

Introduction to Perl

Perl means Practical Extraction and Report Language.

Perl is used for writing scripts to process text documents. Some users use Perl for writing CGI (Common Gateway Interface) scripts. It is considered a combination of sed and awk plus a whole lot more.

Perl may be used in both the UNIX and the Microsoft environment.

It is free. You may download your own copy from www.perl.com. Make sure you get the version for the operating system which you are running.

You will find Perl practical and easy to use.

Some features of the language:

- Implicit variables defined by language implementation. An example is `$_`.
- Interchangeability of Functions and operators. In Perl, functions are sometimes called without the parenthesis around the argument list.
- Perl has only one number data type which is the double data type.
- Variables are implicitly declared. The type of data

will be inferred by the compiler.

- **Strings versus numbers.** Some Perl operators impose a type on the data, which will cause implicit data conversion.
- **Scalar and list context.** Sometimes an array will appear as an operand where a scalar is required. The array variable will be interpreted as the length of the array.
- **There is always more than one way to do any task.**

Don't worry, you will see why all of the above are considered a features before you are finished with this course.

An Example of a script in UNIX and an example of the same script in Perl:

The Problem:

List all users by name and username that are found in the `/etc/passwd` file that have a group number of 2004. The output will be the name followed by 3 hyphens followed by the username.

An example of this problem solution without using Perl is included in this [UNIX script](#).

The following example provides a somewhat more elaborate output than the UNIX script with very little more work.

You are not expected to understand all of the statements in the following example. It is placed here to give you an idea of what a [Perl script](#) looks like.

© B.Huseby, 2003-04

The Linux server

Each of you will be given an account on a Linux server. On that server you will be able to do all homework required for this course. This Linux server is located at Portland Community College. It is available on-line 24 hours a day. You may use [telnet](#) to access this server from either UNIX or a PC. [FTP](#) is a tool to transfer files. It is found in both the PC and UNIX environment. Another tool to use on your PC would be Secure Shell which can be obtained free for educational use from www.ssh.com. This tool makes file transfer relatively simple.

The server you will login to is tesla.syi.pcc.edu.

Your username on this system will be your first initial followed by your last name up to 8 characters. For example if your name is Tom Jones, your login username will be [tjones](#). Your password will be mailed to you before your first class. If for some reason there is a conflict in account names, you will be notified of your user name.

File transfer using ftp

At the command prompt type:

```
ftp tesla.syi.pcc.edu
```

You will be prompted to login to your user account.

At the > prompt you may type a ? to get general help. A ? followed by a command name gives help for that command. For Example:

```
> ? get
```

Some commands to remember are the following:

binary ---- transfers files properly.

get file ---- gets a file from the server for you and places it in your current folder (directory).

put file --- takes a file from your current folder and writes it to the current folder on the server.

quit ---- stops the ftp session.

How to telnet

From the UNIX or DOS command prompt type:

[telnet tesla.pcc.edu](#)

[login: tjones](#)

[password:](#)

Basic Perl Rules

Perl like all other programming languages has some basic rules you must follow in order for the source file you write to be recognized by the Perl interpreter.

Subtopics:

[White space](#) [Semi Colons](#) [Comments](#) [Blocks](#) [Assignment Statements](#)
[Elementary Input](#) [Elementary Output](#)

Web Links: [Basic Syntax](#), [Perl Style Guide](#)

[Go to Top](#)

White space is used for readability.

White space is defined to be any space, tab or new line character. For example a program may be written as the following:

```
perl -e "$cnt=1;while (<>){cnt++;print $cnt,$_;$}print ' total lines
=',$cnt- 1;" p10
```

The action of this script is to list each line of p10 with a line number. After the last line of the file has been displayed, the total number of lines is displayed.

A better way to represent this script would be the following:

```
perl -e "$cnt = 0;
    while ( <> )
    {
        $cnt++;
        print $cnt, $_;
    }
    print ' total lines =', $cnt;
" p10
```

If the script were in a file called **x.pl**, the script could contain comments:

```

$cnt = 0;                #initialize the line counter
while ( <> )           #input a line of the file
{
    $cnt++;           #count the line
    print $cnt, $_;     #display the count and the line
}
print ' total lines =', $cnt; #display total number of lines

```

To execute this script from the command prompt type:

```
perl x.pl p10
```

Note: The extension .pl is not required. You may name your Perl scripts with any name you like. If you were to run a Perl script in the Microsoft environment, the .pl will allow you to have the script automatically start.

[Go to Top](#)

Semi colons are used as a statement terminators.

In the example above the semi-colon is found at the end of each statement. Note: the loop control does not have a semi colon. A misplaced semicolon at the end of the loop control line would generate a syntax error. See the example below.

```

$cnt = 0;                #initialize the line counter
while ( <> )           #input a line of the file
{
    $cnt++;           #count the line
    print $cnt, $_;     #display the count and the line
}
print ' total lines =', $cnt; #display total number of lines

```

[Go to Top](#)

Comments

Comments are represented by the pound symbol (#). All characters from that symbol to the end of the line are not recognized by the Perl interpreter. Comments should describe the action in the Perl script.

```
# the comment
```

[Go to Top](#)

Blocks

Blocks look like familiar blocks in the C and C++ languages. Use { } like in C and C++ to enclose multiple statements. Use the curly braces where required to complete alternation and repetition control statements.

```
$cnt = 0;           #initialize the line counter
while ( <> )       #input a line of the file
{
    $cnt++;        #count the line
    print $cnt, $_; #display the count and the line
}
print ' total lines =', $cnt; #display total number of lines
```

[Go to Top](#)

Assignment statements

Assignment statements are a way to assign a value to a variable. When writing a Perl script, it is sometimes necessary to save an answer for use later. See \$cnt in the above example. In that example, \$cnt is assigned a value of 0 outside the loop. Within the loop, using the increment operator, one is added to \$cnt each time the loop is executed.

```
$cnt = 0;           #initialize the line counter
while ( <> )       #input a line of the file
{
    $cnt++;        #count the line
```

```

    print $cnt, $_;    #display the count and the line
}
print ' total lines =', $cnt; #display total number of lines

```

Another example of an assignment statement, this time calculating the area of a circle:

```
$area=2*3.14159*$radius*$radius;
```

[Go to Top](#)

Elementary input

Input of data into Perl program is done through the angle bracket operator, `<>`. Each time this statement is executed, one line of input is read from STDIN (Standard Input).

A program named `testio.pl` might consist of the following:

```

while(<>)
    {print;}

```

In this example, three lines of input from Standard Input are read and displayed. Each line is placed into a default variable `$_`. Since the print statement does not have a list, the list is assumed to be `$_`. This script may be run the the following ways:

```
perl testio.pl p10
```

or

```
perl testio.pl
```

The first execution reads all lines from the file `p10`, displaying them to the screen. The second execution will cause you to use the keyboard to input each line for the data, after which the computer will display what you typed. In UNIX, to terminate input from the keyboard you must type **CTRL D**.

The above script may be rewritten to not use the implicit variable `$_`. In that case your new script would look like the following:

```
while($line=<>)
  {print $line;}
```

Now the variable \$line is the location where the input line is placed, not in the implicit variable location \$_.

[Go to Top](#)

Elementary Output:

The print statement is the simplest form of output in a Perl script. A print without arguments will print the contents of the implicit variable \$_. Print will accept a list of items to print. The items which you print may be the contents of variables or strings within quotes. A comma separating the items in the list will cause a space to appear in the output. Space separated items in a list will not have a space between the items on the display.

For example:

```
print "Hello, world";      #one item in the list
print "Hello","world";    #two items in the list
```

The output would be:

```
Hello, worldHelloworld
```

The output from these two print statements was put on the same line. That is because a new line was not specified. Following the 'd' character in world put a `\n`. The second Hello world lacks the comma and also a space. The comma in this case separated the items in the list, so it was not printed. The space will not be printed unless it is part of one of the strings.

```
print "Hello, world\n";    #one item in the list
print "Hello ", "world\n"; #two items in the list
```

Now the output looks like the following:

```
Hello, world
Hello world
```

© B.Huseby, 2003-04

Starting Perl

There are a variety of ways to start Perl from UNIX. Each method has it's own merits and use.

Subtopics:

[The General Form](#) [Script on the Command Line](#) [Putting a Script inside a File](#) [Starting Perl from within a Script](#)
[Command Line Options](#)

Web Link: [Running Perl Programs](#)

[Go to Top](#)

The General form:

```
perl scriptfile inputfile
```

In this case the command perl is typed, followed by the instruction file name (scriptfile) followed by the file name of the file that is going to be processed (inputfile).

[Go to Top](#)

Using a -e to put a script on the command line

Sometimes a script is very short. If that is the case, the actual script file contents (enclosed in single quotes) can be placed on the command line. To tell perl that the script is

on the command line, use the `-e` option.

```
perl -e 'the script' inputfile
```

Examples:

```
perl -w -e 'while(<>){ chomp; print;}' pres1
```

The above script will input all the lines from the `pres1` file, removing all linefeed characters (`chomp` does this) before displaying the output lines.

```
perl -e '$_="cabcccdef";
chop; print; print "\n"; s/c{3,5}?/X/; print ; print
"\n"; '
```

The above script assigns the default input variable to a string. `Chop` removes the last character from the string. The string is output followed by a newline character. The next action is to substitute in `$_` for occurrences of 3 to 5 consecutive `c`'s with a single letter `X`. The string is output again followed by a newline character.

[Go to Top](#)

Putting a script inside a separate file

Contents of file `p1`

```
$_="cabcccdef";
chop;
print;
print "\n";
s/c{3,5}?/X/;
```

```
print ;
print "\n";
```

perl p1

The above line starts Perl with the script called p1. No input file is given in this example. This example has the same action as the previous script example.

Contents of file p2

```
while (<>)
{
s/, \|\|\/g; #remove all comma spaces at end of
field
print;
}
```

perl p2 pres1

The above line starts Perl with the script called p2 with the data file pres1. The output from this script will be all lines from the data file with any combination of , | (comma space pipe) being replaced by just a | (pipe) symbol. The backslash used before the pipe symbol is keeping perl from using it as a special character.

[Go to Top](#)

Starting Perl from within a script

Given the following file named p3

```
#!/usr/local/bin/perl -w
```

```
while (<>)  
{  
    print;  
    print "-----\n";  
}
```

p3 pres1

The above line starts Perl from within the script called p3. The data file being used is pres1. The first line of this script that looks like a comment has a very special action. Because it is the first line and it starts with the `#!` symbol combination, the UNIX shell will start the command specified. That command is perl found in the `/usr/local/bin` directory (This might be a different directory on other UNIX systems). To find where the executable of Perl is located on your UNIX system type "which perl" at the UNIX command prompt. It also is starting Perl with warning messages (`-w` option) turned on. The action for this script will be to print each line in the pres1 file with a dashed line immediately after the line.

NOTE: This first line startup has no meaning in a Microsoft operating system.

[Go to Top](#)

Command Line Options

Getting help on command line options

[perl -h](#)

Checking the syntax of your command (Does not execute)

```
perl -w -c -e 'print <>;' <pres1
```

Display warning messages

```
perl -w -e 'while(<>){ chomp; print;}' pres1
```

Examples:

[warning_messages](#)

[output](#)

© B.Huseby, 2003-04