



The Kalmanovitz Library and
The Center for Knowledge Management

PERL: File Handles and Control Structures

Gilberto da Gente

Bioinformatics Resource Specialist

Wednesday, October 8th 2008

University of California, San Francisco



UCSF

Filehandles

- Files are accessed within a Perl program through *filehandles*
- They are linked to filenames within a file system through an *open* statement.
- By convention, Perl filehandle names are written in all uppercase.

```
#open statement:  
open (FILEHANDLE, "filename");
```

```
#Example:  
open (INPUT, "index.html");
```

Update, Create, and Write Files

- Files may be opened for write access, update, and creation.
 - ✓ write replaces the file contents,
 - ✓ update appends new data to the end of the current contents.
- These options are indicated by appending either a single or a double greater than (>) symbol to the file name as a prefix:

Form:

```
open (FILEHANDLE, ">filename"); # write access  
open (FILEHANDLE, ">>filename"); # update
```

Examples:

```
open (INPUT, ">index.html");  
open (INPUT, ">>index.html");
```

logical or and die operators

- Perl will continue operating regardless of whether the open was successful or not, so
 - ✓ you need to test the open statement.
- Like other Perl constructs, the open statement **returns a true or false value**, indicating success or failure.
- logical or and die “ || **die**” operators test for this return value.

Form:

```
open (FILEHANDLE, "filename") || die "Message  
written to STDERR";
```

Example:

```
open (INPUT, "index.html") || die "Error  
opening file index.html ";
```

Read files

The file, once opened and associated with a filehandle, can be read with the angle brackets operator (<>), which can be used in a variety of constructs.

Form:

<FILEHANDLE>

Example:

```
while (<INPUT>) { # read one line at a time until EOF
  chop; # remove newline
  print "line = $_ \n"; } # print line read using default
scalar variable
```

Write files

Once a file has been opened for either write or update access, data can be sent to that file through the print operator.

Form:

```
print FILEHANDLE content;
```

Example:

```
print OUTPUT "$next \n";  
# outputs contents of $next followed by  
newline char.
```

Close files

Files are closed implicitly when another open is encountered with the same filehandle.

Normally they are closed explicitly with the close statement.

Form:

```
close (FILEHANDLE);
```

Example:

```
close (INPUT);
```

Statement Blocks

- Statement blocks provide a mechanism for grouping statements that are to be executed as a result some **expression being evaluated**.
- They are used in all of the control structures.
- Statement blocks are designated by pairs of **curly braces**.

Form:
BLOCK

Example:

```
{stmt_1;  
stmt_2;  
stmt_3; }
```

if statement

if statement

Form:
if (EXPR) BLOCK

Example:
if (expression) {
 true_stmt_1;
 true_stmt_2;
 true_stmt_3;
}

if/else statement

Form:
if (EXPR) BLOCK
else BLOCK

Example:
if (expression) {
 true_stmt_1;
 true_stmt_2;
 true_stmt_3; }
else {
 false_stmt_1;
 false_stmt_2;
 false_stmt_3; }

if/elseif/else statement

Example:

```
if (expression_A) {  
    A_true_stmt_1;  
    A_true_stmt_2;  
    A_true_stmt_3; }  
elseif (expression_B) {  
    B_true_stmt_1;  
    B_true_stmt_2;  
    B_true_stmt_3; }  
else {  
    false_stmt_1;  
    false_stmt_2;  
    false_stmt_3; }
```

Form:

```
if (EXPR) BLOCK  
elseif (EXPR) BLOCK . . .  
else BLOCK
```

if statement examples

```
$number = 95;  
if($number == 95) {  
    print 'Number is ninety-five!';  
} # this prints "Number is ninety-five!";
```

```
$name = 'David';  
$birth_month = 'May';  
if($name eq 'David' && $birth_month eq  
'May'){  
    print 'David was born in may!';}  
else {  
    print 'go away';}# prints "go away"
```

Labels and Control Structures

Form:

LABEL: while (EXPR) BLOCK

- The LABEL in this and the following control structures is optional.
- In addition to description, it also provides function in the quasi-goto statements: last, next, and redo.
- Perl conventional practice calls for labels to be expressed in uppercase to avoid confusion with variables or key words.

while statement

Form:

```
LABEL: while (EXPR) BLOCK
```

While loops continually reiterate as long as the conditional statement remains true.

Example:

```
ALABEL: while (expression) {  
    stmt_1;  
    stmt_2;  
    stmt_3; }
```

while statement examples

```
open (NCBIFILE, "ncbi.txt") || die "couldn't open the
file!";
while ($record = < NCBIFILE >) {print $record; }
close(NCBIFILE);
```

```
print "how old are you? ";
$a = <STDIN>;
chomp($a);
while ($a > 0) {
print "At one time, you were $a years old.\n";
$a--; }
```

```
while(1){print "whats going on?";}
```

until statement

Form:

```
LABEL: until (EXPR) BLOCK
```

Example:

```
ALABEL: until (expression) { # while no  
    stmt_1;  
    stmt_2;  
    stmt_3; }
```

```
10 9 8 7 6 5 4 3 2 1  
Blastoff.
```

```
$count = 10;  
until ($count == 0) {  
    print "$count ";  
    $count--;  
}  
print "\nBlastoff.\n";
```

for statement

Form:

```
LABEL: for (EXPR; EXPR; EXPR) {BLOCK}
```

Example:

```
ALABEL: for (initial exp; test exp;
increment exp) e.g. ($i=1; $i<5; $i++) {
    stmt_1;
    stmt_2;
    stmt_3; }
```

```
for ($count=1; $count<11;
$count++) { print "$count\n"; }
```

1
2
3
4
5
6
7
8
9
10

foreach statement

Form:

```
LABEL: foreach VAR (EXPR)
{BLOCK}
```

Example:

```
ALABEL: foreach $i (@aList) {
    stmt_1;
    stmt_2;
    stmt_3; }
```

```
foreach $geneid (@genes) {
    print "$geneid\n";
}
```

Loop Control Operators

- Causes execution to jump from where they occur to some other position, defined with respect to the block structure of the encompassing control structure.
- They function as limited forms of goto statements.

last operator

Last causes control to jump from where it occurs to the first statement following the enclosing block.

Example:

```
ALABEL: while (expression) {  
    stmt_1;  
    stmt_2;  
    last;  
    stmt_3; }  
# last jumps to here
```

last operator

Example:

```
ALABEL: while (expression) {  
    stmt_1;  
    stmt_2;  
    BLABEL: while (expression) {  
        stmt_a;  
        stmt_b;  
        last ALABEL;  
        stmt_c; }  
    stmt_3; }  
# last jumps to here
```

If last occurs within nested control structures, the jump can be made to the end of an outer loop by adding a label to that loop and specifying the label in the last statement.

next operator

The next operator is similar to last except that execution jumps to the end of the block, but remains inside the block, rather than exiting the block. Thus, iteration continues normally.

Example:

```
ALABEL: while (expression) {  
    stmt_1;  
    stmt_2;  
    next;  
    stmt_3;  
    # next jumps to here }
```

As with last, next can be used with a label to jump to an outer designated loop.

redo operator

The redo operator is similar to next except that execution jumps to the top of the block without re-evaluating the control expression.

Example:

```
ALABEL: while (expression) {  
    # redo jumps to here  
    stmt_1;  
    stmt_2;  
    redo;  
    stmt_3; }
```

As with last, redo can be used with a label to jump to an outer designated loop.