE);
```