Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

Perl Programming

Student Guide - Volume 2

DTP-250 Rev C

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

D61834GC20
Edition 2.0
May 2010
D66909

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Sun Microsystems, Inc. Disclaimer

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided.Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

This page intentionally left blank.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

This page intentionally left blank.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

Table of Contents

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@oktesson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

About This Course

Course Goals

Upon completion of this course, you should be able to:

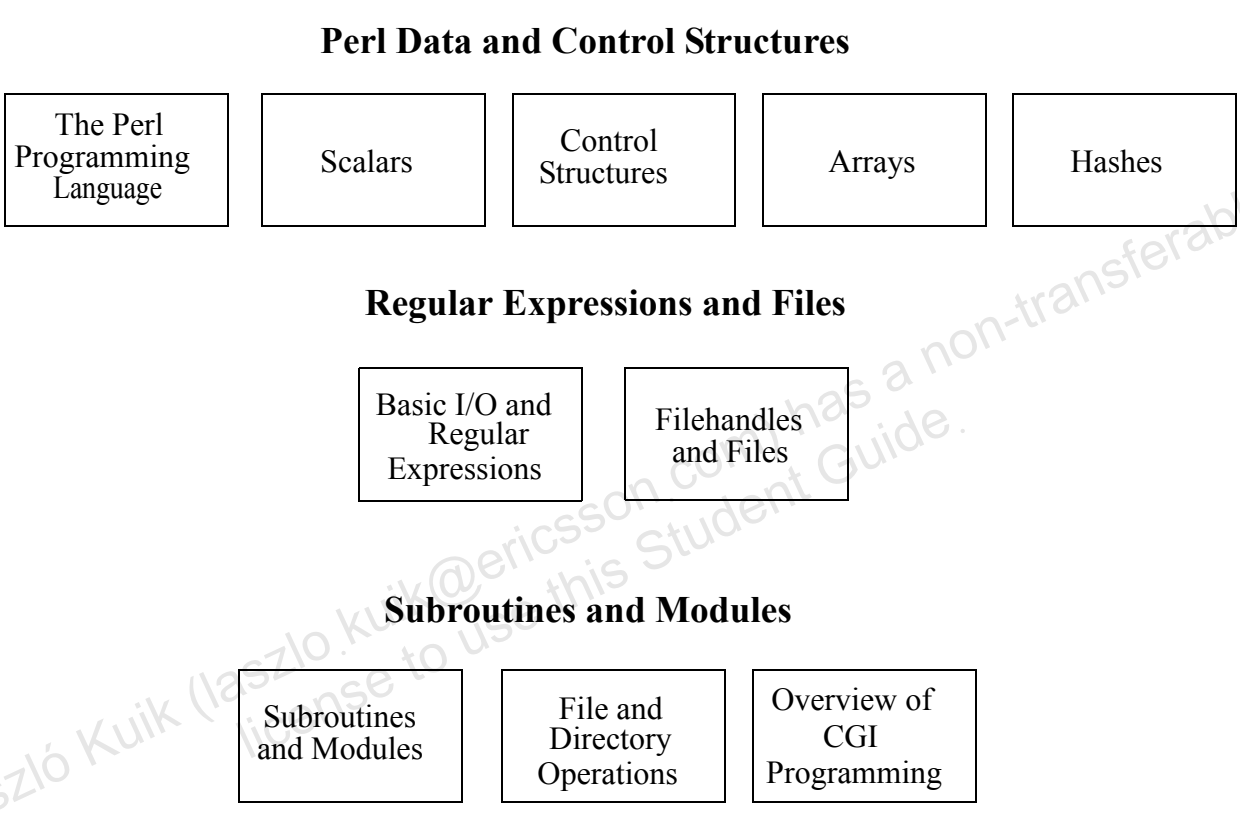
- Create Perl scripts using scalars, arrays, hashes, and control structures
- Create Perl scripts that perform file I/O and modification
- Create Perl subroutines, packages, and modules using the concepts and data structures described in the course

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Course Map

The following course map enables you to see what you will accomplish and where you will go in reference to the course goals.



Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Topics Not Covered

This course does not cover the following topics. Many of these topics are covered in other courses offered by Sun Services:

- Shell programming – Covered in SA-245: *Shell Programming for System Administrators*

Refer to the Sun Services catalog for specific information and registration.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

How Prepared Are You?

To be sure you are prepared to take this course, can you answer yes to the following questions?

- Can you use the `vi` text editor?
- Can you interact with the Solaris™ 10 Operating System as an end user?
- Can you use a graphical user interface?

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Introductions

Now that you have been introduced to the course, introduce yourself to the other students and the instructor, addressing the items listed.

- Name
- Company affiliation
- Title, function, and job responsibility
- Experience related to topics presented in this course
- Reasons for enrolling in this course
- Expectations for this course

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

How to Use Course Materials

To enable you to succeed in this course, these course materials use a learning module that is composed of the following components:

- **Goals** – You should be able to accomplish the goals after finishing this course and meeting all of its objectives.
- **Objectives** – You should be able to accomplish the objectives after completing a portion of instructional content. Objectives support goals and can support other higher-level objectives.
- **Lecture** – The instructor will present information specific to the objective of the module. This information will help you learn the knowledge and skills necessary to succeed with the activities.
- **Activities** – The activities take on various forms, such as an exercise, self-check, discussion, and demonstration. Activities are used to facilitate mastery of an objective.
- **Visual aids** – The instructor might use several visual aids to convey a concept, such as a process, in a visual form. Visual aids commonly contain graphics, animation, and video.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@enocsp.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Icons

Additional resources – Indicates other references that provide additional information on the topics described in the module.

Discussion – Indicates a small-group or class discussion on the current topic is recommended at this time.

Note – Indicates additional information that can help students but is not crucial to their understanding of the concept being described. Students should be able to understand the concept or complete the task without this information. Examples of notational information include keyword shortcuts and minor system adjustments.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

Use `ls -al` to list all files.
`system% You have mail.`

`Courier` is also used to indicate programming constructs, such as variable name, subroutines, and keywords; for example:

The `chomp` function can be used to remove all newline characters from incoming lines.

The `print` statement can be used to print a string.

`Courier bold` is used for characters and numbers that you type; for example:

To list the files in this directory, type:
`# ls`

`Courier italic` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

Type **`chmod a+rwx filename`** to grant read, write, and execute rights for filename to world, group, and users.

Palatino italic is used for book titles, new words or terms, or words that need to be emphasized; for example:

Read Chapter 6 in the *User's Guide*.
These are called *class* options.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Subroutines and Modules

Objectives

Upon completion of this module, you should be able to:

- Create a script that uses the `strict` pragma
- Create subroutines that accept passed parameters and return desired results based on the values passed
- Include a subroutine that uses the `my` operator to create private variables
- Use a Perl library file in your script
- Use a Perl package in your script
- Create a Perl module and call it from a script

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Relevance

Discussion – The following questions are relevant to understanding subroutines and modules:

- Why would it be a good idea to break programs into smaller parts?
- What are the advantages of making scripts shareable?

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Introducing Subroutines and Modules

As programs grow and increase in complexity, additional tools are needed to structure them. The flow of the main program should always be kept readable. Subroutines group related statements together and allow code to be reused. They make scripts easy to modify and maintain over time.

In addition to subroutines you create, many public subroutine libraries and modules exist. Perl's extensive collection of libraries and modules reduces development time, because a module exists for almost every problem.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Subroutines (User-Defined Functions)

There is no difference between user-defined functions and subroutines in Perl. A subroutine (or function) is defined by the keyword sub, followed by its name and a block of statements in curly braces.

```
sub subroutinename {
    statement;
    statement;
...
}
```

Like many other Perl constructs, subroutines have their own namespace.

Invocation

A subroutine is usually called by name followed by parentheses. It can be called independently or as part of a statement.

```
sub();
$xyz = sub1() + sub2();
&sub();
```

Earlier versions of Perl required an `&` before a subroutine name to indicate that the variable was a subroutine. This is just like a preceding a scalar with a `$` or an array with an `@`. Perl 5 no longer requires the `&` before a subroutine name.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Subroutine Example

The script that follows includes a subroutine that prints a text message.

e0801a.plx

```
  1 #!/usr/bin/perl -w
  2
  3 # A Basic Subroutine
  4
  5 hello();
  6
  7 sub hello{
  8   print "Hello World!\n";
  9 }
```

```
$ e0801a.plx
Hello World!
```

The `hello` subroutine prints “Hello World” and demonstrates the basic structure of a subroutine. The subroutine is called on Line 5 by specifying parentheses after the name. This tells Perl to execute the subroutine. However, this is not the only way to call a subroutine.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The following script demonstrates the various ways a subroutine may be called.

e0801b.plx

```
 1 #!/usr/bin/perl -w
 2
 3 hello();
 4 &hello();
 5 &hello;
 6 hello;
 7
 8 sub hello{
 9   print "Hello World!\n";
10 }
```

```
$ e0801b.plx
Unquoted string "hello" may clash with future reserved word
at ./e0801a.plx line 6.
Useless use of a constant in void context at ./e0801a.plx
line 6.
Hello World!
Hello World!
Hello World!
```

The subroutine calls on Lines 3–5 all execute fine. However, the call on Line 6 produces an error. Line 3 shows the Perl-5 way of calling a subroutine, which is the name followed by parentheses. Line 4 is an optional method using the special identifier for subroutines, `&`. The `&` identifies subroutines just like `@` identifies arrays and `%` identifies hashes. However, `&` is optional and no longer desirable. Line 5 demonstrates another method: `&` can be used without `()`.

Line 6, however, produces an error. To call a subroutine in this way, one additional line must be added to the script.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The following script demonstrates the use of a predeclared subroutine. By predeclaring, Perl is given a hint that the subroutine code is coming.

```
e0801c.plx
  1 #!/usr/bin/perl -w
  2
  3 sub hello;
  4
  5 hello();
  6 &hello();
  7 &hello;
  8 hello;
  9
 10 sub hello{
 11   print "Hello World!\n";
 12 }

$ e0801c.plx
Hello World!
Hello World!
Hello World!
Hello World!
```

By predeclaring or predefining the subroutine (that is, placing it before its first invocation), it is possible to call the subroutine on Line 8 without producing an error.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Return Values

In Perl, a subroutine always returns a value after executing. A subroutine is always a part of some expression and, therefore, must return a value. The value returned is either the value passed in the `return` statement or the value returned by the last expression evaluated in the subroutine.

For example, the following script calls a subroutine that prints three names.

```
e0802.plx
  1 #!/usr/bin/perl -w
  2
  3 $ret = names();
  4 print "Returned value: $ret\n";
  5
  6 sub names{
  7   print "Larry\n";
  8   print "Moe\n";
  9   print "Curly\n";
 10 }
```

```
$ e0802.plx
Larry
Moe
Curly
Returned value: 1
```

Why does the subroutine return a 1? The return value of a successful `print` command is 1. Because a `print` command was the last expression in the subroutine, a 1 is returned.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

If the `return` statement is added to the script, the result changes to the value specified with the `return` statement.

```
e0803.plx
  1 #!/usr/bin/perl -w
  2
  3 $ret = names();
  4 print "Returned value: $ret\n";
  5
  6 sub names{
  7   print "Larry\n";
  8   print "Moe\n";
  9   return "Joe";
 10   print "Curly\n";
 11 }

$ e0803.plx
Larry
Moe
Returned value: Joe
```

Notice how the `return` statement changes the output. First, since “Joe” is returned, that string value is printed by the script. When the subroutine encounters a `return` statement, no other statements in the block are executed. Therefore, the final `print` statement in the subroutine is skipped.

Note – Returning a zero indicates a subroutine did not complete successfully.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Passing Parameters

Usually, parameters pass data to a subroutine. The parameters are passed in parentheses when the subroutine is invoked.

```
func($par1, $par2, $par3);
func(@par);
```

When a subroutine executes, the passed parameters are available in the special array `@_`. The first parameter is stored in `$_[0]`, the second in `$_[1]`, and so on.

The following script takes a string, makes it uppercase, and returns the result.

```
e0804.plx
  1 #!/usr/bin/perl -w
  2
  3 $name = "Larry";
  4
  5 $ret = myuc($name);
  6
  7 print "Value returned: $ret\n";
  8
  9 sub myuc{
 10   $_[0] = uc($_[0]);
 11   return $_[0];
 12 }

$ e0804.plx
Value returned: LARRY
```

The variable is passed to the subroutine `&myuc`. The string is returned in uppercase, which is the desired result. Using parameters is easy. However, look at the script more closely.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Passing by Reference

The values stored in `@_` are references to the original variables passed to the subroutine, not copies. Working with `@_` directly modifies the original variables passed to the subroutine.

For example, if the script is updated to print out a little more information, the following output is produced.

```
e0805.plx
  1 #!/usr/bin/perl -w
  2
  3 $name = "Larry";
  4 print "\$name before sub = $name\n";
  5
  6 $ret = myuc($name);
  7
  8 print "\$name after sub = $name\n";
  9 print "Value returned: $ret\n";
 10
 11 sub myuc{
 12   $_[0] = uc($_[0]);
 13   return $_[0];
 14 }

$ e0805.plx
$name before sub = Larry
$name after sub = LARRY
Value returned: LARRY
```

The subroutine has not only changed `$ret`, but it has also changed the value of `$name`.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Figure 8-1 shows the relationship between `$_[0]` and `$name`.

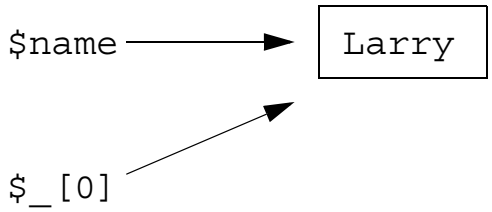


Figure 8-1 Passing by Reference

The values stored in `@_` are not copies of the variables passed to the subroutine; they are references to the actual variable themselves. In effect, `$name` and `$_[0]` modify the same value stored in memory. So when `$_[0]` is modified, so is `$name`.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Passing by Value

To solve this problem, you can write the previous example so that a temporary variable stores the value passed to the subroutine. This is known as passing variables by value.

e0806.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $name = "Larry";
 4 print "\$name before sub = $name\n";
 5
 6 $ret = myuc($name);
 7
 8 print "\$name after sub = $name\n";
 9 print "Value returned: $ret\n";
10
11 sub myuc{
12   $copy_of_name = $_[0];
13   $copy_of_name = uc($copy_of_name);
14   return $copy_of_name;
15 }
```

```
$ e0806.plx
$name before sub = Larry
$name after sub = Larry
Value returned: LARRY
```

Now, changes are made to a copy and the original variable is not affected.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Figure 8-2 shows how the two variables point at different values in memory.

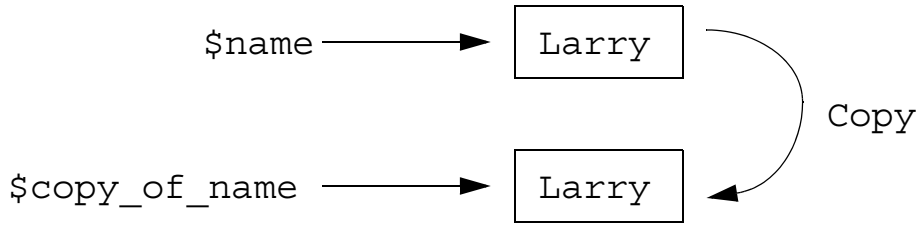


Figure 8-2 Passing by Value

The value stored in `$name` is copied to the new variable `$copy_of_name`.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Scope

Scope is the concept of where a variable has meaning. Perl variables are global by default. Variables can be accessed anywhere within a script, even if used for the first time in a subroutine. For example, in the previous script, the $copy_of_name variable can be accessed anywhere within the script. The following script demonstrates this.

e0807.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $name = "Larry";
 4 print "\$name before sub = $name\n";
 5
 6 $ret = myuc($name);
 7
 8 print "\$name after sub = $name\n";
 9 print "Value returned: $ret\n";
10 print "\$copy_of_name is $copy_of_name\n";
11
12 sub myuc{
13   $copy_of_name = $_[0];
14   $copy_of_name = uc($copy_of_name);
15   return $copy_of_name;
16 }
```

```
$ e0807.plx
$name before sub = Larry
$name after sub = Larry
Value returned: LARRY
$copy_of_name is LARRY
```

Notice on Line 10, that the variable can be used even though it is not initialized until the subroutine is called. You can use the variable anywhere in the script. Thus, it has scope anywhere in the script. This behavior might produce unwanted side effects if variables are reused by accident in a script.

It is preferable to limit the scope of variables to a subroutine. With Perl's scoping mechanisms, a variable's life can be limited to only a subroutine's code block. This prevents side effects and makes subroutines more modular.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@edlesson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

my Variables

Variables declared with the my keyword have meaning only within the code block in which they are declared. Normally this is a code block inside a subroutine, but it could be any code block.

For example, the following script declares the variable $hero three times. What is the value of $hero at the end of the script?

```
e0808.plx
  1 #!/usr/bin/perl -w
  2 # Scope and my
  3 $hero = "Batman";
  4
  5 print "\$hero in main: $hero\n";
  6
  7 {
  8   my $hero = "Robin";
  9   print "\$hero in block1: $hero\n";
 10 }
 11
 12 {
 13   my $hero = "Batgirl";
 14   print "\$hero in block2: $hero\n";
 15 }
 16
 17 print "\$hero in main: $hero\n";
```

```
$ e0808.plx
$hero in main: Batman
$hero in block1: Robin
$hero in block2: Batgirl
$hero in main: Batman
```

The $hero variables declared in the two code blocks have scope only in those code blocks. When the statements in the block are complete, the variable is released from memory. Therefore, when the final print statement is executed, the original value for $hero remains unchanged.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

When this principle is applied to a subroutine, the results are the same. The following script uses the variable `$sidekick` in the main script and in the`printsk` subroutine. Notice that the original value is unchanged even after `$sidekick` is used in the subroutine.

```
e0809.plx
  1 #!/usr/bin/perl -w
  2 $hero = "Batman"; $sidekick = "Robin"; $car =
"Batmobile";
  3
  4 print "$hero\'s sidekick is $sidekick\n";
  5 print "$hero\'s car is a $car\n\n";
  6
  7 printsk();
  8 printcar();
  9
 10 print "$hero\'s sidekick is $sidekick\n";
 11 print "$hero\'s car is a $car\n";
 12
 13 sub printsk{
 14   my $sidekick = "Batgirl";
 15   print "$hero\'s sidekick is: $sidekick\n";
 16 }
 17
 18 sub printcar{
 19   print "$hero\'s car is a $car\n\n";
 20 }
```

```
$ e0809.plx
Batman's sidekick is Robin
Batman's car is a Batmobile

Batman's sidekick is: Batgirl
Batman's car is a Batmobile

Batman's sidekick is Robin
Batman's car is a Batmobile
```

The `$sidekick` declared on Line 14 exists only in the subroutine. Its scope is limited to Lines 13–16. Once the subroutine ends, that variable and its value disappear.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

local Variables

The local statement broadens the scope of variables. In addition to being visible in a subroutine, local variables are also visible in subroutines called from within the block in which they are declared. To demonstrate, a new subroutine, &printall, has been added to the previous script.

e0810a.plx

```
 1 #!/usr/bin/perl -w
 2 $hero = "Batman"; $sidekick = "Robin"; $car = "Batmobile";
 3
 4 print "$hero\'s sidekick is $sidekick\n";
 5 print "$hero\'s car is a $car\n\n";
 6
 7 printall();
 8
 9 print "$hero\'s sidekick is $sidekick\n";
10 print "$hero\'s car is a $car\n";
11
12 sub printall{
13   local $car = "Pinto";
14    printsk();
15    printcar();
16 }
17
18 sub printsk{
19   my $sidekick = "Batgirl";
20   print "$hero\'s sidekick is: $sidekick\n";
21 }
22
23 sub printcar{
24   print "$hero\'s car is a $car\n\n";
25 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

```
$ e0810a.plx
Batman's sidekick is Robin
Batman's car is a Batmobile

Batman's sidekick is: Batgirl
Batman's car is a Pinto

Batman's sidekick is Robin
Batman's car is a Batmobile
```

When `$car` is declared with `local` on Line 13, it has scope in `printsk` or `printcar`. Therefore, the value of the `local` variable `$car` is printed when line 7 is called, not the original value. The original value for `$car` remains unchanged outside the subroutines. When the subroutines end, the values declared and used in them cease to exist.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

What happens if $car is declared with my in printall? Then, $car has meaning or scope only in the printall subroutine. Therefore, the original value of $car is used and the output is the same as in the first example.

e0810b.plx

```
 1 #!/usr/bin/perl -w
 2 $hero = "Batman"; $sidekick = "Robin"; $car = "Batmobile";
 3
 4 print "$hero\'s sidekick is $sidekick\n";
 5 print "$hero\'s car is a $car\n\n";
 6
 7 printall();
 8
 9 print "$hero\'s sidekick is $sidekick\n";
10 print "$hero\'s car is a $car\n";
11
12 sub printall{
13   my $car = "Pinto";
14    printsk();
15    printcar();
16 }
17
18 sub printsk{
19   my $sidekick = "Batgirl";
20   print "$hero\'s sidekick is: $sidekick\n";
21 }
22
23 sub printcar{
24   print "$hero\'s car is a $car\n\n";
25 }
```

```
$ e0810b.plx
Batman's sidekick is Robin
Batman's car is a Batmobile

Batman's sidekick is: Batgirl
Batman's car is a Batmobile

Batman's sidekick is Robin
Batman's car is a Batmobile
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Pragmas

One of the features of Perl is its pragmatic modules or pragmas. Pragmas alter the compilation or execution of your Perl programs. Essentially, they give the compiler hints about how to handle the code. Pragmas are invoked using the `use` keyword. For this course, there is one particular pragma of interest.

```
use strict;
```

The `strict` pragma requires that all variables in a script be declared as `my` or as explicitly specified global variables before they are used. Each variable must be bound to a specific code block. For example, if the `strict` pragma is added to the previous example, the following error is produced:

```
$ e0810b.plx
Global symbol "$hero" requires explicit package name at
./e0810b.plx line 4.
Global symbol "$sidekick" requires explicit package name at
./e0810b.plx line 5.
Global symbol "$car" requires explicit package name at
./e0810b.plx line 6.
Execution of ./e0810b.plx aborted due to compilation errors.
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

In fact, to get the program to run, these variables must be initialized as `my` or global variables. (Global variables are described in “Global Variables, Package Variables, and `strict`” on page 8-34.) The previous script updated to use `my` looks like the following:

e0810c.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 my $hero = "Batman";
 5 my $sidekick = "Robin";
 6 my $car = "Batmobile";
 7
 8 print "$hero\'s sidekick is $sidekick\n";
 9 print "$hero\'s car is a $car\n\n";
10
11 printall();
12
13 print "$hero\'s sidekick is $sidekick\n";
14 print "$hero\'s car is a $car\n";
15
16 sub printall{
17   my $car = "Pinto";
18    printsk();
19   printcar($car);
20 }
21
22 sub printsk{
23   my $sidekick = "Batgirl";
24   print "$hero\'s sidekick is: $sidekick\n";
25 }
26
27 sub printcar{
28   my $car = $_[0];
29   print "$hero\'s car is a $car\n\n";
30 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

```
$ e0810c.plx
Batman's sidekick is Robin
Batman's car is a Batmobile

Batman's sidekick is: Batgirl
Batman's car is a Pinto

Batman's sidekick is Robin
Batman's car is a Batmobile
```

All the variables are declared with `my`. The variables have scope only in their own code block. Therefore, if values are to be used in a code block, they must be passed to or declared in that block. From a style perspective, this is a much better way to write code.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Context Sensitivity

A context-sensitive subroutine returns the correct data type whether it is in a scalar or list context. Several built-in subroutines include this feature. For example, `split` provides either the list of substrings or their number, depending on the context. Another example is `localtime`.

e0811.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 # return the list ($sec, $min, $hour,
 5 # $mday, $mon, $year, $wday, $yday, $isdst)
 6 my @time = localtime() ;
 7
 8 foreach (@time){
 9   print "$_--";
10 }
11 print "\n";
12
13 # returns the string "Thu Mar 18 14:48:16 2010"
14 my $time = localtime() ;
15 print "$time\n";
```

```
$ e0811.plx
16--48--14--18--2--110--4--76--1--
Thu Mar 18 14:48:16 2010
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

To implement context sensitivity in subroutines, use the command `wantarray`. This command determines whether the caller wants to assign the returned value to a list or to a scalar variable. In the first case, it returns true; in the second case, it returns false.

e0812.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 my $str = "a:b:c";
 5
 6 my @arr1 = split3($str); #array
 7 my $length = split3($str); #scalar
 8
 9 print "The values in \$str are:\n";
10
11 foreach (@arr1){
12   print "$_ ";
13 }
14 print "\nThe length is $length\n";
15
16 sub split3 {
17   my $tempstr = $_[0];
18   my @list = split /:/, $tempstr;
19
20   my $len = @list;
21
22   return @list if wantarray();
23   return $len;
24 }
```

```
$ e0812.plx
The values in $str are:
a b c
The length is 3
```

Lines 6 and 7 call the subroutine in an array and scalar context. If called in a list context, Line 22 is executed, and Line 23 is skipped. But if `wantarray` returns false, Line 23 is returned with its scalar value.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Libraries

Up to now, the use of subroutines has been limited to one Perl script. For subroutines to be truly useful, the subroutines must be shared between files. The easiest way to do this is by using Perl libraries. Libraries are an older way of sharing subroutines, and their creation and use in publicly-available software is discouraged. Packages and modules are strongly preferred. However, you may still encounter the use of libraries, and Perl comes with several libraries for backward compatibility, it also comes with modules that perform the same tasks so that converting old code is easy. The first step to creating a library is to create subroutines. The script that follows builds a basic math library.

e0813.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 my $x = 2; my $y = 3;
 5
 6 my $sum = add($x, $y);
 7 my $diff = subtract($y, $x);
 8 my $product = mult($x, $y);
 9
10 print "Sum = $sum\n";
11 print "Diff = $diff\n";
12 print "Product = $product\n";
13
14 sub add {
15   my ($a, $b) = ($_[0], $_[1]);
16   return $a + $b;
17 }
18
19 sub subtract {
20   my ($a, $b) = ($_[0], $_[1]);
21   return $a - $b;
22 }
23
24 sub mult {
25   my ($a, $b) = ($_[0], $_[1]);
26   return $a * $b;
27 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

```
$ e0813.plx
Sum = 5
Diff = 1
Product = 6
```

The previous script includes three subroutines, `add`, `subtract`, and `mult`. Lines 6– 8 demonstrate them in action. Each takes two numbers as arguments and performs a simple math operation.

Creating a Library

The first step in creating a library is to save the subroutines in a separate file with a `.plx` extension. For this library, the file is saved as `mathlib.plx` as shown.

mathlib.pl

```
 1 # My math lib
 2
 3 sub add {
 4   my ($a, $b) = ($_[0], $_[1]);
 5   return $a + $b;
 6 }
 7
 8 sub subtract {
 9   my ($a, $b) = ($_[0], $_[1]);
10   return $a - $b;
11 }
12
13 sub mult {
14   my ($a, $b) = ($_[0], $_[1]);
15   return $a * $b;
16 }
17 1;
```

Notice the differences from the previous file. The first comment line is removed and no pragma is required. A last line of `1;` is added. When a file is used as a library, package, or module, Perl expects the last expression in the file to return a true value. Entering `1;` on a line by itself does this.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Now the library file is ready. The script that calls the subroutines is created next. In this script, a number of steps must be added to the front of the file to access the library.

- The directory where the library is stored must be added to the `@INC` array. Remember from Module 1, "The Perl Programming Language," that the `@INC` array is searched by Perl for libraries and modules to include in scripts.
- The `require` statement must be added to the script. This tells Perl at runtime to go out and find the file listed and make the subroutines there available to the calling script.
- After the previous two steps are done, the subroutines can be called by name as before.

An updated version of the script that calls the new library follows.

e0814.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 # Move directory to the front of array
 5 unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
 6 require "mathlib.plx";
 7
 8 my $x = 2; my $y = 3;
 9
10 my $sum = add($x, $y);
11 my $diff = subtract($y, $x);
12 my $product = mult($x, $y);
13
14 print "Sum = $sum\n";
15 print "Diff = $diff\n";
16 print "Product = $product\n";
```

Line 5 takes the directory where the library is located and moves it to the front of the `@INC` array. Next, on Line 6, the `require` statement tells Perl to find this file and use its subroutines in this script. After Lines 5 and 6 execute successfully, the subroutines on Lines 10–13 can be called from the script. All this takes place at runtime.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Limitations

The main limitation of this approach is subroutine names must be unique. What if a script writer using your library were to add a new `add` function?

e0815.plx
```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
 5 require "mathlib.plx";
 6
 7 my $x = 2; my $y = 3;
 8
 9 my $sum = add($x, $y);
10 my $diff = subtract($y, $x);
11 my $product = mult($x, $y);
12
13 print "Sum = $sum\n";
14 print "Diff = $diff\n";
15 print "Product = $product\n";
16
17 sub add { # Duplicate Add module
18   my ($a, $b) = ($_[0], $_[1]);
19   return $a + $b + 1;
20 }
```

When the script is executed, the following warning is issued.

```
$ e0815.plx
Subroutine add redefined at mathlib.pl line 3.
Sum = 5
Diff = 1
Product = 6
```

The subroutine added to the script is ignored, and the subroutine from the library is executed instead. To prevent this and put libraries in their own namespace, use the `package` statement.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using an Existing Library

But before going into packages, how might you use an existing Perl library? A good example of an existing Perl library is `find.plx`. This library emulates the UNIX `find` command. With this library, it is easy to write your own scripts to walk a file tree and perform operations based on what is found.

myfind.pl

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 require "find.plx";
 4
 5 my $targetdir = $ARGV[0];
 6 my $regex = $ARGV[1];
 7
 8
 9 find "$targetdir";
10 print "\n";
11
12 sub wanted {
13   if (/$regex/){
14     print "$_ ";
15   }
16 }
```

```
$ myfind.pl . .pl
e0801a.plx  e0801b.plx  e0801c.plx  e0802.plx  e0803.plx
e0804.plx  e0805.plx  e0806.plx  e0807.plx  e0808.plx
e0809.plx  e0810a.plx  e0810b.plx  e0810c.plx  e0811.plx
e0812.plx  e0813.plx  e0814.plx  e0815.plx  e0816a.plx
e0816b.plx  e0816c.plx  e0817.plx  e0818.plx  myfind2.pl
myfind.pl  mathexp.pl  mathpack.pl  textwrap.pl temp.plx
mathlib.pl
```

Line 9 calls the `find` subroutine. This uses a user-defined subroutine named `wanted` to search a given directory tree. When a match is found on Line 13, a code block is executed that can perform whatever operations you want on the matched file or directory. In this example, the matched file is printed to the screen. However, there are a number of possibilities with this library.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Packages

Packages allow included files to contain variables and subroutines with the same names found in the main script. When the `package` statement is used, Perl stores variables and subroutines in a different symbol table. This means the included file has its own namespace. To make an included file a package, add a line like the following to the included file.

```
package package_name;
```

This changes the subroutine library file and the way subroutines are called from it. Below, a new library file is created with the `package` statement included.

mathpack.pl

```
 1 # Math package
 2
 3 package mathpack;
 4
 5 sub add {
 6   my ($a, $b) = ($_[0], $_[1]);
 7   return $a + $b;
 8 }
 9
10 sub subtract {
11   my ($a, $b) = ($_[0], $_[1]);
12   return $a - $b;
13 }
14
15 sub mult {
16   my ($a, $b) = ($_[0], $_[1]);
17   return $a * $b;
18 }
19 1;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The way the subroutines are called when using a package is different. To call a subroutine of another package, you must place the package name in front of the subroutine name with a double colon as separator. It is the same with variables. If subroutines are called in the usual way, without a package name as prefix, they are taken from the current package. So, the previous script changes as follows:

e0816a.plx
```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
 5 require "mathpack.plx";
 6
 7 my $x = 2; my $y = 3;
 8
 9 my $sum = mathpack::add($x, $y);
10 my $diff = mathpack::subtract($y, $x);
11 my $product = mathpack::mult($x, $y);
12
13 print "Sum = $sum\n";
14 print "Diff = $diff\n";
15 print "Product = $product\n";
16
17 my $localsum = add($x, $y);
18 print "LocalSum = $localsum\n";
19
20 sub add { # Duplicate Add module
21   my ($a, $b) = ($_[0], $_[1]);
22   return $a + $b + 1;
23 }
```

```
$ e0816a.plx
Sum = 5
Diff = 1
Product = 6
LocalSum = 6
```

First, notice on Line 5 that the new library file that has the `package` statement is included. Now, all calls to subroutines found in the `mathpack` package are preceded by `mathpack::`. Line 17 demonstrates how local subroutine calls are still the same. Now both the package file and the main script include an `add` subroutine. However, the subroutines reside in different namespaces and no longer conflict.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

One more important point to mention is that any Perl script has a default package created for it. The main script in `e0816.plx` is actually part of the default `main::` package. That is why some of the warnings and error messages displayed so far precede subroutine and variable names with `main::`. To demonstrate, `e0816.plx` is written as follows using the package name to call the `add` subroutine.

```
e0816b.plx
  1 #!/usr/bin/perl -w
  2 use strict;
  3
  4 unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
  5 require "mathpack.plx";
  6
  7 my $x = 2; my $y = 3;
  8
  9 my $sum = mathpack::add($x, $y);
 10 my $diff = mathpack::subtract($y, $x);
 11 my $product = mathpack::mult($x, $y);
 12
 13 print "Sum = $sum\n";
 14 print "Diff = $diff\n";
 15 print "Product = $product\n";
 16
 17 my $localsum = main::add($x, $y);
 18 print "LocalSum = $localsum\n";
 19
 20 sub add { # Duplicate Add module
 21   my ($a, $b) = ($_[0], $_[1]);
 22   return $a + $b + 1;
 23 }
```

Line 17 shows an alternative way of calling the `add` routine in the main module.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Global Variables, Package Variables, and `strict`

The variables used in previous modules have been global variables. Remember, in the previous discussion that the `strict` pragma throws errors when variables are declared as `$x = 1;`. To use global variables under `strict`, they must be defined and accessed using their package names. Because the default package name is `main`, the previous example can be rewritten to make `$localsum` a global variable.

e0816c.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
 5 require "mathpack.plx";
 6
 7 my $x = 2; my $y = 3;
 8
 9 my $sum = mathpack::add($x, $y);
10 my $diff = mathpack::subtract($y, $x);
11 my $product = mathpack::mult($x, $y);
12
13 print "Sum = $sum\n";
14 print "Diff = $diff\n";
15 print "Product = $product\n";
16
17 main::localsum = main::add($x, $y);
18 print "LocalSum = $main::localsum\n";
19
20 sub add { # Duplicate Add module
21   my ($a, $b) = ($_[0], $_[1]);
22   return $a + $b + 1;
23 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

```
$ e0816c.plx
Sum = 5
Diff = 1
Product = 6
LocalSum = 6
```

Lines 17 and 18 demonstrate two concepts. First, they show how a global variable is declared and referenced when using the `strict` pragma. Second, they show that this is also the method used to access variables declared in a package. With this format, a dollar sign and the package name precedes the actual variable name. For example, to access a variable `$abc` declared in `mathpack`, you would write `$mathpack::abc`.

Up to now, all files have been included at runtime. However, what if you want to include files at compile time? To do this, modules are used.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Modules

Modules are the standard way of including subroutines in Perl scripts. What is a module? Typically, it is a file with the same name as its package definition but with a `.pm` extension. To continue the current example, change the name and file name to make it a module.

Mathmod.pm

```
 1 # Math package
 2
 3 package Mathmod;
 4
 5 sub add {
 6   my ($a, $b) = ($_[0], $_[1]);
 7   return $a + $b;
 8 }
 9
10 sub subtract {
11   my ($a, $b) = ($_[0], $_[1]);
12   return $a - $b;
13 }
14
15 sub mult {
16   my ($a, $b) = ($_[0], $_[1]);
17   return $a * $b;
18 }
19 1;
```

First, on Line 3, the package name is changed to reflect the fact that the file is now a module. The file name is changed to `Mathmod.pm` instead of `mathpack.plx`. Note the package and file name start with an uppercase letter, unless it is a pragma. This convention is followed consistently to prevent confusion. Starting your modules with an uppercase letter prevents potential naming conflicts.

Modules are typically included in scripts at compile time using the `use` statement. For example, to include this module in a script the command is:

```
use Mathmod;
```

The `.pm` extension is assumed.

However, `use` includes a file at compile time. Remember from earlier that the `@INC` array is modified at runtime so library files are found.

```
unshift @INC, "$ENV{HOME}/lf/mod08/examples/";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

This does not help the program at compile time.

The BEGIN and END Blocks

There are two special code blocks in Perl that control when a piece of code is compiled or executed. In a `BEGIN { ... }` block, all commands are executed at compile time. In an `END { ... }` block, all commands are executed after a subroutine or the main program is aborted, even if the program is terminated by `die`.

So, the solution to this problem appears to be the use of a `BEGIN` block.

```
BEGIN {unshift @INC, "/home/mydir/lf/mod08/examples/"};
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

So now the @INC array is updated at compile time, and, instead of using a library or package, a module is used. The script is modified as follows.

e0817.plx
```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 BEGIN{ unshift @INC, "$ENV{HOME}/lf/mod08/examples/"; }
 5 use Mathmod;
 6
 7 my $x = 2; my $y = 3;
 8
 9 my $sum = Mathmod::add($x, $y);
10 my $diff = Mathmod::subtract($y, $x);
11 my $product = Mathmod::mult($x, $y);
12
13 print "Sum = $sum\n";
14 print "Diff = $diff\n";
15 print "Product = $product\n";
16
17 my $localsum = add($x, $y);
18 print "LocalSum = $localsum\n";
19
20 sub add { # Duplicate Add module
21   my ($a, $b) = ($_[0], $_[1]);
22   return $a + $b + 1;
23 }
```

```
$ e0817.plx
Sum = 5
Diff = 1
Product = 6
LocalSum = 6
```

The package names are updated to reflect the new name for the package. Instead of using require, use use instead.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The `use lib` Pragma

With Perl 5, there is a better way to manipulate the `@INC` array. Included as part of the Perl distribution is the `lib` pragma. This module is specifically designed to modify the `@INC` array at compile time. It is essentially equivalent to the following statement.

```
BEGIN {unshift @INC, "$ENV{HOME}/lf/mod08/examples/"};
```

This statement has been used in all the previous examples. To use the `lib` pragma, the script changes as follows.

e0818.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 use lib "/$ENV{HOME}/lf/mod08/examples/";
 4 use Mathmod;
 5
 6 my $x = 2; my $y = 3;
 7
 8 my $sum = Mathmod::add($x, $y);
 9 my $diff = Mathmod::subtract($y, $x);
10 my $product = Mathmod::mult($x, $y);
11
12 print "Sum = $sum\n";
13 print "Diff = $diff\n";
14 print "Product = $product\n";
15
16 my $localsum = add($x, $y);
17 print "LocalSum = $localsum\n";
18
19 sub add { # Duplicate Add module
20   my ($a, $b) = ($_[0], $_[1]);
21   return $a + $b + 1;
22 }
```

Thus, the module has changed from a simple library to a full-fledged module.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Notes

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Notes

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using Third-Party, Standard, and CPAN Modules

In the last section, you learned how to write and use your own modules. Modules developed by other individuals are used in exactly in the same way. First, copy the modules into the desired directory. Then, in your program, modify `@INC` if necessary and bind and import the new module with the `use` command. Finally, call the imported subroutines in the usual way.

Perl is delivered with many modules that provide a large range of frequently needed features. They are called standard modules and are located in standard directories of `@INC`. Each has its own man page so that you can get the necessary information about the contained subroutines.

Another huge source for modules is the Comprehensive Perl Archive Network (CPAN), which can be accessed at `http://www.perl.com` or `http://www.cpan.org`. There are more than 1500 freely available modules in this archive, sorted by several criteria. It is always a good idea to take a look at CPAN before writing your own code for a specific problem.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using a Perl Module

Now that you have created your own module, what follows is an example that uses an existing Perl module. The `Text::Wrap` module is a simple text formatting tool. It takes an array or string and formats the output to wrap at a specified character width.

textwrap.pl

```
  1 #!/usr/bin/perl -w
  2 use strict;
  3
  4 use Text::Wrap;
  5
  6 $Text::Wrap::columns = 55 ;
  7
  8 my $big = "As we enter the next phase of the Internet
build-out, or what we call the Net Effect, traditional
business processes are being converted into more flexible
models. This puts IT in the driver's seat for capitalizing
on new business opportunities and lowering costs.";
  9
 10 print "\n";
 11 print wrap("    ","    ",$big);
 12 print "\n\n";
```

```
$ textwrap.pl

    As we enter the next phase of the Internet
  build-out, or what we call the Net Effect,
  traditional business processes are being converted
  into more flexible models. This puts IT in the
  driver's seat for capitalizing on new business
  opportunities and lowering costs.
```

Line 6 sets the width at which the text is wrapped. The arguments to the `wrap` function are:

```
wrap (Left Margin Space First Line, Left Margin Space,
String);
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise: Create Subroutines and Modules

In this exercise, you complete the following tasks:

- Create a script that uses the `strict` pragma
- Create subroutines that accept passed parameters and return desired results based on the values passed
- Include a subroutine that uses the `my` operator to create private variables
- Use a Perl library file in your script
- Use a Perl package in your script
- Create a Perl module and call it from a script

Tasks

For all exercises in this module, turn warnings on, and use the `strict` pragma.

Complete the following steps:

1. Create a script that reads in a number between 1 and 10 and prints the equivalent number using Roman numerals. Allow the user to enter numbers until `q` is entered. If the number is greater than 10, just print the number.

   Use a subroutine to convert the number to a Roman numeral.

   Hint: Use an array store the Roman numerals.

   Sample output:

```
Enter a number from 1 to 10 or (q)uit: 1
Your number 1 is: I
Enter a number from 1 to 10 or (q)uit: 4
Your number 4 is: IV
Enter a number from 1 to 10 or (q)uit: 6
Your number 6 is: VI
Enter a number from 1 to 10 or (q)uit: 11
Your number 11 is: 11
Enter a number from 1 to 10 or (q)uit: q
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

2. Add to the script you created in Step 1. Instead of getting one number for input, get two numbers. Then, create a subroutine that adds the two numbers together. Your script should print out the two numbers in Roman numerals and the result in Roman numerals if it is less than 11. Thus, if you entered 2 and 2, the script should print `II plus II equals IV`.

   Sample output:

```
Please enter two numbers from 1 to 10.
Enter $a  or (q)uit: 2
Enter $b: 6
II plus VI equals VIII
Enter $a  or (q)uit: 4
Enter $b: 9
IV plus IX equals 13
Enter $a  or (q)uit: q
```

3. Add one more subroutine to this script. Pass the two input values to the new subroutine. Have it call the other two subroutines and return a string; for example, `II plus II equals IV`.

   Sample output:

```
Please enter two numbers from 1 to 10.
Enter $a or (q)uit: 2
Enter $b: 6
II plus VI equals VIII
Enter $a or (q)uit: 5
Enter $b: 9
V plus IX equals 14
Enter $a or (q)uit: q
```

4. Create a script that reads in a list of numbers into an array. After entering the numbers, determine the minimum, maximum, and mean (average) of the list. Create the following subroutines:

   a. Find the minimum and the maximum values.

   b. Compute the mean.

   c. Print the results.

   Hint: To find the minimum and the maximum, perform a test like this inside the loop that iterates through the array.

```
$min = $_ if $min >= $_ ;
$max = $_ if $max <= $_ ;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Sample output:

```
Enter a list of numbers one per line.
Press Ctrl-D when you are finished
4
15
8
25
Min: 4.00 Max: 25.00 Mean: 13.00
```

5. Using the previous script, move all your subroutines into a library file. Produce the same output, but call the routines from the library file. Make any changes necessary to include the library.

6. Convert your library to a package. Make any changes necessary in the main script to call the subroutines in the library file. Produce the same output as before.

7. Convert the library file into a Perl module. Make any changes necessary in the main script to call the subroutines in the module.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Summary

Discussion – Take a few minutes to discuss the experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Solutions

1. Create a script that reads in a number between 1 and 10 and prints the equivalent number using Roman numerals. Allow the user to enter numbers until `q` is entered. If the number is greater than 10, just print the number.

   Use a subroutine to convert the number to a Roman numeral.

   Hint: Use an array store the Roman numerals.

   Sample output:

```
Enter a number from 1 to 10 or (q)uit: 1
Your number 1 is: I
Enter a number from 1 to 10 or (q)uit: 4
Your number 4 is: IV
Enter a number from 1 to 10 or (q)uit: 6
Your number 6 is: VI
Enter a number from 1 to 10 or (q)uit: 11
Your number 11 is: 11
Enter a number from 1 to 10 or (q)uit: q
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0801.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 # Get Number from 1 to 10
 5 my $in = 0;
 6 while ($in ne "q"){
 7   print "Enter a number from 1 to 10 or (q)uit: ";
 8   chomp($in=<STDIN>);
 9   last if ($in eq "q" || $in eq "");
10
11   my $retval = c2roman($in);
12   print "Your number $in is: $retval\n";
13 }
14
15 sub c2roman{
16   my $num = $_[0];
17   my @roman;
18   @roman = qw/I II III IV V VI VII VIII IX X/;
19   if ($num > 0 and $num < 11) { return $roman[$num-1]; }
20   else { return $num ; }
21 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

2. Add to the script you created in Step 1. Instead of getting one number for input, get two numbers. Then, create a subroutine that adds the two numbers together. Your script should print out the two numbers in Roman numerals and the result in Roman numerals if it is less than 11. Thus, if you entered 2 and 2, the script should print `II plus II equals IV`.

   Sample output:

```
Please enter two numbers from 1 to 10.
Enter $a  or (q)uit: 2
Enter $b: 6
II plus VI equals VIII
Enter $a  or (q)uit: 4
Enter $b: 9
IV plus IX equals 13
Enter $a  or (q)uit: q
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0802.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 print "Please enter two numbers from 1 to 10.\n";
 5 my $a = 0;
 6 while ($a ne "q"){
 7   print "Enter \$a  or (q)uit: ";
 8   chomp($a=<STDIN>);
 9   last if $a eq "q" or $a eq "";
10
11     print "Enter \$b: ";
12   chomp($b=<STDIN>);
13
14   my $reta = c2roman($a);
15   my $retb = c2roman($b);
16   my $c = sumab($a, $b);
17   my $retc = c2roman($c);
18   print "$reta plus $retb equals $retc\n";
19 }
20
21 sub c2roman{
22   my $num = $_[0];
23   my @roman;
24   @roman = qw/I II III IV V VI VII VIII IX X/;
25   if ($num > 0 and $num < 11) {  return $roman[$num-1]; }
26   else { return $num ; }
27 }
28
29 sub sumab{
30     my $a = $_[0];
31     my $b = $_[1];
32     return $a + $b;
33 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

3. Add one more subroutine to this script. Pass the two input values to the new subroutine. Have it call the other two subroutines, and return a string; for example, `II plus II equals IV`.

   Sample output:

```
Please enter two numbers from 1 to 10.
Enter $a or (q)uit: 2
Enter $b: 6
II plus VI equals VIII
Enter $a or (q)uit: 5
Enter $b: 9
V plus IX equals 14
Enter $a or (q)uit: q
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0803.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3
 4 my $a = 0; my $b = 0;
 5 print "Please enter two numbers from 1 to 10.\n";
 6
 7 while ($a ne "q"){
 8   print "Enter \$a or (q)uit: ";
 9   chomp($a=<STDIN>);
10   last if $a eq "q" or $a eq "";
11
12     print "Enter \$b: ";
13   chomp($b=<STDIN>);
14
15   print prtroman($a,$b),"\n";
16 }
17
18 sub prtroman{
19   my $a = $_[0];
20   my $b = $_[1];
21   my $c = sumab($a,$b);
22
23     $c = c2roman($c);
24   $a = c2roman($a);
25   $b = c2roman($b);
26
27   return "$a plus $b equals $c";
28 }
29
30 sub c2roman{
31   my $num = $_[0];
32   my @roman;
33   @roman = qw/I II III IV V VI VII VIII IX X/;
34   if ($num > 0 and $num < 11) {  return $roman[$num-1]; }
35   else { return $num ; }
36 }
37
38 sub sumab{
39   my $a = $_[0];
40   my $b = $_[1];
41     return $a + $b;
42 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

4. Create a script that reads in a list of numbers into an array. After entering the numbers, determine the minimum, maximum, and mean (average) of the list. Create the following subroutines:

   a. Find the minimum and the maximum values.

   b. Compute the mean.

   c. Print the results.

   Hint: To find the min and the max, perform a test like this inside the loop that iterates through the array.

```
$min = $_ if $min >= $_ ;
$max = $_ if $max <= $_ ;
```

   Sample output:

```
Enter a list of numbers one per line.
Press Ctrl-D when you are finished
4
15
8
25
Min: 4.00 Max: 25.00 Mean: 13.00
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0804.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 print "Enter a list of numbers one per line.\n";
 4 print "Press ctrl-d when you are finished\n";
 5 my @in = <STDIN> ;
 6 chomp @in;
 7
 8 # Call subroutines.
 9 my $mean = mean(@in) ; # One value returned
10 my($min, $max) = minmax(@in) ; # Two values returned
11 output($mean, $min, $max) ; # No value returned
12
13 sub mean {
14    my ($sum,$mean) ;
15    foreach (@_) {
16      $sum += $_
17    }
18    $mean = $sum / ($#_+1) ;
19    return $mean ; # Return one value.
20 }
21
22 sub minmax {
23    my ($min, $max) ;
24    $min = $max = $_[0] ; # Yes, this works.
25    foreach (@_) {
26       $min = $_ if $min >= $_ ;
27       $max = $_ if $max <= $_ ;
28       }
29    return ($min, $max) ; # Returning two values.
30 }
31
32 sub output {
33    my ($mean,$min,$max) = @_[0,1,2] ;
34    printf "Min: %4.2f Max: %4.2f Mean: %4.2f\n",$min,$max,$mean ;
35 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

5. Using the previous script, move all your subroutines into a library file. Produce the same output, but call the routines from the library file. Make any changes necessary to include the library.

   Suggested solution:

lab0805.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 unshift(@INC,"/home/mw119255/lf/mod08/labs/");
 4 require "Stats.plx";
 5
 6 print "Enter a list of numbers one per line.\n";
 7 print "Press ctrl-d when you are finished\n";
 8 my @in = <STDIN> ;
 9 chomp @in;
10
11 # Call subroutines.
12 my $mean = mean(@in) ; # One value returned
13 my($min, $max) = minmax(@in) ; # Two values returned
14 output($mean, $min, $max) ; # No value returned
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Stats.pl

```
 1 #!/usr/bin/perl -w
 2 # Stats Library
 3
 4 sub mean {
 5    my ($sum,$mean) ;
 6    foreach (@_) {
 7      $sum += $_
 8    }
 9    $mean = $sum / ($#_+1) ;
10    return $mean ; # Return one value.
11 }
12
13 sub minmax {
14    my ($min, $max) ;
15    $min = $max = $_[0] ; # Yes, this works.
16    foreach (@_) {
17       $min = $_ if $min >= $_ ;
18       $max = $_ if $max <= $_ ;
19       }
20    return ($min, $max) ; # Returning two values.
21 }
22
23 sub output {
24    my ($mean,$min,$max) = @_[0,1,2] ;
25    printf "Min: %4.2f Max: %4.2f Mean: %4.2f\n",$min,$max,$mean ;
26 }
27 1;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

6. Convert your library to a package. Make any changes necessary in the main script to call the subroutines in the library file. Produce the same output as before.

   Suggested solutions:

lab0806.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 unshift(@INC,"/home/mw119255/lf/mod08/labs/");
 4 require "Statsp.plx";
 5
 6 print "Enter a list of numbers one per line.\n";
 7 print "Press ctrl-d when you are finished\n";
 8 my @in = <STDIN> ;
 9 chomp @in;
10
11 # Call subroutines.
12 my $mean = Stats::mean(@in) ; # One value returned
13 my($min, $max) = Stats::minmax(@in) ; # Two values returned
14 Stats::output($mean, $min, $max) ; # No value returned
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Statsp.pl

```
 1 #!/usr/bin/perl -w
 2 # Stats Package
 3 package Stats;
 4
 5 sub mean {
 6    my ($sum,$mean) ;
 7    foreach (@_) {
 8      $sum += $_
 9    }
10    $mean = $sum / ($#_+1) ;
11    return $mean ; # Return one value.
12 }
13
14 sub minmax {
15    my ($min, $max) ;
16    $min = $max = $_[0] ; # Yes, this works.
17    foreach (@_) {
18       $min = $_ if $min >= $_ ;
19       $max = $_ if $max <= $_ ;
20       }
21    return ($min, $max) ; # Returning two values.
22 }
23
24 sub output {
25    my ($mean,$min,$max) = @_[0,1,2] ;
26    printf "Min: %4.2f Max: %4.2f Mean: %4.2f\n",$min,$max,$mean ;
27 }
28 1;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

7. Convert the library file into a Perl module. Make any changes necessary in the main script to call the subroutines in the module.

   Suggested solutions:

lab0807.plx

```
 1 #!/usr/bin/perl -w
 2 use strict;
 3 use lib "/home/mw119255/lf/mod08/labs/";
 4 use Stats ;
 5
 6 print "Enter a list of numbers one per line.\n";
 7 print "Press ctrl-d when you are finished\n";
 8 my @in = <STDIN> ;
 9 chomp @in;
10
11 # Call subroutines.
12 my $mean = Stats::mean(@in) ; # One value returned
13 my($min, $max) = Stats::minmax(@in) ; # Two values returned
14 Stats::output($mean, $min, $max) ; # No value returned
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Stats.pm

```
 1 #!/usr/bin/perl -w
 2 # Stats Package
 3 package Stats;
 4
 5 sub mean {
 6    my ($sum,$mean) ;
 7    foreach (@_) {
 8      $sum += $_
 9    }
10    $mean = $sum / ($#_+1) ;
11    return $mean ; # Return one value.
12 }
13
14 sub minmax {
15    my ($min, $max) ;
16    $min = $max = $_[0] ; # Yes, this works.
17    foreach (@_) {
18       $min = $_ if $min >= $_ ;
19       $max = $_ if $max <= $_ ;
20       }
21    return ($min, $max) ; # Returning two values.
22 }
23
24 sub output {
25    my ($mean,$min,$max) = @_[0,1,2] ;
26    printf "Min: %4.2f Max: %4.2f Mean: %4.2f\n",$min,$max,$mean ;
27 }
28 1;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

File and Directory Operations

Objectives

Upon completion of this module, you should be able to:

- Use file operators to determine the characteristics of a file
- Display the contents of a directory using `chdir` and globbing
- Display the contents of a directory using directory handles and `readdir`
- Rename files
- Create symbolic links to files
- Display all symbolic links in a directory
- Set file permissions for files based on their extensions

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Relevance

Discussion – The following question is relevant to understanding file and directory operations:

What type of operations need to be performed on files or directories?

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Introducing File and Directory Operations

Working with files does not consist only of reading data from or writing data to them. What if a file has to be opened for output, but is write protected or does not exist at all? Perl offers a large range of commands for handling file and directory administration that allow you to do the following:

- Change the current directory
- Get the contents of a directory
- Get information about file permissions
- Modify file permissions
- Remove or rename files

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

File and Directory Tests

Up to now, we always terminated our program immediately if an error occurred in opening a file:

```
open FH, "mydata" or die "Open failed: $!\n";
```

To avoid this situation or at least to get more detailed information about what is wrong, it is preferable to check in advance whether the file exists or has the necessary permissions.

For these tasks, Perl offers specific file and directory test operators, similar to those found in UNIX shells.

For example, `-e` tests if a file or directory exists. The `-w` operator tests if it is writable. So, "`-e file`" returns true or false. A simple script to test for the existence of a file follows:

```
e0901.plx
  1 #!/usr/bin/perl -w
  2
  3 $file = "temp.plx";
  4
  5 if (-e $file){
  6   print "File $file exists\n";
  7 }
  8 else {
  9   print "Can't find $file\n";
 10 }

$ e0901.plx
File temp.plx exists
```

Line 5 shows the `-e` operator in action. This expression produces true or false, and prints the appropriate message.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table 9-1 contains a list of the general file and directory operators.

Table 9-1 General File and Directory Operators

Operator	Meaning
`-A`	The access age in days
`-b`	The entry is a block-special file
`-B`	The file is binary
`-c`	The entry is a character-special file
`-C`	The `inode` modification age in days
`-d`	The entry is a directory
`-e`	The file or directory exists
`-f`	The entry is a plain file
`-g`	The file or directory is `setgid`
`-k`	The file or directory has Sticky Bit set
`-l`	The entry is a symbolic link
`-M`	The modification age in days
`-p`	The entry is a named pipe
`-s`	The file or directory exists and has a non-zero size
`-S`	The entry is a socket
`-T`	The entry is text
`-u`	The file is `setuid`
`-z`	The file exists and has zero size

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

A basic directory tester might be written as follows:

e0902a.plx
```
  1 #!/usr/bin/perl -w
  2
  3 $file = "..";
  4
  5 if (-d $file){
  6
  7   print "Directory $file exists\n";
  8
  9   $age = -M $file;
 10
 11   print "The directory was last modified $age days ago\n";
 12 }
```

```
$ e0902a.plx
Directory .. exists
The directory was last modified 69.8609837962963 days ago
```

On Line 5, `-d` tests for existence implicitly. If the file exists and it is a directory, a message prints and the number of days since the directory was last modified prints.

When a test is performed, Perl uses the `stat(3)` system call to store the complete list of information from the `inode`. If multiple tests are needed for a particular file, then `stat` is called each time, unless _ is used. Using _ in a script tells Perl to use the `inode` information from the last call. This both simplifies the code and provides a performance benefit by avoiding repeated system calls to `stat`.

e0902b.plx
```
  1 #!/usr/bin/perl -w
  2
  3 $file = "temp.plx";
  4
  5 -e $file or die "File doesn't exist.\n";
  6 -w _ or die "File isn't writable.\n";
  7
  8 print "File is exists and is writeable.\n";
```

```
$ e0902b.plx
File is exists and is writeable.
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Permissions

Table 9-2 contains the most commonly used file and directory permission test operators.

Table 9-2 File and Directory Permission Operators

Operator	Meaning
`-o`	The file or directory is owned by the effevtive user
`-O`	The file or directory is owned by the real user
`-r`	The file or directory is readable
`-R`	The file or directory is readable by the real user
`-w`	The file or directory is writable
`-W`	The file or directory is writable by the real user
`-x`	The file or directory is executable
`-X`	The file or directory is executable by the real user

File test operators make it very easy to test for effective permissions. The script that follows shows how the `-r`, `-w`, `-x`, and `-o` operators can test a file for its permissions.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

e0903.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $file = $ARGV[0];
 4
 5 print "You have the following permissions to $file:\n";
 6
 7 if (-e $file){
 8   if (-r _ ){print "read ";}
 9   if (-w _ ){print "write ";}
10   if (-x _ ){print "execute ";}
11   if (-o _ ){print "\nYou also own this file";}
12 }
13
14 print "\n";
```

```
$ e0903.plx temp.pl
You have the following permissions to temp.pl:
read write execute
You also own this file
```

Lines 7 through 11 show how the operators test the various characteristics of a file.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

However, some operators are not limited to true/false return values. For example with the `-s` operator, the size of a file is returned.

```
e0904a.plx
  1 #!/usr/bin/perl -w
  2
  3 $file = $ARGV[0];
  4
  5 print "You have the following permissions to $file:\n";
  6
  7 if (-e $file){
  8   if (-r _ ){print "read ";}
  9   if (-w _ ){print "write ";}
 10   if (-x _ ){print "execute ";}
 11
 12   print "\nThe size of the file is ",(-s _ )," bytes";
 13
 14   if (-o _ ){print "\nYou also own this file";}
 15 }
 16
 17 print "\n";
```

```
$ e0904a.plx temp.pl
You have the following permissions to temp.pl:
read write execute
The size of the file is 58 bytes
You also own this file
```

On Line 12, the result of the `-s` test is returned to the `print` command. The file size is returned, as shown in the output.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Sometimes we want the results of several tests before we proceed. The tests that return a boolean and can be readily connected with "and". For example: (-r $file and -w _) joins two operators. Perl 5.10 allows the tests to be stacked: (-r -e $file).

Consider the following examples of collection of tests:

```
e0904b.plx
1 #!/usr/bin/perl -w
2
3 $file = "temp.plx";
4 if (-r $file and -w _){
5   print "The file $file is readable and writeable.\n";}
6 #or
7 use 5.010;
8 if (-w -r $file){
9   say ""The file $file is readable and writeable.\n";}

$ e0904b.plx
  The file temp.plx is readable and writeable.
  The file temp.plx is readable and writeable.
```

The two tests are equivalent. Notice that the tests in line 8 appear to be in a different order. However, since the test nearest to the file name is done first, they are equivalent.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The stat Function

The stat command retrieves a complete list of inode contents, such as UID, GID, or PERM. A complete reference is found in the man page perlfunc(1). An example script using the stat function follows.

e0904c.plx

```
  1 #!/usr/bin/perl -w
  2
  3 $file = $ARGV[0];
  4
  5 ( $dev,$ino,$mode,$nlink,$uid,
  6         $gid,$rdev,$size,$atime,$mtime,
  7         $ctime,$blksize,$blocks ) = stat $file or die
"Unable to stat $file: $!\n";
  8
  9 print "Stat Results\n";
 10 print "dev: $dev\n";
 11 print "inode: $ino\n";
 12 print "mode: $mode\n";
 13 print "# of links: $nlink\n";
 14 print "uid: $uid\n";
 15 print "gid: $gid\n";
 16 print "rdev: $rdev\n";
 17 print "size: $size\n";
 18 print "last access time: $atime\n";
 19 print "last modify time: $mtime\n";
 20 print "inode change time: $ctime\n";
 21 print "block size: $blksize\n";
 22 print "blocks: $blocks\n";
```

```
$ e0904c.plx temp.pl
Stat Results
dev: 56623424
inode: 23427842
mode: 33261
# of links: 1
uid: 120255
gid: 10
rdev: 0
size: 55
last access time: 985296232
last modify time: 983383192
inode change time: 983931103
block size: 8192
blocks: 2
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Reading Directory Contents

Perl provides a number of tools for searching for specific files or a whole range of files. These tools allow you to manipulate directory contents or even entire directory subtrees.

Using UNIX Commands

One easy method used in previous examples is reading the output of the `ls` command using backquotes.

```
@dir = `ls` ;
@dir = `ls -l` ;
@dir = qx/ls -l *.txt/ ;
```

In addition, to iterate a complete directory structure recursively, you can use the `find` command. The `find` command selects files by criteria, such as owner, size, permissions, or modification time.

```
e0905a.plx
  1 #!/usr/bin/perl -w
  2
  3 $dir = $ARGV[0];
  4 $pattern = $ARGV[1];
  5
  6 @dirstruct = `find $dir -name $pattern -print`;
  7
  8 print @dirstruct;

$ e0905a.plx ../.. *.txt
../../mod03/labs/crapslog.txt
../../mod07/labs/machine1.txt
../../mod07/labs/machine2.txt
../../mod07/examples/out.txt
../../mod07/examples/fw.txt
```

This example shows how `find` located a number of files in different directories. The techniques used in the script have been described previously.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using File-Name Globbing

File-name globbing is a method Perl uses for getting directory contents. With globbing, a file-name pattern is specified in angle brackets. The pattern can contain shell wildcards, such as * or ?. When executed, Perl opens a shell in the background and assigns the pattern to it. The shell then resolves the pattern and returns the list of matching file names to Perl.

```
e0905b.plx
  1 #!/usr/bin/perl -w
  2
  3 for (sort <./*>){
  4   print "$_ \n";
  5 }

$ e0905b.plx
./e0901.plx
./e0902.plx
./e0902a.plx
./e0903.plx
./e0904.plx
./e0904a.plx
./e0905.plx
./e0905b.plx
./e0906.plx
./e0907.plx
./e0908.plx
./temp.plx
./test1
./test1.mirror
```

This script merely reads the contents of the current directory using globbing and prints out the contents.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The following example lists all `.plx` files in a given directory name that is passed from the command line.

e0906.plx

```
  1 #!/usr/bin/perl -w
  2
  3 $dir = $ARGV[0];
  4
  5 for (sort <$dir/*plx>){
  6   print "$_ \n";
  7 }
```

```
$ e0906.plx .
./e0901.plx
./e0902a.plx
./e0902b.plx
./e0903.plx
./e0904a.plx
./e0904b.plx
./e0905a.plx
./e0905b.plx
./e0906.plx
./e0907.plx
./e0908.plx
```

Notice line 5. The glob is the pattern that appears between the <>. The example shows that when a `.` is passed to the script, the search is essentially performed on `<./*plx>`. The glob returns a list of file names. The `foreach` loop prints each file name as shown in the output.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using Directory Handles

In addition to globs, Perl provides a number of directory functions. These functions use a directory handle to manage directory data. A directory handle functions just like a filehandle. However, a directory handle is not a filehandle and has its own namespace. Table 9-3 contains a list of available directory functions.

Table 9-3 Directory Functions

Function	**Meaning**
`opendir DH, "dirname";`	Opens a directory handle for the directory `dirname`
`readdir DH;`	Reads one entry from the directory handle `DH`
`telldir DH;`	States the position of the pointer to `DH`
`seekdir DH, pos;`	Sets the pointer to `DH` to the specified position
`rewinddir DH;`	Sets the pointer to `DH` to the beginning
`closedir DH;`	Closes `DH`

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Opening and reading a directory is very similar to opening and reading a file. The example that follows modifies the previous script and uses directory handles and the `readdir` function to perform the same task.

e0907.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 opendir DH, $dir or die "Open failed: $!\n";
 6
 7 for (sort readdir(DH)){
 8
 9      if (/\.plx/){
10     print "$_\n";
11   }
12
13 }
14
15 closedir(DH);
```

```
$ e0907.plx .
e0901.plx
e0902a.plx
e0902b.plx
e0903.plx
e0904a.plx
e0904b.plx
e0905a.plx
e0905b.plx
e0906.plx
e0907.plx
e0908.plx
temp.pl
```

Notice that the search pattern cannot be specified with the directory commands because they only take directories as arguments. To search for the desired files, the regular expression on Line 9 limits the files that are printed out.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

File and Directory Functions

This section describes the file and directory functions built into Perl. These routines perform functions in Perl that are often similar to command-line utilities found in a UNIX shell.

Table 9-4 describes the file and directory functions.

Table 9-4 File and Directory Functions

Function	Meaning
`unlink filename`	Deletes a file
`rename oldfile, newfile`	Renames a file or directory
`link oldfile, newfile`	Creates a hard link
`symlink oldfile, newfile`	Creates a symbolic link
`readlink linkname`	Returns the path and file pointed to by the link
`mkdir dirname, perms`	Creates a new directory
`rmdir dirname`	Deletes an empty directory
`chdir newdir`	Changes the current directory
`chown uid, gid, filename`	Changes `uid` and `gid` (numeric IDs, -1 to keep ID)
`chmod perms, filename`	Changes the mode (four digits are obligatory; for example, 0644)
`utime atime, mtime, filename`	Changes access and modification times

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The script that follows demonstrates a number of the functions listed in the previous table. After creating the file, it sets the file's permissions and creates a symbolic link to the file.

```
e0908.plx
  1 #!/usr/bin/perl -w
  2
  3 $newfile = $ARGV[0];
  4
  5 # Next two lines create an empty file
  6 open FH, ">$newfile" or die "Open failed: $!\n";
  7 close FH;
  8
  9 chmod 0644, $newfile; # Set permissions to 644
 10
 11 symlink $newfile, "$newfile.mirror"; # Create link

$ e0908.plx test2
$ ls -l test2*
-rw-r--r--   1 x55 staff          0 May  9 09:50 test2
lrwxrwxrwx   1 x55 staff          5 May  9 09:50 test2.mirror -> test2
```

Lines 9 and 11 demonstrate some of the file and directory functions. The `chmod` function works much like the UNIX command. The file is set to `rw- r-- r--`. The `symlink` function creates a link, `test2.mirror`, that points to the file originally created with the script.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise: Create File and Directory Scripts

In this exercise, you complete the following tasks:

- Use file operators to determine the characteristics of a file
- Display the contents of a directory using `chdir` and globbing
- Display the contents of a directory using directory handles and `readdir`
- Rename files
- Create symbolic links to files
- Display all symbolic links in a directory
- Set file permissions for files based on their extensions

Tasks

Complete the following steps:

1. Create a script that reads a list of files from the command line. If the file exists, print the permissions that the user has to that file and if the user owns that file. Otherwise, print a message stating that the file does not exist.

   Sample output:

```
$ lab0901.plx lab0902.plx temp.pl badfile.pl

Permission Tester

Permissions and rights to lab0902.plx: read write execute
owner
Permissions and rights to temp.pl: read write execute owner
File badfile.pl does not exist
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

2. Pass a list of files to a script on the command line, find out which file is the oldest, and print out that file's name and age in days.

   Sample output:

```
$ lab0902.plx lab0901.plx temp.pl

The oldest file is:

temp.plx is 118.093333333333 days old
```

3. Create a script that performs the same function as the `ls` command using `chdir` and globbing. Print an error message if the directory specified on the command line does not exist.

   Sample output:

```
$ lab0903.plx ../examples
e0901.plx
e0902a.plx
e0902b.plx
e0903.plx
e0904a.plx
e0904b.plx
e0905a.plx
e0905b.plx
e0906.plx
e0907.plx
e0908.plx
temp.pl
test2
test2.mirror
testfile
testfile.mirror
```

4. Create the same script as you did previously, but use directory handles and `readdir` instead.

5. Try modifying the original script so that it shows files that begin with "." as well.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

6. Create a simple Perl version of the UNIX `mv` command. Pass the script the old file name and new file name on the command line.

   Sample output:

```
$ lab0906.plx lab0805.plx lab0905.plx
Renamed lab0805.plx to lab0905.plx

$ lab0906.plx lab0805.plx lab0905.plx
Can't rename lab0805.plx: No such file or directory
```

7. Create a simple Perl version of the UNIX `ln -s` command to create symbolic links.

   Sample output:

```
$ lab0907.plx temp.pl test6
Created link test6

$ lab0907.plx temp.pl test6
Can't create link: File exists
```

8. Write a script that displays all the symbolic links in a directory and their destination files, much like `ls -l`.

   Hint: Use the `-l` file test to determine if the file is a link.

   Sample output:

```
$ lab0908.plx

test3 -> temp.pl
test4 -> temp.pl
test5 -> temp.pl
There are 3 files with symbolic links
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

9. Create a script that, given a directory on the command line, sets the permissions for all `.plx` files to 755. An error message should appear if there is a problem changing the permissions.

   Sample output:

```
$ lab0909.plx ../examples

e0901.plx changed
e0902a.plx changed
e0902b.plx changed
e0903.plx changed
e0904a.plx changed
e0904b.plx changed
e0905a.plx changed
e0905b.plx changed
e0906.plx changed
e0907.plx changed
e0908.plx changed
```

10. Build a script that produces the following statistical ouput for a specified directory.

    Read in the directory contents using file globs and the command-line argument. Within a `foreach` loop, build your statistics. Count the following:

    - The number of files
    - The number of directories
    - The number of links
    - The number of files with a special bit set (`-u`, `-g`, `-k`)
    - The number of files less than 1 Kbyte
    - The number of file greater than 1 Kbyte

    Sample output:

```
$ lab0910.plx ~

Files:      16
Directory:  24
Links:      5
Files with special bits set: 0
Files < 1K:  11
Files > 1K:  5
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

11. Sort the contents of a directory, specified on the command line, by size. Read in the directory contents into an array using file globs. Sort the resulting array with `sort`, specifying the sort statement in braces. Use the numeric comparison operator <=>. For simplicity, use `printf` to print the file names and sizes.

Sample output:

```
$ lab0911.plx .
                File       Size
       ./lab0910.plx        503
       ./lab0901.plx        362
       ./lab0902.plx        359
       ./lab0911.plx        310
       ./lab0909.plx        254
          ./test.plx        250
             ./test2        250
       ./lab0908.plx        248
       ./lab0905.plx        177
       ./lab0907.plx        175
       ./lab0903.plx        174
       ./lab0904.plx        172
       ./lab0906.plx        142
           ./temp.pl         55
             ./test3         55
             ./test4         55
             ./test5         55
             ./test6         55
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Summary

Discussion – Take a few minutes to discuss the experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Solutions

1. Create a script that reads a list of files from the command line. If the file exists, print the permissions that the user has to that file and if the user owns that file. Otherwise, print a message stating that the file does not exist.

   Sample output:

```
$ lab0901.plx lab0902.plx temp.pl badfile.pl

Permission Tester

Permissions and rights to lab0902.plx: read write execute
owner
Permissions and rights to temp.pl: read write execute owner
File badfile.pl does not exist
```

   Suggested solution:

```
lab0901.plx
  1 #!/usr/bin/perl -w
  2
  3 print "\nPermission Tester\n\n";
  4
  5 foreach $file (@ARGV){
  6   if (-e $file){
  7     print "Permissions and rights to $file: ";
  8     if (-R _ ) {print "read ";}
  9     if (-W _ ) {print "write ";}
 10     if (-x _ ) {print "execute ";}
 11     if (-o _ ) {print "owner ";}
 12     print "\n";
 13   }
 14   else {
 15     print "File $file does not exist\n";
 16   }
 17 }
 18 print "\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

2. Pass a list of files to a script on the command line, find out which file is the oldest, and print out that file's name and age in days.

   Sample output:

```
$ lab0902.plx lab0901.plx temp.pl

The oldest file is:

temp.plx is 118.093333333333 days old
```

   Suggested solution:

```
lab0902.plx
 1 #!/usr/bin/perl -w
 2 $age = 0; $oldest = 0;
 3 $oldestfile = "";
 4
 5
 6 print "\nThe oldest file is:\n\n";
 7
 8 foreach $file (@ARGV){
 9
10   if (-e $file){
11     $age = -M $file;
12
13     if ($age > $oldest){
14       $oldest = $age;
15       $oldestfile = $file;
16     }
17   }
18   else {
19     print "File $file does not exist\n";
20   }
21
22 }
23
24 print "$oldestfile is $oldest days old\n";
25 print "\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

3. Create a script that performs the same function as the `ls` command using `chdir` and globbing. Print an error message if the directory specified on the command line does not exist.

   Sample output:

```
$ lab0903.plx ../examples
e0901.plx
e0902a.plx
e0902b.plx
e0903.plx
e0904a.plx
e0904b.plx
e0905a.plx
e0905b.plx
e0906.plx
e0907.plx
e0908.plx
temp.pl
test2
test2.mirror
testfile
testfile.mirror
```

   suggested Solution:

```
lab0903.plx
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 chdir $dir or die "Cannot cd to $dir: $!\n";
 6
 7 #Walk through glob results
 8 foreach $file (sort <*>){
 9   print $file,"\n";
10 }
11
12 print "\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

4. Create the same script as you did previously, but use directory handles and `readdir` instead.

   Suggested solution:

lab0904.plx
```
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 opendir DIR,$dir or die "Cannot open $dir: $!\n";
 6
 7 foreach $file (sort readdir(DIR)){
 8   print "$file\n";
 9 }
10
11 print "\n";
12 close(DIR);
```

5. Try modifying the original script so that it shows files that begin with "." as well.

   Suggested solution:

lab0905.plx
```
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 chdir($dir) || die("Cannot cd to $dir : $!\n");
 6
 7 foreach $file (sort <\.* *>){ # .* and * gets all files
 8   print "$file\n";
 9 }
10
11 print "\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

6. Create a simple Perl version of the UNIX `mv` command. Pass the script the old file name and new file name on the command line.

   Sample output:

```
$ lab0906.plx lab0805.plx lab0905.plx
Renamed lab0805.plx to lab0905.plx

$ lab0906.plx lab0805.plx lab0905.plx
Can't rename lab0805.plx: No such file or directory
```

   Suggested solution:

```
lab0906.plx
  1 #!/usr/bin/perl -w
  2
  3 $old = $ARGV[0];
  4 $new = $ARGV[1];
  5
  6 rename $old, $new or die "Can't rename $old: $!\n";
  7
  8 print "Renamed $old to $new\n";
```

7. Create a simple Perl version of the UNIX `ln -s` command to create symbolic links.

   Sample output:

```
$ lab0907.plx temp.pl test6
Created link test6

$ lab0907.plx temp.pl test6
Can't create link: File exists
```

   Suggested solution:

```
lab0907.plx
  1 #!/usr/bin/perl -w
  2
  3 $source = $ARGV[0];
  4 $target = $ARGV[1];
  5
  6 # Create link or die
  7 symlink $source, $target or die "Can't create link: $!\n";
  8 print "Created link $target\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

8. Write a script that displays all the symbolic links in a directory and their destination files, much like `ls -l`.

   Hint: Use the `-l` file test to determine if the file is a link.

   Sample output:

```
$ lab0908.plx

test3 -> temp.pl
test4 -> temp.pl
test5 -> temp.pl
There are 3 files with symbolic links
```

   Suggested solution:

lab0908.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $count = 0;
 4
 5 print "\n";
 6
 7 # Change .plx files
 8 foreach $file (<*>){
 9   if (-l $file){
10
11     $target=readlink $file;
12     print "$file -> $target\n";
13
14     $count++
15   }
16 }
17
18 print "There are $count files with symbolic links\n";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

9. Create a script that, given a directory on the command line, sets the permissions for all `.plx` files to 755. An error message should appear if there is a problem changing the permissions.

   Sample output:

```
$ lab0909.plx ../examples

e0901.plx changed
e0902a.plx changed
e0902b.plx changed
e0903.plx changed
e0904a.plx changed
e0904b.plx changed
e0905a.plx changed
e0905b.plx changed
e0906.plx changed
e0907.plx changed
e0908.plx changed
```

   Suggested solution:

lab0909.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 chdir $dir or die "Cannot cd to $dir $!\n";
 6
 7 print "\n";
 8
 9 # Change .plx files
10 foreach $file (<*.plx>){
11   if (chmod 0755, $file){
12     print "$file changed\n";
13   }
14   else{
15     warn "Couldn't chmod $file: $!\n";
16   }
17 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

10. Build a script that produces the following statistical ouput for a specified directory.

    Read in the directory contents using file globs and the command line argument. Within a `foreach` loop, build your statistics. Count the following:

    - The number of files
    - The number of directories
    - The number of links
    - The number of files with a special bit set (`-u`, `-g`, `-k`)
    - The number of files less than 1 Kbyte
    - The number of file greater than 1 Kbyte

    Sample output:

```
$ lab0910.plx ~

Files:      16
Directory:  24
Links:      5
Files with special bits set: 0
Files < 1K:  11
Files > 1K:  5
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0910.plx

```
#!/usr/bin/perl -w
# init
$file = $dir = $link = $xuid = $k1 = $k2 = 0;

$dir = $ARGV[0];

# Remove trailing / if present
$dir =~ s=/$==;

for (<$dir/*>) {

   $file++ if -f;
   $dir++  if -d;
   $link++ if -l;
   $xuid++ if -u or -g or -k;
   $k1++  if -f and -s $_ <= 1000;
   $k2++  if -f and -s $_ > 1000;

}

print <<end_of_text;

Files:      $file
Directory:  $dir
Links:      $link
Files with special bits set: $xuid
Files < 1k:  $k1
Files > 1k:  $k2
end_of_text

```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

11. Sort the contents of a directory, specified on the command line, by size. Read in the directory contents into an array using file globs. Sort the resulting array with `sort`, specifying the sort statement in braces. Use the numeric comparison operator <=>. For simplicity, use `printf` to print the file names and sizes.

    Sample output:

```
$ lab0911.plx .
                File       Size
       ./lab0910.plx        503
       ./lab0901.plx        362
       ./lab0902.plx        359
       ./lab0911.plx        310
       ./lab0909.plx        254
          ./test.plx        250
             ./test2        250
       ./lab0908.plx        248
       ./lab0905.plx        177
       ./lab0907.plx        175
       ./lab0903.plx        174
       ./lab0904.plx        172
       ./lab0906.plx        142
            ./temp.pl         55
             ./test3         55
             ./test4         55
             ./test5         55
             ./test6         55
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Suggested solution:

lab0911.plx

```
 1 #!/usr/bin/perl -w
 2
 3 $dir = $ARGV[0];
 4
 5 # Remove trailing slash if present
 6 $dir =~ s=/$==;
 7
 8 @dir = <$dir/*>; # Read directory glob
 9
10 # Sorting by size.
11 @sdir = sort { -s $b <=> -s $a } @dir ;
12
13 printf "%20s %10s\n", "File", "Size";
14
15 foreach (@sdir) {
16   printf "%20s %10s\n", $_, -s $_;
17 }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

Overview of CGI Programming

Objectives

Upon completion of this module, you should be able to:

- Send a Hypertext Markup Language (HTML) page to a browser using a Common Gateway Interface (CGI) script
- Use a Here document in a CGI script to send an HTML page to a browser
- Read an HTML form using the `GET` or `POST` method

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Relevance

Discussion – The following questions are relevant to CGI scripts:

- What do CGI scripts add to a Web site?
- What languages and technologies are available for Web scripting? How do they differ from Perl?

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Introducing CGI Programming

Perl plays an important role in the World Wide Web. In many cases, Perl is the engine for dynamic Web applications that perform such tasks as search databases, compose orders, or calculate prices.

This module provides an overview of the communication between a Web client and server and how a Hypertext Markup Language (HTML) page is returned by a program. In addition, HTML form processing is described using the `CGI.pm` module included with Perl 5. This includes some coverage of Hypertext Transfer Protocol (HTTP), which is the protocol that communicates between a Web client and a Web server.

Usually, HTML forms collect data from the client and send it to the server. The Common Gateway Interface (CGI) is a standard that defines how data is sent from a Web server to an invoked program and back to the server. From the view of the program, CGI defines how data is input and output.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Client-Server Communication

This section describes the client-server nature of the Web.

Static HTML Pages

When a static HTML page is requested, the client (the browser) sends the request, using the HTTP protocol, to the Web server on the destination host. The server loads the HTML page from its hard disk and sends it back to the client using HTTP. Figure 10-1 illustrates this transaction.

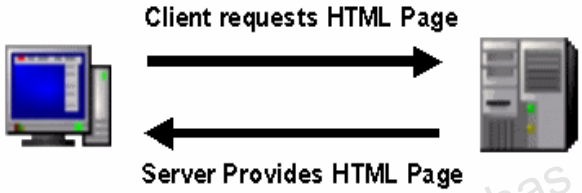


Figure 10-1 HTTP Transaction

Web Programs

For dynamic Web pages, the client sends a request to a program on the server. The program may be a Perl script, a shell script, or a C program. The invoked program is always executed on the server. The program returns a complete HTML page to the server, which, in turn, sends the page back to the client, as shown in Figure 10-2.

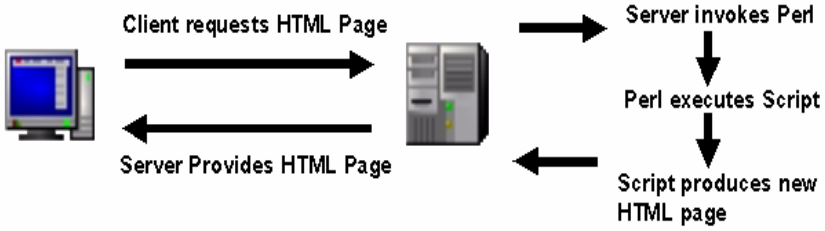


Figure 10-2 Calling a CGI Script

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Responding to a Request

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

Remember, a server always returns an HTML page to the browser. So the starting point for any CGI programmer is to create a program that returns an HTML page to the server. Before sending an HTML page to browser, a CGI program must include an HTTP header before the HTML page.

The header consists of a Multipurpose Internet Mail Extensions-type (MIME-type) definition and a blank line. For example:

```
Content-type: text/html
```

The previous line functions as a minimal HTTP header. When the page is passed to the Web browser, this line tells the browser what type of data to expect. In this example, a text-based HTML page is specified. HTTP requires a blank line between the HTTP header and the HTML page. The rest of the output is a regular HTML page. The following is a sample of what sort of data is returned to the Web server.

```
Content-type: text/html

<HTML>
<HEAD> <TITLE>Database search results</TITLE>  </HEAD>
<BODY><H2> Search Results</H2>
    <H4>
    Randal Schwartz: Learning Perl
    <BR>
    Larry Wall: Programming Perl
    </H4>
    <H5>Wed Aug 02 12:56:03 2000</H5>
</BODY>
</HTML>
```

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

How is this data created in Perl? There are two basic methods for returning CGI data.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Using `print` Statements

The most straightforward method to render the previous HTML page is to use `print` statements. For example:

e1001.cgi

```
 1 #!/usr/bin/perl -w
 2
 3 print "Content-type: text/html\n\n";
 4 print "<HTML>\n";
 5 print "<HEAD> <TITLE>Datbase search results</TITLE></HEAD>\n";
 6 print "<BODY>\n";
 7 print "<H2> Search Results</H2>\n";
 8 print "<P><H4>\n";
 9 print "Randal Schwartz: Learning Perl<BR>\n";
10 print "Larry Wall: Programming Perl</H4>\n";
11 print "<H5>Wed Aug 02 12:56:03 2000</H5>\n";
12 print "</BODY>\n";
13 print "</HTML>\n";
```

Each line of the HTML document is converted into a `print` statement. Line 3 prints the HTTP header, and a blank line is created by explicitly adding two `\n`.

This method works fine, but typing all those `print` statements can get tiresome. In addition, making changes can be quite difficult.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Here Documents

Here documents (which are described in “Here Documents” on page 3-19) allow you print a block of text inside a Perl script. With this feature, the entire HTML document is embedded inside the script. For example:

e1002.cgi

```
 1 #!/usr/bin/perl -w
 2
 3 print <<End_of_HTML;
 4 Content-type: text/html\n\n
 5 <HTML>
 6 <HEAD> <TITLE>Database search results</TITLE> </HEAD>
 7 <BODY>
 8 <H2>Search Results</H2>
 9 <H4>Randal Schwartz: Learning Perl
10 <BR>
11 Larry Wall: Programming Perl
12 </H4>
13 <H5>Wed Aug 02 12:56:03 2000</H5>
14 </BODY>
15 </HTML>
16 End_of_HTML
17
```

This method better separates the display of the HTML code from the program logic. It is also much easier to type than the multiple `print` method.

Testing CGI

With the previous two scripts created, it is now time to test them out. To do this, copy the files to the `cgi-bin` directory of your Web server. Make sure the files are marked executable. Then, to execute the script, enter the following Universal Resource Locators (URLs) into your browser.

```
http://localhost/cgi-bin/e1001.cgi
http://localhost/cgi-bin/e1002.cgi
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

If the Web server is configured correctly and the scripts are written correctly, you should see something like Figure 10-3.

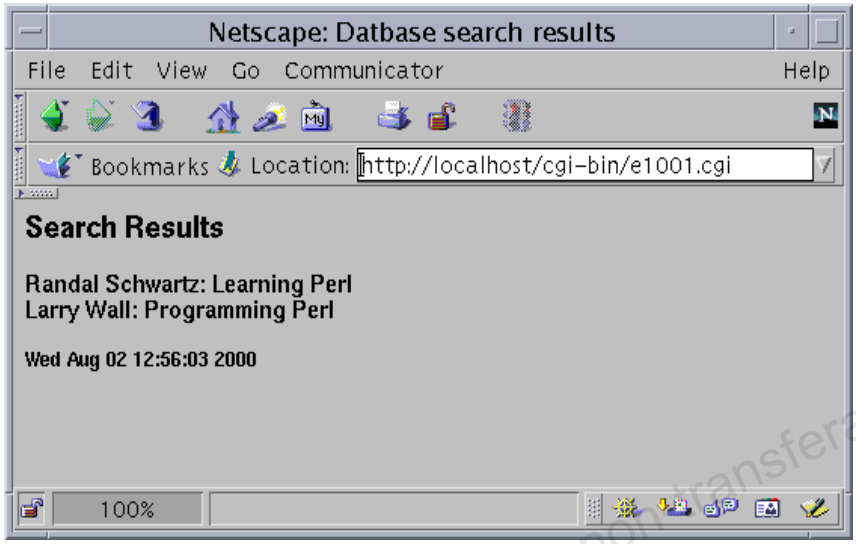


Figure 10-3 CGI Script Output

The output of `e1002.cgi` is the same as that shown in Figure 10-3.

Troubleshooting Tips

The following are items you need to check if your script does not execute:

- Check your Web server's error log. This should give an indication of the problem.
- Make sure that the script is executable.
- Make sure that the path to Perl is correct.
- Execute the script from a shell prompt. If it will not run from a command prompt, it will not run on a Web server.
- Make sure the output of your script on the command line looks just like the HTTP header plus the HTML page shown previously.

Usually, most problems can be resolved by checking the preceding items.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

HTML Forms

Now that you know how to pass HTML data to the server, it is time to examine how to get data from the client. Data is passed from the browser to the server using HTML forms. A basic HTML form looks something like Figure 10-4.

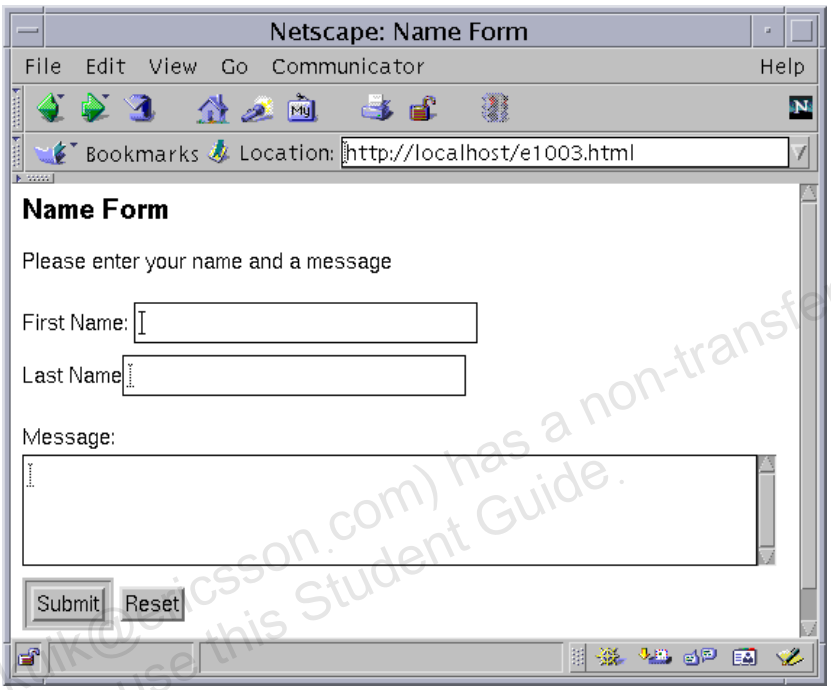


Figure 10-4 HTML Form

Several text fields are used for obtaining the first and last names. A text area gets a short message. A Submit button allows the form to be submitted to the CGI script. The Reset button returns the form to an empty state.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The HTML for this form is as follows:

e1003.html

```
 1 <html>
 2 <head>
 3 <title>Name Form</title>
 4 </head>
 5 <body bgcolor="FFFFFF">
 6 <h2>Name Form</h2>
 7 <p>Please enter your name and a message</p>
 8 <form action="/cgi-bin/e1003.cgi" method=get>
 9 <p>First Name: <input type="text" name="txtFirst" size="25" maxlength="24"><br>
10 Last Name<input type="text" name="txtLast" size="25" maxlength="25"></p>
11 Message:<br>
12 <textarea name="txtMessage" rows="4" cols="55" wrap="virtual"></textarea><br>
13 <input type="Submit" name="Submit" value="Submit">
14 <input type="Reset" name="Reset" value="Reset">
15 </form>
16 </body>
17 </html>
```

Line 8 defines the form and the CGI script that processes the data submitted in this page. It also specifies the HTTP method for submission (more on that follows). Line 9 includes the HTML tag for the text box. What is important to us is the `name` attribute, `txtFirst`. Whatever is specified here is what gets passed to the CGI script. Lines 10 and 12 also specify fields that are passed to the CGI script, `txtLast` and `txtMessage`. Line 13 creates a special Submit button that, when clicked, sends the data to the server and the CGI script.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The form is ready, so a script is needed. The script that follows uses a Here document along with `CGI.pm` to display the data passed from the form. The `CGI.pm` file is a Perl module included with the Perl distribution. The module provides easy-to-use subroutines for processing HTML forms.

e1003.cgi

```
 1 #!/usr/bin/perl -w
 2 use CGI(":standard");
 3
 4 # Get form data
 5 $first = param("txtFirst");
 6 $last = param("txtLast");
 7 $message = param("txtMessage");
 8
 9 print <<End_of_HTML;
10 Content-type: text/html\n\n
11 <html>
12 <head><title>Name Form Results</title></head>
13 <body bgcolor="FFFFFF">
14 <h2>Name Form Results</h2>
15 <p>
16 <b>First Name: </b>$first<br>
17 <b>Last Name: </b>$last<br></p>
18 <p><b>Message:</b><br>
19 $message</p>
20 </body>
21 </html>
22 End_of_HTML
```

On Line 2, the standard routines are requested from the `CGI.pm` module. The `:standard` option includes the forms-processing routines along with other features for displaying HTML programmatically. Lines 5–7 perform primary tasks. The `param` routine takes the HTML field name as a parameter and returns the value from the form. Each of the fields from the form gets the value passed from the form. On Lines 16–19, the variables created display the values passed from the form.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@Ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

With the form ready and the script ready, fill out the form and submit it to the CGI script, as shown in Figure 10-5.

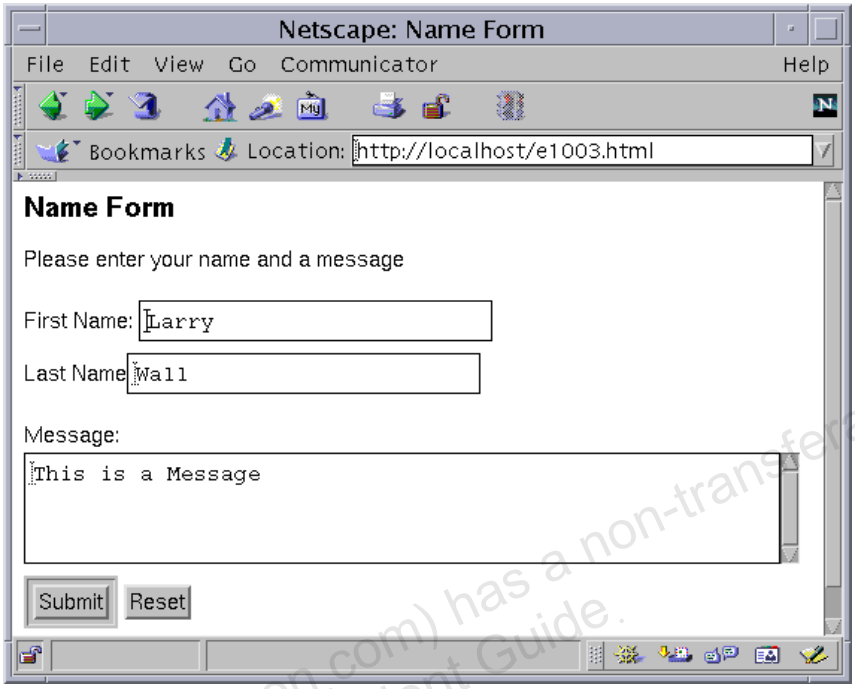


Figure 10-5 HTML Form Data

After submitting the form, the script returns the HTML page shown in Figure 10-6 to the browser.

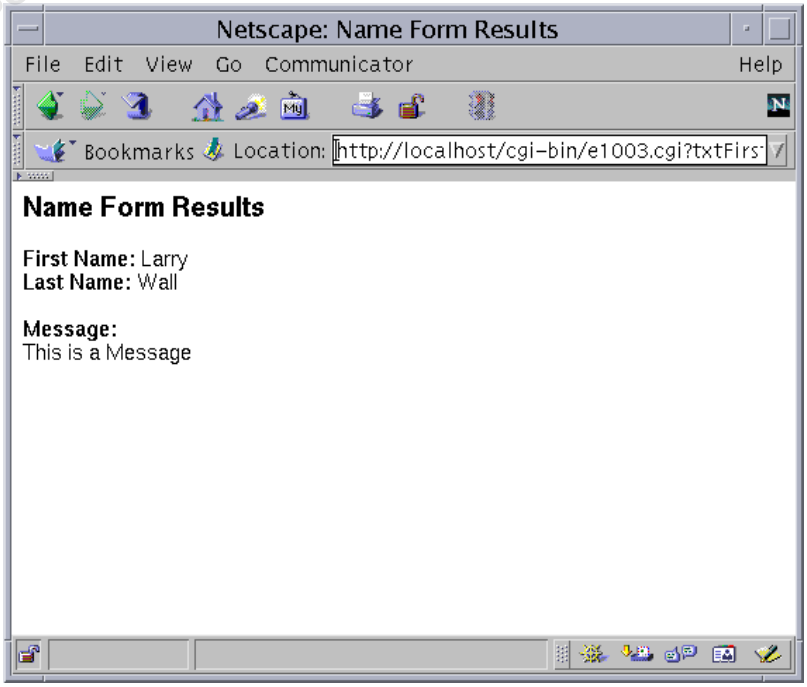


Figure 10-6 CGI Script Results

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The GET Method

Notice the URL in Figure 10-6 on page 10-12. Part of the URL is obscured in the picture. Here is the full URL:

```
http://localhost/cgi-
bin/e1003.cgi?txtFirst=Larry&txtLast=Wall&txtMessage=This+is
+a+Message&Submit=Submit
```

If you look at this URL closely, you see the data and field names created in the HTML form. When the GET HTTP method submits a form, all form data is passed as part of the URL. The ? denotes that form data follows and the data that follows ? is known as the query string. Each field and its value is turned into a name/value pair separated by an =. Each pair is separated by &. The other feature to note is that no spaces are allowed in a query string. Any space or special character is converted into an alternative representation as specified in the HTTP standard.

There are some drawbacks to this method of form submission. First, all the data is right there in the URL for all to see. The data can be encrypted, but this makes for an even longer and uglier query string. In addition, the query string is passed to the script using an environment variable, QUERY_STRING, which is limited to the maximum size of an environment variable. Thus, if form data exceeds this, it is truncated.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The POST Method

The POST method sends form data as part of the HTTP message and has no size limitations. In many cases, it is the preferred way of sending information to a CGI script. To make this script use POST, you only need to make one change.

e1004.html

```
  1 <html>
  2 <head>
  3 <title>Name Form</title>
  4 </head>
  5 <body bgcolor="FFFFFF">
  6 <h2>Name Form</h2>
  7 <p>Please enter your name and a message</p>
  8 <form action="/cgi-bin/e1003.cgi" method=post>
  9 <p>First Name: <input type="text" name="txtFirst" size="25"
maxlength="24"><br>
 10 Last Name<input type="text" name="txtLast" size="25"
maxlength="25"></p>
 11 Message:<br>
 12 <textarea name="txtMessage" rows="4" cols="55"
wrap="virtual"></textarea><br>
 13 <input type="Submit" name="Submit" value="Submit">
 14 <input type="Reset" name="Reset" value="Reset">
 15 </form>
 16 </body>
 17 </html>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kun (laszlo.kun@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Line 8 now specifies `POST` instead of `GET`. If the form is filled out with the same data, the page returned by the CGI script now looks like Figure 10-7.

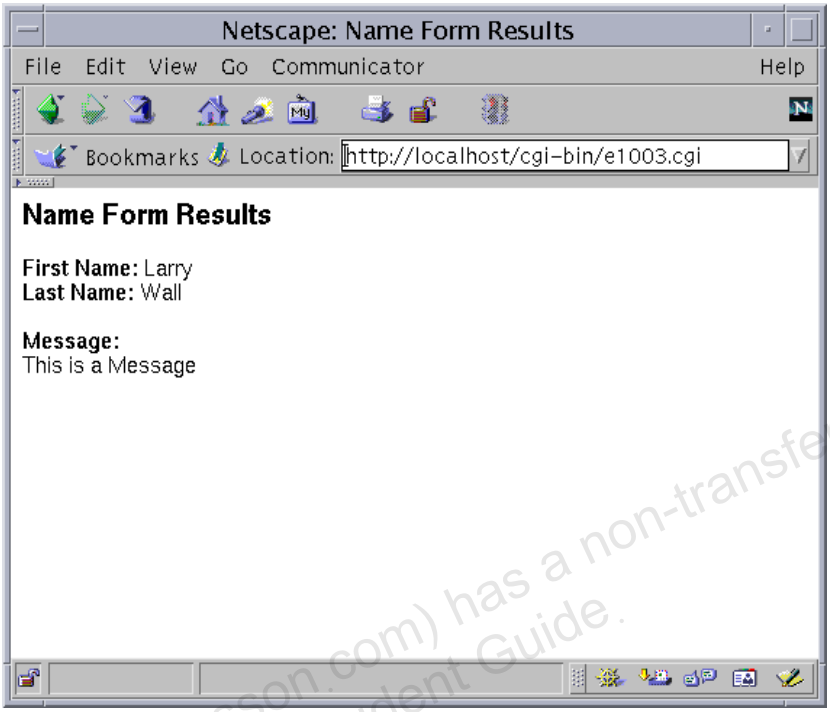


Figure 10-7 CGI Script Results With `POST`

Notice there is no query string in the URL any more. There are trade-offs between using either method.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The GET and POST Methods Compared

Table 10-1 lists the most important features of the GET and POST methods.

Table 10-1 GET and POST Methods

Method Characteristics	GET	POST
Is this the default method?	Yes	No
How is the data sent?	Data is appended to the URL: `http://www.xyz.com/cgi-bin/myscript.plx?name=Duval&fname=Frank`	Data is sent within the body of the HTTP message
How is data received by the script?	Using the environment variable `$ENV{QUERY_STRING}`	Using standard input `<STDIN>`
Is the data length restricted?	Yes, to the maximum length of an environment variable	No

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

CGI.pm-Generated HTML

The `CGI.pm` module includes functions to display HTML tags. You can create the sample page created previously (`e1001.cgi`) using `CGI.pm` functions. The following is a sample:

`e1005.cgi`

```
 1 #!/usr/bin/perl -w
 2 use CGI(":standard");
 3 print header;
 4 print start_html("Database search results");
 5 print h2("Search Results");
 6 print h4("Randal Schwartz: Learning Perl");
 7 print br;
 8 print h4("Larry Wall: Programming Perl");
 9 print h5("Wed Aug 02 12:56:03 2000");
10 print end_html;
```

Each tag is a function. When text is passed to the function, the script encloses the text in the tag specified. The details of `CGI.pm` code tags are beyond the scope of this course, but the previous example should give you an idea of how it works.

The method used to generate the HTML depends on the situation. Considerations include:

- What type of tools are you using to develop your HTML?
- Does your output need to conform to a certain style?
- Will you need to change the HTML output often?

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Object Oriented CGI.pm-Generated HTML

Object-oriented programming is a very big topic. Only some basic concepts associated with cgi will be coverd here. We will not talk about pointers and referencing/dereferencing. We will use the pre-defined CGI.pm module to create objects that help us generate html pages.

A Perl class is a Perl package that includes some special features. The subroutines are called methods and the data is referred to as properties. The methods can be used to manipulate the data.

Creating an object based on the class will provide a private repository for a set of data. The object can be used to call the methods. Here is a sample of OO CGI programming to produce the same result as the previous script.

e1005a.cgi

```
 1 #!/usr/bin/perl -w
 2 use 5.010;
 3 use CGI;     # the object will be able to access all
 4              # of the methods. No need to import them
 5 $worker = CGI->new();  # create the object
 6 say $worker->header; # have the object call the methods
 7 say $worker->start_html("Database search results");
 8 say $worker->h2("Search Results");
 9 say $worker->h4("Randal Schwartz: Learning Perl");
10 say $worker->br();
11 say $worker->h4("Larry Wall: Programming Perl");
12 say $worker->h5("Wed Aug 02 12:56:03 2010");
13 say $worker->end_html();
```

The arrow operator (->) represents a method call. The call to new() is used to return a reference to an object. One advantage to using an object would have been accessing its data. We could have made calls like " @list = $worker->param()" to get the parameter keys from the form which invoked this script. Calling "$worker->param($list[i]) " would get the value associated the the key for this object. Each submission of the form would produce a different object with its own set of data. Also, methods like "excapeHTML()" could be used to take away the dangers associated with some types of malicous user input.

As the two examples show, the CGI.pm can be used in two different ways: a function interface and an object-oriented interface. The CGI.pm was designed for object-orinted use. In general, it is probably more efficient this way. Using such things as FastCGI along with the OO aproach can significantly reduce a server's

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

memory consumption. CGI can be buse to produce the forms, process the information submitted in the forms, and generate the response. A complete treatment of this topic is beyhond the scope of this book.

An OO example of the from response is given below. It is called by e1003b.html.

e1003b.cgi

```
 1 #!/usr/bin/perl
 2 use 5.010;
 3 use CGI;
 4 $worker = CGI->new();
 5
 6 # Get form data
 7 $first = $worker->param("txtFirst");
 8 $last = $worker->param("txtLast");
 9 $message = $worker->param("txtMessage");
10
11 #generate the response
12 say $worker -> header();
13 say $worker -> start_html("Name Form Results");
14 print <<End_of_HTML;
15 <h2>Name Form Results</h2>
16 <p>
17 <b>First Name: </b>$first<br>
18 <b>Last Name: </b>$last<br></p>
19 <p><b>Message:</b><br>
20 $message</p>
21 End_of_HTML
22 say $worker -> end_html;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise: Create CGI Scripts Using Perl

In this exercise, you complete the following tasks:

- Send an HTML page to a browser using a CGI script
- Use a Here document in a CGI script to send an HTML page to a browser
- Read an HTML form using the `GET` or `POST` method

Preparation

To prepare for this exercise:

- Locate the Apache Web server included in the default installation of the Solaris 10 Operating System. (Normally, this is at `/usr/apache/bin`.)
- Start the Apache server using this command:
  `$ /usr/apache/bin/apachectl start`
- Make sure the Web server is serving up HTML documents.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Tasks

Complete the following steps:

1. Create a simple CGI script that prints a simple “Hello Web!” Web page.
2. Create a simple CGI script that prints a simple "Hello Web!" message using a Here document.
3. Create a simple message submission HTML form. Get a name, email address, and message. Then submit these three fields to a CGI script. Use the `CGI.pm` module to process the form and display the data passed to the script using the `GET` method.
4. Modify the form and script you created previously to use the `POST` method of form submission.
5. Modify the form and script you created above to use the object-oriented interface.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Summary

Discussion – Take a few minutes to discuss the experiences, issues, or discoveries you had during the lab exercises.

- Experiences
- Interpretations
- Conclusions
- Applications

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Exercise Solutions

1. Create a simple CGI script that prints a simple “Hello Web!” Web page.

   Suggested solution:

```
lab1101.cgi
  1 #!/usr/bin/perl -w
  2
  3 print "Content-Type: text/html\n\n";
  4 print "<html>\n";
  5 print "<head><title>Hello Web</title></head>\n";
  6 print "<body>\n";
  7 print "<h2>This is generated from my CGI</h2>\n";
  8 print "<p>Hello Web</p>\n";
  9 print "</body>\n";
 10 print "</html>\n";
 11
```

2. Create a simple CGI script that prints a simple “Hello Web!” message using a Here document.

   Suggested solution:

```
lab1102.cgi
  1 #!/usr/bin/perl -w
  2
  3 print <<EndofHTML
  4 Content-Type: text/html\n\n
  5 <html>
  6 <head><title>Hello Web</title></head>
  7 <body>
  8 <h2>This is generated from my CGI</h2>
  9 <p>Hello Web</p>
 10 </body>
 11 </html>
 12 EndofHTML
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

3. Create a simple message submission HTML form. Get a name, email address, and message. Then submit these three fields to a CGI script. Use the `CGI.pm` module to process the form and display the data passed to the script using the `GET` method.

   Suggested solutions:

lab1103.html

```
 1 <html>
 2 <head>
 3 <title>Message Form</title>
 4 </head>
 5 <body bgcolor="#FFFFFF" link="#0000EE">
 6 <h4>Message Form</h4>
 7 <p>
 8 Please enter the following values to submit your message.
 9 </p>
10 <form action="/cgi-bin/lab1103.cgi" method=get>
11 <p>
12 <b>Name: </b><input type="text" name="txtName"><br>
13 <b>E-Mail: </b><input type="text" name="txtEmail"><br>
14 <b>Message:</b><br>
15 <textarea name="txtMessage" rows="4" cols="55">
16 </textarea><br>
17 <input type="submit" name="Submit" value="Submit">
18 <input type="reset" name="Reset" value="Reset">
19 </p>
20 </form>
21 </body>
22 </html>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kutik (laszlo.kutik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

lab1103.cgi

```
 1 #!/usr/bin/perl -w
 2
 3 use CGI(":standard");
 4
 5 $name = param("txtName");
 6 $email = param("txtEmail");
 7 $message = param("txtMessage");
 8
 9 print <<EndofHTML
10 Content-Type: text/html\n\n
11 <html>
12 <head><title>Form CGI</title></head>
13 <body>
14 <h3>Here is the infomation from the form</h3>
15 <p>
16 <b>Name: </b>$name<br>
17 <b>E-Mail: </b>$email<br>
18 <b>Message: </b><br>
19 <p>
20 $message
21 </p>
22 </p>
23 </body>
24 </html>
25 EndofHTML
26
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

4. Modify the form and script you created previously to use the `POST` method of form submission.

   Suggested solutions:

lab1104.html

```
 1 <html>
 2 <head>
 3 <title>Message Form</title>
 4 </head>
 5 <body bgcolor="#FFFFFF">
 6 <h4>Message Form</h4>
 7 <p>Please enter the following values.</p>
 8 <form action="/cgi-bin/lab1104.cgi" method=post>
 9 <p>
10 <b>Name: </b><input type="text" name="txtName"><br>
11 <b>E-Mail: </b><input type="text" name="txtEmail"><br>
12 <b>Message:</b><br>
13 <textarea name="txtMessage" rows="4" cols="55">
14 </textarea><br>
15 <input type="submit" name="Submit" value="Submit">
16 <input type="reset" name="Reset" value="Reset">
17 </p>
18 </form>
19 </body>
20 </html>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

lab1104.cgi

```
 1 #!/usr/bin/perl -w
 2
 3 use CGI(":standard");
 4
 5 $name = param("txtName");
 6 $email = param("txtEmail");
 7 $message = param("txtMessage");
 8
 9 print <<EndofHTML
10 Content-Type: text/html\n\n
11 <html>
12 <head><title>Form CGI</title></head>
13 <body>
14 <h3>Here is the infomation from the form</h3>
15 <p>
16 <b>Name: </b>$name<br>
17 <b>E-Mail: </b>$email<br>
18 <b>Message: </b><br>
19 <p>
20 $message
21 </p>
22 </p>
23 </body>
24 </html>
25 EndofHTML
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

5. Modify the form and script you created above to use the object-oriented interface.

   Suggested solutions:

lab1005.html

```
 1 <html>
 2 <head>
 3 <title>Message Form</title>
 4 </head>
 5 <body bgcolor="#FFFFFF">
 6 <h4>Message Form</h4>
 7 <p>Please enter the following values.</p>
 8 <form action="/cgi-bin/lab1005.cgi" method=post>
 9 <p>
10 <b>Name: </b><input type="text" name="txtName"><br>
11 <b>E-Mail: </b><input type="text" name="txtEmail"><br>
12 <b>Message:</b><br>
13 <textarea name="txtMessage" rows="4" cols="55">
14 </textarea><br>
15 <input type="submit" name="Submit" value="Submit">
16 <input type="reset" name="Reset" value="Reset">
17 </p>
18 </form>
19 </body>
20 </html>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

lab1005.cgi

```
 1 #!/usr/bin/perl
 2 use 5.010;
 3 use CGI;
 4
 5 $q = CGI -> new();
 6 $name = $q  -> param("txtName");
 7 $email = $q -> param("txtEmail");
 8 $message = $q -> param("txtMessage");
 9
10 say $q->header();
11 say $q -> start_html("Form CGI");
12 print <<EndofHTML;
13 <h3>Here is the infomation from the form</h3>
14 <p>
15 <b>Name: </b>$name<br>
16 <b>E-Mail: </b>$email<br>
17 <b>Message: </b><br>
18 <p>
19 $message
20 </p>
21 EndofHTML
22 say $q -> end_html();
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix A

Formats

Defining a Format

Perl was originally designed as an extraction and report language so it functions well as a report generator. Instead of `print` or `printf`, this report generator is based on templates in which complete page formats can be defined.

The definition of a page format begins with the keyword `format`, followed by the format name and an equal sign. The format itself consists of picture lines and argument lines. Picture lines determine the format of one single output line. Argument lines contain the variables that should be written in the format of the preceding picture line. You can add optional comment lines with a beginning #. Finally, the format definition ends with a dot on a line by itself.

```
format FORM1 =
@<<<<<<<< @<<<<<<<<<<<<<< @###### @###### @####.## $
$host, $ip, $kbytesin, $kbytesout, $costs
.
```

Picture lines contain text, which is printed exactly as it looks, and placeholders, which are recognized by the initial `@` or `^`. The following characters determine the kind of justification and the length of the field. Argument lines contain the variables whose values are inserted into the placeholders of the preceding picture line. The variables are listed in order.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The placeholders are described in Table A-1.

Table A-1 Placeholders

<table>
<tr><th>Placeholder</th><th>Description</th></tr>
<tr><td>@<<<<<<</td><td>Left-justified field with fixed length</td></tr>
<tr><td>@>>>>>></td><td>Right-justified field with fixed length</td></tr>
<tr><td>@||||||</td><td>Centered field with fixed length</td></tr>
<tr><td>@####.##</td><td>Right-justified numerical field with fixed-length and optional decimal point</td></tr>
<tr><td>@*</td><td>Multiline field</td></tr>
<tr><td>^|||||||</td><td>Centered field with successive filling</td></tr>
<tr><td>^<<<<<<<</td><td>Left-justified field with successive filling</td></tr>
<tr><td>^>>>>>>></td><td>Right- justified field with successive filling</td></tr>
</table>

In multiline fields (@*), text is written over more than one line, like an ordinary paragraph, without specific formatting. Fields that start with a caret (^) are successively filled up with the text of a variable. That means the text can be spread over more than one caret field.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Defining the Format for the Page Header

The header of a page has to be defined with an additional format block. It is specified in the same way as the rest of the format. Its name has to be the name of the page format with "_TOP" appended.

```
format FORM1_TOP =
Hostname  IP-AddressKB inKB outCharges
========  ============================
.
```

Page headers are automatically printed on top of each page by Perl. Footer formats are not directly supported by Perl; nevertheless, examples of how to create them follow.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Printing Using Formats

To print data in a desired page format, use the `write` command. The `write` command accepts a filehandle as argument. Without an argument, it writes to the currently selected standard filehandle, which is usually `STDOUT`. The page format with the same name as the filehandle is always used for output.

```
write; # prints to STDOUT, using the format STDOUT

write FORM1;
# prints to filehandle FORM1 (which has to be
# opened in advance), using the format FORM1

open FORM1, "%ENV{HOME}/account.dat" ;

for ($i=1;$i<=$nhosts;$i++) {
    write FORM1 ;
}
```

Sample Output:

```
Hostname  IP-Address      KB in   KB out  Charges
========  ==========================================
gogol     193.173.34.113  23456   3405    5.37
dosto     193.173.34.78   7317    1018    1.67
tschechov 193.173.34.227  166493  9644    35.23
    .
    .
    .
```

Note that no further variables or strings can be specified as arguments. All data that has to be printed is already determined by the format.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Changing the Format of a Filehandle

The format of a filehandle cannot be changed directly. If you specify a filehandle to the `write` command, the format with the same name is always used. To choose another format, select the desired filehandle as default. Then, change the format of that default filehandle. To print to it, use `write` without arguments.

```
write;
# writes to STDOUT, using the format STDOUT

select FORM1;
#selects filehandle FORM1 as standard

$~ = FORM2;
specifies  FORM2 as the standard output format

$^ = FORM3_TOP ;
#optional: specifies FORM3_TOP as another header format

write;
# writes to FORM1, using FORM2 as format and FORM3_TOP
# as header
```

Special Variables for Page Formats

Beside `$~` and `$^`, there are three other special variables related to page formats, which are described in Table A-2. They control page size, page numbers, and footers.

Table A-2 Special Variables

Variable	Meaning
`$%`	The current page number
`$=`	The number of lines per page
`$-`	The number of lines remaining on the page

With `"$line = $= - $-"`, you can get the current line number, which can be used, for example, to build a page footer.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix B

References

Introducing References

References are the pillar of advanced Perl programming. Remember the `loop` variable with `foreach` loops, `$_` in `map` and `grep` statements, or the values in `@_` ? All of these are examples of references.

References are the basis of all complex data structures. References can be use in nested structures; for example, arrays of hashes or arrays of arrays. References can pass arrays or hashes to subroutines or create new functions at runtime. In addition, references are the basis for object-oriented programming in Perl.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The Nature of References

A reference is like a pointer in C or a hard link in file systems. It holds the memory address to other data stored in memory. Printing out a reference demonstrates this.

```
print $scref, "\n";
print $arref, "\n";
```

Sample Output:

```
SCALAR(0x80d60c8)
ARRAY(0x80d60fc)
```

References are stored in scalar variables. In the previous example, `$scref` holds a reference to another scalar value, and `$arref` holds a reference to an array. Instead of accessing data directly by its variable name, data can also be accessed by a reference.

You can create a number of references to the same data. The memory address and data type are fixed. Assigning this information to several reference variables provides several ways to access the same data. Perl adds one to a counter for every reference. This ensures that data is not destroyed as long as one reference is pointing to it. Reference examples are shown in Table B-1.

Table B-1 Reference Examples

Memory address	Contents	Variable	Type
0x80d60c8	Snoopy	`$str`	Scalar
0x80d6130	SCALAR(0x80d60c8)	`$ref1`	Scalar
0x80d618e	SCALAR(0x80d60c8)	`$ref2`	Scalar

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@sigsoft.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

References to Scalars

This section reviews how references are used with scalars.

Creating Named and Anonymous References

Create references by placing a backslash \ in front of a variable. The following example demonstrates how to create a reference to a scalar variable.

```
$str = "Milou";
$sref = \$str;
```

Now, `$sref` holds a reference to the scalar variable `$str`. References are not limited to variables. The following script creates an anonymous reference to a string value.

```
$sref = \"Milou";
# reference to the anonymous string "Milou"

$sref = \23;
# reference to the anonymous number 23
```

Using References

To get access to the data a reference points to, the reference must be dereferenced. This is done by enclosing the reference in curly braces or prepending the type of the object that is being pointed to.

```
$str = "Tintin";
$sref = \$str;
print ${$sref}; # prints "Tintin"
${$sref} = "Haddock"; # "Tintin" replaced by "Haddock"
```

Dereferenciation works with references to anonymous strings. However new values cannot be assigned to anonymous scalars. They are treated as constants.

```
$sref = \"Tintin";
print ${$sref}; # prints also "Tintin"
${$sref} = "Haddock" # ERROR
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

You can specify a more complex expression between the braces, as long as the final expression yields a reference.

```
@srefs = (\"Tintin", \"Haddock");
print ${$srefs[0]};
# dereferences the first field of @srefs
```

If there is nothing to evaluate, you can omit the braces.

```
$sref = \"Tournesol";
print $$sref;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

References to Arrays

This section reviews how references are used with arrays.

Creating a Reference

Like scalars, arrays are referenced by the \ operator.

```
@array = ("dupond" , "dupont");
$aref = \@array;
```

A reference to an anonymous array is created with a special notation. The list of values is enclosed in square brackets. If a backslash is placed in front of a list of values, no single reference is returned. Instead, a list of references to each element of the list is returned.

```
$aref = ["dupond" , "dupont"];
# reference to an anonymous array
@lref  = \("tintin", "milou");
# references to anonymous scalars:
# (\"tintin", \"milou")
```

Using a Reference

References to arrays can be dereferenced, again by placing the reference between curly braces.

```
@new = @{$aref};
# @new gets a copy of ("dupond" , "dupont")
join (" + ", @{$aref});
# "dupond + dupont"
print ${$aref}[1];
# "dupont"
```

A second way to dereference single elements of arrays (and hashes) is to use the arrow operator. With the arrow operator, Perl dereferences the variable implicitly. In object-oriented Perl programming, this notation is common.

```
print $aref->[0]; # dupond
$aref->[1] = "tournesol";
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Passing Arrays to a Subroutine

To pass two arrays as arguments to a subroutine, there are two possibilities. The first option to pass them by value: `func(@a, @b)`, but then both arrays are integrated in one single flat list and their original structure is lost. The second option, which avoids both disadvantages, is to pass the arrays by reference. A hash may be passed in exactly the same way.

```
func(\@a,\@b);

sub func {

    my ($aref, $bref) = @_ ;
    print $aref->[1], $bref->[1] ; # prints the second
    element of both arrays.
    }
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

References to Hashes

References to hashes work similarly to references to arrays.

```
%hash = ("hero" => "tintin", "animal" => "milou");
```

Here is an example of a reference to a named hash:

```
$href = \%hash;
```

Here is a reference to an anonymous hash:

```
$href = {hero => "tintin", animal => "milou"};
```

This is how you use a reference:

```
%new = %{$href} ;
print values %{$href};
#("tintin","milou")
```

This is how to use the arrow operator:

```
print $href->{animal};
# "milou"

$href->{hero} = "haddock";
# tintin" replaced by "haddock"
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Subroutines, Filehandles, and Other References

This section describes references to subroutines, filehandles, and other references.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

References to Subroutines

Here are some examples of references to subroutines:

```
$fref1 = \&mysub;
# reference to a named subroutine

$fref2 = sub {print "I hate $_[0]"};
# reference to an anonymous subroutine
&{$fref1}($x);
# using a referenced subroutine

$fref->($y);
# using arrow notation
```

To create a function dispatcher, build anonymous subroutines, store their references in a hash or array, and call them on demand by reference. Another possible use of references to subroutines is functions that are dynamically created at runtime.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

References to Filehandles

References to filehandles are used if filehandles have to be passed to subroutines. Prefix the filehandle by an asterisk (`*`).

```
$fhref = \*FILEHANDLE;
print $fhref "That's pretty flexible.";
```

References to References

You can reference references again. References to references are used when nested structures with more than two dimensions are needed.

```
$str = "abcdef";
$ref = \$str;
$refref = \$ref;
print ${${$refref}}; # prints "abcdef"
print $$$refref ;# the same, but shorter
```


Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Multidimensional Arrays

Multidimensional arrays are described briefly in “Multidimensional Arrays” on page 4-16.

```
@marr = ([1,2],[5,6],[8,9]);
```

A multidimensional array is built from references. When initialized, the inner-dimension arrays must be put in square brackets because they are anonymous arrays. To access a single element, dereference one of the inner arrays, and specify the correct subscript for it.

```
$a =  ${$marr[2]}[1];
$a = $marr[2]->[1];

# dereference $marr[2] and return element 1 of this
# array:   9
```

You can leave out the arrow between two subscipts, leading to the notation used before.

```
$a = $marr[2][1];
```

You can create arrays with more than two dimensions. You can build an array consisting of references, which point to arrays, that contain references again, which point to arrays, and so on. There is no need for the enclosed arrays to have the same length.

```
@mmarr = ([[1,2,3],[4,5,6]] , [[11,12],[13,14]] ,....
print $mmarr[1][0][1] ;
# 12
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Complex Data Structures

You can build complex data structures from arrays of arrays, arrays of hashes, hashes of arrays, hashes of hashes, or even from more nested structures like arrays of hashes of hashes. They are built in the same manner: data is stored in suitable structures (hashes or arrays) and references to these structures are stored again in other structures. As with multidimensional arrays, there is no need for homogeneity. A hash can store references to other hashes as well as references to scalar values and references to arrays.

Arrays of Arrays

The following is an example of arrays of arrays.

```
@arr = (     [a0,a1,...] ,
             [b0,b1,...] , ... );
print $arr[1][0] ;
```

Hashes of Arrays

The following is an example of hashes of arrays.

```
%hash = (      key0=>[va0,va1,...],
               key1=>[vb0,vb1,...], ...);
print $hash{key1}[0];
```

Arrays of Hashes

The following is an example of arrays of hashes.

```
@arr = ({key0,va0,key1,va1,...} ,
          {key0,vb0,key1,vb1,...} , ... );
print $arr[2]{key1};
```

Hashes of Hashes

The following is an example of hashes of hashes.

```
%hash = ( key0=>{keya0,va0,keya1,va1,...} ,
            key1=>{keyb0,vb0,keyb1,vb1,...} , ... );
print $hash{key1}{keyb2};
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Example: Data Structure

The following example shows how a small address and information database is constructed.

Assume a user "jordan" with the following data:

- Some single values as scalars (Internet Protocol [IP] address, host name, and disk usage):

```
$ip      = "154.65.211.197";
$host  = "sparc";
$disk  = 345772;
```

- The complete address in an array:

```
@addr = ("Jeff Jordan", "6, Helios Place", "London",
"01023452");
```

- Accounting data for the last six months in an array:

```
@acc = (354, 656, 223, 355, 995, 331);
```

- A scheduler in a hash:

```
%alert = ("23.7.00", "N.T., London", "8.8.00",
"Lafayette, Paris", "13.8.00", "Springer, Berlin");
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

To hold all this data in one single structure, you can use a hash or an array. This example uses a hash: `%jordan`. You can store the three scalar values as simple key and value pairs. The name of the former variable serves as key; for example `$jordan{"ip"}`. For the two arrays, you must store references; for example `$jordan{"addr"} = \@addr`. The hash works in the same way.

```
%jordan
%jordan = ( "ip", "", "host", "", "disk", "", "addr", [],
"acc", [], "alert", {} ) ;
# Initializing not needed, used just for illustration.

$jordan{"ip"} = "154.65.211.197" ;
$jordan{"host"}  = "sparc" ;
$jordan{"disk"}  = 345772 ;

$jordan{"addr"}= [ "Jeff Jordan", "6, Helios Place",
"London", "01023452" ];

$jordan{"acc"}  = [354, 656, 223, 355, 995, 331] ;

$jordan{"alert"} = {"23.7.00","N.T., London",
"8.8.00","Lafayette, Paris", "13.8.00","Springer, Berlin"};
```

The hash for the user `"jordan"` contains an entire data structure. But what about other users? Do you have to build a separate hash for every user? Create a new hash `%stuff` that will contain user names as keys and references to the data structures described previously as values.

```
%staff
%staff = ( "jordan",{},"marple",{},"poirot",{},....) ;
$staff{"jordan"} = \%jordan ;
```

Perfect! There is only one structure for all data of all users. From a technical point of view, it is a hash that contains references to hashes, which, in turn, contains single-value references to arrays and a reference to a hash.

```
print $staff{"jordan"} , "\n" ;
print $staff{"jordan"}{"host"} , "\n" ;
print $staff{"jordan"}{"addr"} , "\n" ;
print $staff{"jordan"}{"addr"}[1] , "\n" ;
```

Sample output:

```
HASH(0x80d9cdc)
sparc
ARRAY(0x80dc980)
6, Helios Place
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix C

Signals and Interprocess Communication

Sending and Receiving Signals

To send a signal to another process, use the `kill` command. It works nearly the same way as its UNIX equivalent. The `kill` command expects the number or name of the signal as its first argument. By prefixing the number with a minus sign, a signal is sent to a complete program group. As a second argument, `kill` expects a list of process IDs to which the signal should be delivered.

```
kill sig, procid-list ;

kill 9, 2340;  # kills (sends SIGKILL) the
               # process with ID 2340
kill KILL, 2340;# does the same
```

The list of all signals can be accessed by the UNIX command `kill -l` or by `man -s5 signal`.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The signals currently defined by <signal.h> are described in Table C-1.

Table C-1 Signals

Name	Value	Default Event
SIGHUP	1	ExitHangup (see termio(7I))
SIGINT	2	ExitInterrupt (see termio(7I))
SIGQUIT	3	CoreQuit (see termio(7I))
SIGILL	4	CoreIllegal Instruction
SIGTRAP	5	CoreTrace or Breakpoint Trap
SIGABRT	6	CoreAbort
SIGEMT	7	CoreEmulation Trap
SIGFPE	8	CoreArithmetic Exception
SIGKILL	9	ExitKilled
SIGBUS	10	CoreBus Error
SIGSEGV	11	CoreSegmentation Fault
SIGSYS	12	CoreBad System Call
SIGPIPE	13	ExitBroken Pipe
SIGALRM	14	ExitAlarm Clock
SIGTERM	15	ExitTerminated
SIGUSR1	16	ExitUser Signal 1

You are not limited to sending signals from within your program. You can also receive them. To determine how the program should react to an incoming signal, the special hash %SIG is used. The signal names serve as keys of this hash: HUP, INT, QUIT, TERM, KILL, and so on.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The values of the keys determine how the program processes the corresponding signal. By default, all values are undefined, which results in the usual behavior (terminate with Control-C, and so on). If the program should ignore a signal, the value of the respective key must be `IGNORE`. To reset default behavior, the value must be set to `DEFAULT`.

```
$SIG{'INT'} = 'IGNORE';   # now your program
                          # ignores Ctrl-C
$SIG{'INT'} = 'DEFAULT';  # from now on the program
                          # can be interrupted again
```

Start the following script in the background and then try to kill it (using `kill pid`). The `TERM` signal is ignored.

```
appc01.pl
  1 #!/usr/bin/perl -w
  2 $SIG{'TERM'} = 'IGNORE';
  3 sleep 120;
```

It is also possible to write complex signal handlers that execute when a specific signal is delivered. To do this, a reference to the corresponding subroutine has to be assigned as the value to the `SIG` key.

```
$SIG{'INT'} = \&myfunc ;  # instead of being
sub myfunc {              # interrupted, your program
    print "Bad Luck!\n"   # prints "Bad Luck!"
}
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Interprocess Communication

Perl knows several methods of interprocess communication (IPC). Although IPC is an advanced subject that cannot be covered in a basic course, a short overview of its different methods follows.

Signals

A signal is sent with `kill` and processed by a signal handler, which is defined through `%SIG` You can use signals to control processes or to synchronize read and write actions on the same file.

Anonymous Pipes

Pipes to another process can be opened by `open(PH, "| process")` and `open(PH, "process |")`. With this method, the shell establishes the pipe and starts the required process.

You can create internal pipes in Perl. When creating child processes with the `fork` function, you could set up a pipe between them using the following command:

```
pipe READPH, WRITEPH ;
```

One process can write to the handle `WRITEPH`, and the other can read from `READPH`.

Named Pipes

Anonymous pipes are restricted to processes that are related. Therefore, independent processes must use named pipes. With named pipes, an entry in the file system is created that can be accessed from both the reading and the writing process.

When you run `ls -l`, named pipes are marked with a `"p"` as the first attribute. A named pipe can be created by the UNIX command `mknod pipename p` or `mkfifo`. Use `open`, `<...>`, and `print`, or the unbuffered system calls `sysopen`, `sysread`, and `syswrite` to perform the respective tasks. To establish communication between two processes, one process has to write to the named pipe, and the other process has to read from it.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Shared Memory, Sockets, and RPC

Perl also implements System V IPC. Semaphores are supported as well as message queues and shared memory. Share memory is created with `shmget`. Shared memory is read by `shmread`, and `shmwrite` writes to shared memory.

For process communication between different hosts, Perl offers sockets and remote procedure call (RPC) methods.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix D

Perl Debugger

Using the Perl Debugger

With the Perl debugger, you can examine your program at runtime and do the following:

- Go through the program step-by-step
- Show the contents of variables at any time
- Evaluate additional expressions that are not contained in your program
- Change variable contents
- Set breakpoints wherever they are required

The Perl debugger is called by the command line option `-d`.

```
perl -d myscript.plx
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.
László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The following example demonstrates how the debugger works. The script returns the number of files in a specified directory, but it does not work because the glob operator does not behave as expected.

```
appd01a.plx
  1 #!/usr/bin/perl -w
  2 # Count # of directories
  3
  4 for (@ARGV) {
  5
  6   $num = <$_/*>;
  7   print "directory $_ contains $num files.\n" ;
  8
  9 }
```

```
$ appd01a.plx .
directory . contains ./appd01a.plx files.
```

What is wrong? To find out, you start the debugger.

```
$ perl -d appd01a.plx .

Loading DB routines from perl5db.pl version 1.25
Emacs support available.

Enter h or 'h h' for help.

main::(appd01a.pl:4):   for (@ARGV) {
  DB<1>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

The debugger always shows the next line to be processed. Here, it is Line 4:

```
foreach (@ARGV) {.
```

Then, you are asked for the first debug command:

```
DB<1>
```

By typing `h` or `h h`, you can get help:

```
DB<1> h
List/search source lines:               Control script execution:
  l [ln|sub]  List source code            T           Stack trace
  - or .      List previous/current line  s [expr]    Single step [in expr]
  v [line]    View around line            n [expr]    Next, steps over subs
  f filename  View source in file         <CR/Enter>  Repeat last n or s
  /pattern/ ?patt?   Search forw/backw    r           Return from subroutine
  M           Show module versions        c [ln|sub]  Continue until position
Debugger controls:                        L           List break/watch/actions
  o [...]     Set debugger options        t [expr]    Toggle trace [trace expr]
  <[<]|{[{]|>[>] [cmd] Do pre/post-prompt b [ln|event|sub] [cnd] Set breakpoint
  ! [N|pat]   Redo a previous command     B ln|*      Delete a/all breakpoints
  H [-num]    Display last num commands   a [ln] cmd  Do cmd before line
  = [a val]   Define/list an alias        A ln|*      Delete a/all actions
  h [db_cmd]  Get help on command         w expr      Add a watch expression
  h h         Complete help page          W expr|*    Delete a/all watch exprs
  |[|]db_cmd  Send output to pager        ![!] syscmd Run cmd in a subprocess
  q or ^D     Quit                        R           Attempt a restart
Data Examination:     expr     Execute perl code, also see: s,n,t expr
  x|m expr       Evals expr in list context, dumps the result or lists methods.
  p expr         Print expression (uses script's current package).
  S [[!]pat]     List subroutine names [not] matching pattern
  V [Pk [Vars]]  List Variables in Package.  Vars can be ~pattern or !pattern.
  X [Vars]       Same as "V current_package [Vars]".  i class inheritance tree.
  y [n [Vars]]   List lexicals in higher scope <n>.  Vars same as V.
For more help, type h cmd_letter, or run man perldebug for all docs.
  DB<2>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Now, you can continue with the example. Tell the debugger that it should execute the first line. You can use three commands to do this, as shown in Table D-1.

Table D-1 Debugger Execution Commands

Command	Description
`s`	Executes the next single step
`n`	Executes the next single step, but processes a subroutine in one step
`<cr>`	Repeats the last n or s command

In this case, `n` is typed to execute the next step.

```
DB<2> n
main::(appd01a.plx:6):      $num = <$_/*>;
  DB<2> <CR>
main::(appd01a.plx:7):      print "directory $_ contains $num files.\n" ;
  DB<2>
```

At this position, you want to take a look at some variables. What is contained in `$_`, `$num`, and `<$_/*>`?

```
DB<2> p $_
.
  DB<3> p $num
./appd01a.plx
  DB<4> p <$_/*>
./appd01.plx./appd01a.plx./appd01b.plx./temp.pl./temp.txt
```

The glob operator seems to work. But instead of the number of elements, `$num` gets the first element of the list, although in scalar context. Obviously the glob operator does not recognize the scalar context.

As a test, you can assign `<$_/*>` to an array first and then evaluate the array in scalar context.

```
DB<5> @arr = <$_/*>

  DB<6> $num = @arr

  DB<7> p $num
5
  DB<8>
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Finally, that is the desired result. Next, you quit the debugger by typing `q` and correct the wrong line in the program.

```
DB<8> q
$
```

```
appd01b.plx .
  1 #!/usr/bin/perl -w
  2 # Count # of directories
  3
  4 for (@ARGV) {
  5
  6   @arr = <$_/*>;
  7   $num = @arr;
  8   print "directory $_ contains $num files.\n" ;
  9
 10 }
```

Now, when you run the corrected program, you get the desired results:

```
$ appd01b.plx .
directory . contains 5 files.
```

Refer to the help command of the debugger or the man page `perldebug` for a complete list of debugging commands. Note that you can use a pipe in front of the debugger's help command to output the page:

```
| h
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix E

Perl Special Variables

Special Variables

This appendix is a reference for Perl special variables. If you want to use the equivalent long variable names instead of the symbolic forms, put this at the top of your program:

```
use English;
```

Special Variable List

He explanations that follow for the different variables are taken from the Perl man page `perlvar`.

`$_ ($ARG)`

This variable is the default input, output, and pattern-searching space. This is used as the default with many scalar functions, such as `ord` or `int`; with file test operators; with many list operators, such as `print`, `grep` or `match`; with pattern matching operations; with `foreach` loops; and as the default input variable for filehandles.

`$<digit>`

This variable contains the subpattern from the corresponding set of parentheses in the last pattern matched. Patterns matched in nested blocks that have been exited already are not counted. They are all read only.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

`$& ($MATCH)`

This variable is the string matched by the last successful pattern match (not counting any matches hidden within a `BLOCK` or `eval` enclosed by the current `BLOCK`). This variable is read only.

`$` ($PREMATCH)`

This variable is the string preceding whatever was matched by the last successful pattern match (not counting any matches hidden within a `BLOCK` or `eval` enclosed by the current `BLOCK`). This variable is read only.

`$' ($POSTMATCH)`

This variable is the string following whatever was matched by the last successful pattern match (not counting any matches hidden within a `BLOCK` or `eval` enclosed by the current `BLOCK`). This variable is read only.

`$+ ($LAST_PAREN_MATCH)`

The last bracket matched by the last search pattern. This is useful if you don't know which of a set of alternative patterns matched. This variable is read only.

`$* ($MULTILINE_MATCHING)`

Use of `$*` is deprecated in modern Perl.

Set this variable to 1 to do multiline matching within a string, or set this variable to 0 to tell Perl that it can assume that strings contain a single line for the purpose of optimizing pattern matches. Pattern matches on strings containing multiple newlines can produce confusing results when `$*` is 0. The default is 0. Note that this variable influences the interpretation of only `^` and `$`. A literal newline can be searched for even when `$*` equals 0.

`$.($INPUT_LINE_NUMBER, $NR)`

This variable is the current input line number for the last filehandle from which you read (or on which you performed a `seek` or `tell`). An explicit close on a filehandle resets the line number. Because `<>` never does an explicit close, line numbers increase across `ARGV` files. Localizing `$.` has the effect of also localizing Perl's notion of the last read filehandle.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

$/ ($INPUT_RECORD_SEPARATOR, $RS)

This variable is the input record separator, which is newline by default. It works like awk's RS variable, including treating empty lines as delimiters if it is set to the null string.

Note – An empty line cannot contain any spaces or tabs.

You can set it to a multicharacter string to match a multicharacter delimiter or to undef to read to the end of a file. Note that setting it to \n\n means something slightly different than setting it to "", if the file contains consecutive empty lines. Setting it to "" treats two or more consecutive empty lines as a single empty line. Setting it to \n\n causes it to blindly assume that the next input character belongs to the next paragraph, even if it is a newline. Remember, the value of $/ is a string, not a regular expression.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

`$|` (`$OUTPUT_AUTOFLUSH`)

If this variable is set to nonzero, it forces a flush right away and after every write or print on the currently selected output channel. The default is 0 (regardless of whether the channel is actually buffered by the system or not; `$|` tells you only whether you have asked Perl explicitly to flush after each write). Note that `STDOUT` is typically line-buffered if output is to the terminal, and `STDOUT` is block-buffered, otherwise. Setting this variable is useful primarily when you are outputting to a pipe, such as when you are running a Perl script under `rsh` and you want to see the output as it is happening. This has no effect on input buffering.

`$,` (`$OUTPUT_FIELD_SEPARATOR, $OFS`)

This variable is the output field separator for the `print` operator. Ordinarily, the `print` operator prints out the comma-separated fields you specify. To get behavior more like `awk`, set this variable as you would set `awk`'s `OFS` variable to specify what is printed between fields.

`$\` (`$OUTPUT_RECORD_SEPARATOR, $ORS`)

This variable is the output record separator for the `print` operator. Ordinarily, the `print` operator prints out the comma-separated fields you specify, with no trailing newline or record separator assumed. To get behavior more like `awk`, set this variable as you would set `awk`'s `ORS` variable to specify what is printed at the end of the print.

`$"` (`$LIST_SEPARATOR`)

This variable is like `$`, except that it applies to array values interpolated into a double-quoted string (or similarly interpreted string). The default is a space.

`$;` (`$SUBSCRIPT_SEPARATOR, $SUBSEP`)

The subscript separator for multidimensional array emulation. If you refer to a hash element as `$foo{$a,$b,$c, }`, it really means `$foo{join($;, $a, $b, $c)}`. However, do not put `@foo{$a,$b,$c}` (which is a slice; note the `@`), which means (`$foo{$a},$foo{$b},$foo{$c}`). The default is `\034`, the same as `SUBSEP` in `awk`. Note that if your keys contain binary data, there might not be any safe value for `$;`. Consider using real multidimensional arrays.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

$# ($OFMT)

Use of $# is deprecated.

This variable is the output format for printed numbers. This variable is an attempt to emulate awk's OFMT variable. There are times, however, when awk and Perl have differing notions of what is, in fact, numeric. The initial value is %.ng, where n is the value of the macro DBL_DIG from your system's float.h. This is different from awk's default OFMT setting of %.6g, so you need to set $# explicitly to get awk's value.

$% ($FORMAT_PAGE_NUMBER)

This variable is the current page number of the currently selected output channel.

$= ($FORMAT_LINES_PER_PAGE)

This variable is the current page length (printable lines) of the currently selected output channel. The default is 60.

$- ($FORMAT_LINES_LEFT)

This variable is the number of lines left on the page of the currently selected output channel.

$~ ($FORMAT_NAME)

This variable is the name of the current report format for the currently selected output channel. The default is the name of the filehandle.

$^ ($FORMAT_TOP_NAME)

This variable is the name of the current top-of-page format for the currently selected output channel. The default is the name of the filehandle with _TOP appended.

$: ($FORMAT_LINE_BREAK_CHARACTERS)

This variable is the current set of characters after which a string may be broken to fill continuation fields (starting with ^) in a format. The default is \n-, to break on white space or hyphens.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

$^L ($FORMAT_FORMFEED)

This variable formats output to perform a form feed. The default is \f.

$^A ($ACCUMULATOR)

This variable is the current value of the write accumulator for format lines. A format contains formline commands that put their result into $^A. After calling its format, write prints out the contents of $^A and empties, so you never actually see the contents of $^A unless you call formline yourself and then look at it. Refer to the perlform(5) man page and the formline entry in the perlfunc(1) man page.

$? ($CHILD_ERROR)

This variable is the status returned by the last pipe closed, back quotes (``) command, or system operator. Note that this is the status word returned by the wait system call (or else is made to look like it). Thus the exit value of the subprocess is actually ($? >> 8), and $? & 255 gives which signal, if any, the process died from, and whether there was a core dump. Inside an END subroutine, $? contains the value that is going to be given to exit. You can modify $? in an END subroutine to change the exit status of the script.

$! ($OS_ERROR, $ERRNO)

If used in a numeric context, this variable yields the current value of errno, with all the usual caveats. (This means that you should not depend on the value of $! to be anything in particular unless you have got a specific error return indicating a system error.) If used in a string context, this variable yields the corresponding system error string. You can assign $! to set errno if, for instance, you want $! to return the string for error n or you want to set the exit value for the die operator.

$^E ($EXTENDED_OS_ERROR)

This variable provides more specific information about the last system error than that which is provided by $!, if available. (If not, it is just $! again.) The caveats mentioned in the description of $! apply here, too.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

`$@ ($EVAL_ERROR)`

This variable is the Perl syntax error message from the last `eval` command. If null, the last `eval` is parsed and executed correctly (although the operations you invoked may have failed in the normal fashion). Note that warning messages are not collected in this variable. You can, however, set up a routine to process warnings by setting `$SIG{__WARN__}`.

`$$ ($PROCESS_ID, $PID)`

This variable is the process number of the Perl process running this script.

`$< ($REAL_USER_ID, $UID)`

This variable is the real `uid` of this process.

`$> ($EFFECTIVE_USER_ID, $EUID)`

This variable is the effective `uid` of this process.

Note – `$<` and `$>` can be swapped only on machines supporting `setreuid`.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

`$( ($REAL_GROUP_ID, $GID)`

This variable is the real `gid` of this process. If you are on a machine that supports membership in multiple groups simultaneously, this gives a space-separated list of the groups you are in. The first number is the one returned by `getgid`, and the subsequent numbers are the ones returned by `getgroups`, one of which may be the same as the first number. However, a value assigned to `$(` must be a single number used to set the real `gid`. So the value given by `$(` should not be assigned back to `$(` without it being forced to be numeric, such as by adding zero.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

$) ($EFFECTIVE_GROUP_ID, $EGID)

This variable is the effective `gid` of this process. If you are on a machine that supports membership in multiple groups simultaneously, it gives a space-separated list of the groups you are in. The first number is the one returned by `getegid` and the subsequent ones are returned by `getgroups`, one of which may be the same as the first number. Similarly, a value assigned to `$)` must also be a space-separated list of numbers. The first number is used to set the effective `gid`, and the rest (if any) are passed to `setgroups`. To get the effect of an empty list for `setgroups`, just repeat the new effective `gid`; that is, to force an effective `gid` of 5 and an effectively empty `setgroups` list, use `$) = "5 5";`.

Note – `$<`, `$>`, `$(` and `$)` can be set only on machines that support the corresponding `set[re][ug]id` routine. `$(` and `$)` can be swapped only on machines supporting `setregid`.

$0 ($PROGRAM_NAME)

This variable contains the name of the file containing the Perl script being executed. On some operating systems, assigning a value to `$0` modifies the argument area that the `ps(1)` program sees. This is more useful as a way of indicating the current program state than it is for hiding the program you are running.

$[

Its use is discouraged.

This variable is the index of the first element in an array and of the first character in a substring. The default is 0, but you could set it to 1 to make Perl behave more like `awk` (or Fortran) when subscripting and when evaluating the `index` and `substr` functions. As of Perl 5, assignment to `$[` is treated as a compiler directive and cannot influence the behavior of any other file.

$] ($PERL_VERSION)

This variable is the version plus patch level of the Perl interpreter divided by 1000. You can use this variable to determine whether the Perl interpreter executing a script is in the right range of versions.

$^D ($DEBUGGING)

This variable is the current value of the debugging flags.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

`$^F ($SYSTEM_FD_MAX )`

This variable is the maximum system file descriptor, which is ordinarily 2. System file descriptors are passed to `exec` processes, while higher file descriptors are not. Also, during an `open`, system file descriptors are preserved even if the `open` fails. (Ordinary file descriptors are closed before the `open` is attempted.) Note that the close-on-exec status of a file descriptor is decided according to the value of `$^F` at the time of the open, not at the time of the execution.

`$^H`

This variable is the current set of syntax checks enabled by `use strict` and other block-scoped compiler hints.

`$^I ($INPLACE_EDIT)`

This variable is the current value of the inplace-edit extension. Use `undef` to disable inplace editing.

`$^O ($OSNAME)`

This variable is the name of the operating system under which this copy of Perl is running. The value is identical to `$Config{'osname'}`.

`$^P ($PERLDB)`

This variable is the internal variable for debugging support. Different bits mean the following (subject to change):

0x01	Debug subroutine enter and exit
0x02	Line-by-line debugging
0x04	Switch off optimizations
0x08	Preserve more data for future interactive inspections
0x10	Keep information about source lines on which a subroutine is defined
0x20	Start with single-step on

Note that some bits might be relevant at compile-time only and some at runtime only. This is a new mechanism, and the details may change.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

$^S

This variable is the current state of the interpreter. It is undefined if parsing of the current module or evaluation is not finished (which might happen in $SIG{__DIE__} and $SIG{__WARN__} handlers). It is true if inside an eval; otherwise, it is false.

$^T ($BASETIME)

This variable is the time since the script began running, in seconds since the epoch (beginning of 1970). The values returned by the -M, -A, and -C file tests are based on this value.

$^W ($WARNING)

This variable is the current value of the warning switch, which is either true or false.

$^X ($EXECUTABLE_NAME)

This variable is the name that the Perl binary itself was executed as, from C's argv[0].

$ARGV

This variable contains the name of the current file when reading from <>.

@ARGV

The array @ARGV contains the command-line arguments intended for the script. Note that $#ARGV is the number of arguments minus 1, because $ARGV[0] is the first argument, not the command name.

@INC

The array @INC contains the list of places to look for Perl scripts to be evaluated by the do EXPR, require, or use constructs. It initially consists of the arguments to any -I command-line options, followed by the default Perl library. (It is probably /etc/perl/lib, followed by ., to represent the current directory.) If you need to modify this at runtime, you should use the use lib pragma to get the machine-dependent library properly loaded. For example:

```
use lib '/mypath/libdir/';
use SomeMod;
```

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

@_

Within a subroutine, the array @_ contains the parameters passed to that subroutine.

%INC

The hash %INC contains entries for each file name that has been included through do or require. The key is the file name you specified, and the value is the location of the file actually found. The require command uses this hash to determine whether a given file has already been included.

%ENV

The hash %ENV contains your current environment. Setting a value in ENV changes the environment for child processes.

%SIG

The hash %SIG sets signal handlers for various signals. It contains values only for the signals actually set within the Perl script. If your system has the sigaction function, then signal handlers are installed using it. This means you get reliable signal handling. If your system has the SA_RESTART flag, the flag is used when signals handlers are installed. This means that supported system calls continue rather than returning when a signal arrives.

You can set certain internal hooks using the %SIG hash. The routine indicated by $SIG{__WARN__} is called when a warning message is about to be printed. The warning message is passed as the first argument. The presence of a __WARN__ hook causes the ordinary printing of warnings to STDERR to be suppressed. You can use this to save warnings in a variable or to turn warnings into fatal errors.

The routine indicated by $SIG{__DIE__} is called when a fatal exception is about to be thrown. The error message is passed as the first argument. When a __DIE__ hook routine returns, the exception processing continues as it would have in the absence of the hook, unless the hook routine itself exits by a goto, a loop exit, or a die. The __DIE__ handler is explicitly disabled during the call, so that you can die from a __DIE__ handler.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix F

Perl Functions

Perl Function Reference

This appendix is a reference for the Perl functions found in this course. Perl offers many networking and system-database-access functions, as well as other functions. For a complete list of built-in Perl functions refer to the `perlfunc` manual page.

Perl Functions by Category

Table F-1 through Table F-11 provide brief descriptions of Perl functions by topic. A detailed explanation of each function in alphabetical order is provided after the topical listing.

Table F-1 String Functions

Function	Description
`chomp`	Removes the ending newline
`chop`	Removes the ending character
`crypt`	Encrypts strings
`defined`	Tests if the value is defined
`eval`	Evaluates Perl expressions
`index`	Gives the position of a substring within a string
`lc`	Returns a string in lowercase
`lcfirst`	Returns a string with the first letter in lowercase
`length`	Gives the length of a string

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kajik (laszlo.kajik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-1 String Functions (Continued)

Function	Description
`quotemeta`	Backslashes all special characters
`reverse`	Reverses a string
`rindex`	Like `index` but starts looking from the end
`substr`	Returns a substring of a string
`uc`	Returns a string in uppercase
`ucfirst`	Returns a string with the first letter in uppercase
`undef`	Undefines a variable

Table F-2 Array Functions

Function	Description
`defined`	Tests if an array is defined
`grep`	Extracts the matching elements of a list
`join`	Joins elements of a list to a single string
`map`	Performs the commands for all elements of a list
`pop`	Removes the last element of an array
`push`	Pushes new elements onto the end of an array
`reverse`	Returns a list in reverse order
`scalar`	Forces a scalar context
`shift`	Removes the first element of an array
`sort`	Returns a sorted list
`splice`	Splices an array into two subarrays
`split`	Splits a string in separated fields
`undef`	Removes the whole array
`unshift`	Pushes new elements to the front of an array

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-3 Hash Functions

Function	Description
`defined`	Tests if a hash or value is defined
`delete`	Deletes a key/value pair of a hash
`each`	Returns a key and its value
`exists`	Checks if a key exists in a hash
`keys`	Returns a list of all keys
`undef`	Removes the whole hash
`values`	Returns a list of all values

Table F-4 Arithmetic Functions

Function	Description
`abs`	Absolute value
`atan2`	Arctangent of x/y
`cos`	Cosine
`exp`	e to the power of
`int`	Integer portion
`log`	Natural logarithm
`rand`	Random number
`sin`	Sine
`sqrt`	Square root
`srand`	Seed rand

Table F-5 Conversion Functions

Function	Description
`chr`	Returns the character for a numeric ASCII value
`hex`	Returns the decimal value of a hexadecimal string
`oct`	Returns the decimal value of a octal or hexadecimal string
`ord`	Returns the numeric ASCII value of a character

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-5 Conversion Functions (Continued)

Function	Description
`pack`	Converts values into a binary structure using a template
`unpack`	Converts a binary structure back into the original values
`vec`	Returns a value from a bit vector

Table F-6 Time Functions

Function	Description
`gmtime`	Returns a readable time structure in Greenwich Mean Time (GMT)
`localtime`	Returns a readable time structure in the local time zone
`time`	Returns the seconds since 1.1.1970

Table F-7 Input/Output Functions

Function	Description
`binmode`	Indicates a file to be read in binary mode (DOS)
`close`	Closes a filehandle
`dbmclose`	Closes a DBM database file
`dbmopen`	Opens a DBM database file and attaches it to a hash
`eof`	Checks for the end of file
`fcntl`	Interfaces to the `fcntl(2)` file control function
`fileno`	Returns the file descriptor of a filehandle
`flock`	Provides a file locking function
`format`	Defines a page format
`getc`	Gets a character
`ioctl`	Interfaces to the `ioctl(2)` system call
`open`	Opens a file and associates it with a filehandle

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-7 Input/Output Functions (Continued)

Function	Description
`pipe`	Returns two connected pipes
`print`	Prints to `STDOUT` or a filehandle
`printf`	Prints out a formatted string
`read`	Reads data of fixed length
`seek`	Sets file pointer at new position
`select`	Returns currently selected filehandle
`select`	Interfaces to the `select(2)` file descriptor system call
`sprintf`	Returns a formatted string
`sysopen`	Provides a system call to `open(2)` to bypass `stdio`
`sysread`	Provides a system call to `read(2)` to bypass `stdio`
`syswrite`	Provides a system call to `write(2)` to bypass `stdio`
`tell`	Returns a file-pointer position
`write`	Prints to a filehandle with a special page format

Table F-8 File Operation Functions

Function	Description
`chmod`	Changes file permissions
`chown`	Changes owner and group
`link`	Creates a hard link
`lstat`	Provides a variation of `stat`
`mkdir`	Creates a new directory
`readlink`	Reads a link
`rename`	Renames a file
`rmdir`	Removes an empty directory
`stat`	Reads inode information
`symlink`	Creates a symbolic link

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kujik (laszlo.kujik@eriesoft.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-8 File Operation Functions (Continued)

Function	Description
`truncate`	Truncates a file
`unlink`	Deletes a file
`utime`	Changes time stamps

Table F-9 Directory Functions

Function	Description
`closedir`	Closes a directory handle
`opendir`	Opens a directory handle
`readdir`	Reads the next entry of a directory
`rewinddir`	Sets the directory pointer to the beginning
`seekdir`	Sets the directory pointer to a new position
`tellir`	Returns the position of the directory pointer

Table F-10 Processes and System Interaction Functions

Function	Description
`alarm`	Sends `SIGALARM` to the current process
`chdir`	Changes the current directory
`chroot`	Changes the root directory
`defined`	Tests if a subroutine is defined
`die`	Terminates the program
`dump`	Causes a core dump
`exec`	Executes another program instead of the current one
`exit`	Terminates the block, subroutine, or program
`fork`	Creates a child process
`getlogin`	Gets the login name

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-10 Processes and System Interaction Functions (Continued)

Function	Description
`getpgrp`	Gets the process group
`getppid`	Gets the parent process ID
`getpriority`	Gets the priority of a process
`glob`	Returns a pattern after file name expansion
`kill`	Sends a signal to a process
`setpgrp`	Sets the process group of a process
`setpriority`	Sets the priority for a process
`sleep`	Sleeps some seconds
`syscall`	Executes a system call
`system`	Runs a command without terminating the current program
`times`	Returns the user and system time of a process
`umask`	Gets or sets the `umask`
`wait`	Waits for the child
`waitpid`	Waits for a specified child
`warn`	Prints a warning message

Table F-11 Subroutine and Module Functions

Function	Description
`bless`	Declares a data structure as object
`caller`	Returns information about the calling routine
`goto`	Jumps to a labelled line or a subroutine
`import`	Imports variable or function names from a module
`local`	Creates a variable with local values
`my`	Creates a private variable
`no`	Unimports previously imported code
`package`	Declares a namespace

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-11 Subroutine and Module Functions (Continued)

Function	Description
`ref`	Returns the type of a reference
`require`	Imports code from a module
`return`	Terminates a subroutine and returns a value
`tie`	Binds a variable to a package
`tied`	Returns the tied object
`undef`	Removes the subroutine
`untie`	Removes the binding of a variable
`use`	Imports code and names from a module

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Alphabetical List of Perl Functions

This section provides a detailed description of each function. The functions are listed in alphabetical order.

Note – All functions that take a string as an argument can also take any expression that evaluates to a string. Every function that takes an array as argument can also work with a simple list of values.

abs

```
abs value       abs
```

The `abs` function returns the absolute value of the argument or `$_`.

alarm

```
alarm value
```

The `alarm` function sends a `SIGALARM` signal to the executing Perl process after `value` seconds. Every call of `alarm` resets the timer to the new value; `alarm(0)` turns it off. It returns the number of seconds remaining on the timer.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

atan2

```
atan2 x, y
```

The `atan2` function returns the arctangent of `x/y` in the range of `-p` to `p`.

binmode

```
binmode filehandle
```

The `binmode` function is used only with operating systems that distinguish between files in text and files in binary format. It indicates that the filehandle has to be read in binary mode. For UNIX systems, `binmode` does not change anything.

bless

```
bless reference [, classname]
```

The `bless` function takes a reference to a data structure (usually an anonymous hash) and marks it as an object of the current or the specified class. It returns the given reference.

caller

```
caller       caller value
```

The `caller` function is used within a subroutine to get information about who has called the actual function. If invoked without an argument, `caller` returns the following list: (`$package`, `$filename`, `$line`). When `value` is specified, it goes back the corresponding number of stack frames and returns an extended list about this stack information (`$package`, `$filename`, `$line`, `$subname`, `$hasargs`, `$wantarray`).

chdir

```
chdir directory       chdir
```

The `chdir` function changes the current directory to the specified one. It returns 1 if successful, and 0 otherwise. If `directory` is not specified, `chdir` changes to the home directory.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

chmod

```
chmod perm, filelist
```

The chmod function changes access permissions for a list of files. Permissions have to be specified numerically (using four digits). It returns the number of changed files.

chomp

```
chomp string        chomp array        chomp
```

The chomp function removes the line endings of a string, list, or $_ only if the last character corresponds to $/ (usually returns \n). In paragraph mode, ($/ = '') chomp removes all newline characters from all incoming lines. It returns the number of chomped characters.

chop

```
chop string        chop array     chop
```

The chop function removes the last character of a string, of all elements of an array, or of $_. chop does not care what type of character is removed. It returns the character that has been deleted.

chown

```
chown uid, gid, filelist
```

The chown function changes the owner and group information of a list of files. The user and group have to be specified as IDs, not as names. To change only the user, gid has to be -1. It returns the number of changed files.

chr

```
chr number        chr
```

The chr function returns the character that is represented by number or $_ in the character set. For example, chr(65) returns A in ASCII.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

chroot

```
chroot dirname        chroot
```

The chroot function performs a chroot(2) system call to the specified directory (or to $_). This directory turns into the new root directory to which all absolute path names are related. Only the superuser can use chroot.

close

```
close [filehandle]
```

The close function closes a filehandle that was opened by open. If filehandle is omitted, the currently selected filehandle is closed.

closedir

```
closedir dirhandle
```

The closedir function closes a directory handle that was opened by opendir.

cos

```
cos value    cos
```

The cos function returns the cosine specified by value or the cosine of $_. value has to be indicated in radians.

crypt

```
crypt $plain, $salt
```

The crypt function uses the C library function crypt(3) to encrypt the string $plain irreversibly. crypt(3) is the standard way for encrypting passwords. You have to add a string $salt from which crypt takes only the two first characters into account. To verify if a password is correct, you write:

```
print "password ok\n" if crypt($plain, $passwd) eq $passwd;
```

where $plain is the guessed password in plain text and $passwd is the encrypted password to be checked.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

dbmclose

```
dbmclose %hash
```

The dbmclose function closes a DBM database file that was attached to a hash. It is deprecated. Perl instead calls the command untie.

dbmopen

```
dbmopen %hash, filename, mode
```

The dbmopen function opens the specified DBM file and binds it to a hash. filename has to be written without the ending .dir or .pag. If the file does not exist, it is created using the permissions given in mode. If you specify mode as undef, the file is not created, and the function returns a false value.

It is deprecated. Perl instead calls the command tie.

defined

```
defined expression
```

The defined function checks if expression is defined. Returns true if yes, or false if no. expression can be a scalar, an array, a hash, a value of a hash, or a subroutine. With subroutines, whole arrays, or hashes, defined checks whether the constructs exist. With scalars or hash values, defined checks whether they have a defined value.

delete

```
delete $hash{$key}        delete @hash{@key}
```

The delete function deletes the specified key/value pairs of a hash and returns them or undef if nothing is deleted.

Use undef to delete a complete hash and unlink to delete a file.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

die

```
die string       die list       die
```

Usually the `die` function prints the specified string or list (or the word "died" if no string or list is specified) to `STDERR`. Then, it terminates the program with the value stored in `$!`. If the given string or list does not end with a newline, `die` appends the name of the file, the current line number, the input line number, and a newline to that string or list.

If `die` is called within `eval`, it prints its message to `$@` and terminates `eval` with a value of `undef`. In this way, you can raise an exception without terminating your program.

dump

```
dump        dump label
```

The `dump` function causes an immediate core dump. When reincarnated (for example; with `undump(1)`), starts at the indicated label.

The idea is to speed up the code using a tool like `undump(1)`, which reads a dump and yields it as an executable program.

each

```
each %hash
```

The `each` function iterates through a hash. Every call of `each` returns another two-element list consisting of only one key and its value. When the hash pointer reaches the end of the hash, an empty list is returned so that loops can be abandoned. There is only one pointer per hash that `each`, `keys`, and `values` use together.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

eof

```
eof filehandle       eof       eof()
```

The `eof` function returns true if the filehandle has reached the end of the file or if the filehandle is not open. When `eof` is called without a filehandle, it checks the end-of-file for the file where the last read operation happened. Calling `eof()` with empty parentheses checks the end-of-file for the diamond operator `<>`. It returns true if the last file specified on the command line reaches its end.

eval

```
eval EXPR       eval BLOCK
```

The `eval` function traps fatal syntax or runtime errors. `EXPR` can be any usual Perl expression, such as `$a = $b / $c`. `eval` evaluates this expression and returns its result or the result of the last operation, if it has to execute a block. When a fatal error occurs during evaluation, `eval` returns `undef` without bringing Perl to termination. Therefore, you can use `eval` to stabilize parts of your program or to check for the availability of special features on the system on which your program runs.

exec

```
exec list
```

The `exec` function terminates the running Perl program and executes the system command given in `list` instead. If `exec` cannot start the new command, the running program is not terminated, and `exec` returns `undef`.

```
exec "cat filexy | sort | tail -20l" or die "Problems with
exec.\n";
```

exit

```
exit [value]
```

The `exit` function terminates the enclosing block, the current subroutine, or the running program and returns the specified value. Just before termination, the corresponding `END` block is executed. `value` defaults to 0.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

exists

```
exists $hash{key}
```

The exists function checks whether the specified key exists in the hash. It returns true if it exists, even if the corresponding value is undefined. Otherwise, it returns false.

exp

```
exp value       exp
```

The exp function returns e to the power of value or of $_.

fcntl

```
fcntl filehandle, function, $var
```

The fcntl function is the Perl interface to the C file control function fcntl(2) for manipulation of file descriptors.

fileno

```
fileno filehandle
```

The fileno function returns the file descriptor for the specified filehandle.

flock

```
flock filehandle, operation
```

The flock function is the interface to the C function flock(2). It uses an operating system-dependent routine for locking files.

Its operation can be LOCK_EX for exclusive lock, LOCK_SH for shared lock, LOCK_NB for non blocking lock, or LOCK_UN for removing locking.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

fork

```
fork
```

The fork function performs a fork(2) system call to create a child process. fork returns the process ID of the child to the parent and 0 to the child process. In the case of an error, it returns undef to the parent.

format

The format function declares a page format that can be used by write.

getc

```
getc filehandle       getc
```

The getc function reads a single character from the filehandle or STDIN if no filehandle is specified. It returns an empty string when eof is reached. To read in characters in an unbuffered manner from STDIN, for example to react to single key strokes, you must set your terminal to the correct mode. (Refer to the stty man page.)

getlogin

```
getlogin
```

The getlogin function returns the current login name (from the utmp file) or false on failure.

getpgrp

```
getpgrp pid       getpgrp 0       getpgrp
```

The getpgrp function returns the process group for the specified process or assumes the current process, if pid is 0 or omitted.

getppid

```
getppid
```

The getppid function returns the process ID of the parent process.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

getpriority

```
getpriority which, who
```

The `getpriority` function returns the priority for a process, process group, or user. `getpriority(0,0)` yields the priority of the current process. Refer to the `getpriority(2)` man page for details.

glob

```
glob pattern
```

The `glob` function returns the result of a file name expansion on the specified pattern. It leads to the same result as `pattern`.

gmtime

```
gmtime expr       gmtime
```

The `gmtime` function accepts a time expression in the format of `time` (the number of seconds since 1.1.1970 in GMT) and returns a string like "`Fri Jul 14 10:51:54 2000`" in scalar context. In an list context, `gmtime` returns a list of nine values (`$sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst`), where the individual fields mean seconds, minutes, hour, day of the month, year, day of the week, day of the year, and daylight saving time (yes or no). If the time expression is omitted, `gmtime` returns the current time given by `time`. The time is always interpreted and returned for the GMT.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

goto

```
goto LABEL        goto &subname
```

The `goto LABEL` function jumps to the statement marked with `LABEL` and resumes execution there. You cannot jump into constructs that require initialization, like subroutines or `foreach` loops. You can jump out of those constructs.

`goto &subname` not only invokes `&subname` but replaces the actual subroutine call by the desired one, so it looks like the new subroutine had been called from the main program.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

grep

```
grep expr, @array       grep block @array
```

The grep function evaluates expr or block for every element of the array. $_ is set to the respective array element, by which the elements can be modified. In list context, grep returns the list of elements for which the result of expr is true. In scalar context, only the number of those elements is returned.

hex

```
hex value       hex
```

The hex function interprets a given value or $_ as a hexadecimal string and returns the equivalent decimal value. For example hex('ff') returns 255.

import

```
import module list
```

If defined within a package, use invokes the import function when loading the package. It imports a list of items from the specified module.

index

```
index string, substr, [offset]
```

The index function returns the position of the first occurrence of a substring within a string. Positions are counted from 0. If offset is specified, searching starts at that position. When the substring is not found, index returns -1.

int

```
int value       int
```

The int function returns the integer part of the value or of $_.

ioctl

```
ioctl filehandle, function, $var
```

The ioctl function is the Perl interface to the ioctl(2) C function.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

join

```
join expr, @array
```

The join function joins all elements of the array into a single string in which the fields are separated by the specified expr (character or string). It returns the resulting string.

keys

```
keys %hash
```

The keys function returns a list of all the keys of the hash when evaluated in list context. The order of the keys is the same as with for each and values and depends on memory position of the values. In scalar context, the number of keys is returned.

kill

```
kill signal, pidlist
```

The kill function sends a signal to a list of processes given by pidlist. You may specify the signal by its quoted name or by number (without minus sign). Use the UNIX command kill -l for a list of available signals. If signal is specified with a leading minus, a whole process group is signalled. It returns the number of processes signalled.

lc

```
lc string       lc
```

The lc function returns string or $_ in lowercase.

lcfirst

```
lcfirst string       lcfirst
```

The lcfirst function returns string or $_ with only the first letter in lowercase.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

length

```
length string       length
```

The `length` function returns the length of `string` or `$_` in bytes.

link

```
link oldfile, newfile
```

The `link` function creates a hard link from `oldfile` to `newfile`. It returns 1 if successful or 0 otherwise.

local

```
local variable       local(var1, var2,..)
```

The `local` function creates a variable that has a local value in the current block or subroutine but is also visible in the rest of the program (unlike variables created with `my`). Because of the high precedence of `local`, enclose multiple arguments in parentheses.

localtime

```
localtime expr       localtime
```

The `localtime` function works exactly like `gmtime`, but the time for the local time zone is returned.

log

```
log value       log
```

The `log` function returns the natural logarithm of `value` or of `$_`.

lstat

```
lstat filename
```

The `lstat` function acts exactly like `stat`, but if the file name is a symbolic link, `lstat` reads the information for the link itself, instead of the file to which the link points.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

map

```
map expr, @array        map block @array
```

The map function evaluates expr or block for every element of the array. $_ is set to the respective array element by which the elements can directly be modified. It returns the list of all results.

mkdir

```
mkdir dirname, perm
```

The mkdir function creates a new directory with the specified name and permissions. You must specify the permissions in four-digit numeric mode. It returns 1 if successful and 0 otherwise.

my

```
my variable        my(var1, var2,..)
```

The my function creates a private variable that exists only in the respective block or subroutine. Because of the high precedence of my, enclose multiple arguments in parentheses.

no

```
no module
```

The no function is the opposite of use. It unimports everything that was imported by use.

oct

```
oct value        oct
```

The oct function interprets a given value (or $_) as an octal string, or if starting with 0x, as a hexadecimal string and returns the equivalent decimal value. To convert from decimal to octal, use sprintf "%lo", $decimal.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@carlsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

open

```
open filehandle, filename
```

The open function opens the specified file and associates it with a filehandle. It returns true if the operation is successful and undef if not. You can prefix the file name by special symbols with the meanings described in Table F-12:

Table F-12 open Options

File Name	Meaning
file	Opens the file for input
<file	Opens the file for input as well
>file	Opens the file for output. Creates the file if the file does not exist. Overwrites the file if it does.
>>file	Opens the file to append data. Creates the file if the file does not exist.
+<file	Opens the file for read and write
+>file	Opens the file for read and write
+>>file	Opens the file for read and append
| cmd	Opens a pipe to a command
cmd |	Opens a pipe from command

László Kun (laszlo.kun@oracle.com) has a non-transferable license to use this Student Guide.

Note – The - as a file name stands for STDIN, and >- stands for STDOUT.

opendir

```
opendir dirhandle, dirname
```

The opendir function opens the specified directory and associates it with a directory handle. It returns true if successful or false otherwise. When a directory is opened by opendir, it can be read by readdir.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

ord

```
ord char       ord string       ord
```

The ord function takes a given character or the first character of a given string (or of $_) and returns its numeric ASCII value. For example, ord('A') returns 65.

pack

```
pack template, valuelist
```

The pack function takes a list of values and converts it into a binary structure using the specified template. It returns the resulting string. The template consists of a sequence of characters that indicate the type of the respective elements of the list. Each character can be followed by a number that gives its repeat count. Table F-13 describes the template characters.

Table F-13 Template Characters for pack

Character	Meaning
a / A	Null-padded/space-padded ASCII string
b / B	Bit string in low-to-high/high-to-low order
c / C	Signed/unsigned character value
d / f	Double/single float in native format
h / H	Hexadecimal string, low/high nibble first
i / I	Signed/unsigned integer
l / L	Signed/unsigned long value
n / N	Short/long in (big endian) network-byte order
p / P	Pointer to a string/structure (fixed-length string)
s / S	Signed/unsigned short value
u	Uuencoded string
v / V	Short/long in VAX (little endian) order
w	BER compressed integer
x	Null byte
X	Back up a byte

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kulik (laszlo.kulik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table F-13 Template Characters for `pack` (Continued)

Character	Meaning
`@`	Null fill to absolute position

Refer to the man page `perlfunc` for a detailed explanation of `pack`.

package

```
package namespace
```

The `package` function indicates that all names that appear up to the end of the enclosing block use the specified namespace. `package` is usually used at the beginning of a module to declare the namespace for the module. If names of another namespace must be used, the namespace is explicitly placed in front of the variable name, as in `$namespace::variable`.

pipe

```
pipe readhandle, writehandle
```

The `pipe` function is the interface to the system call `pipe(2)`. It opens and returns a pair of connected handles.

pop

```
pop @array        pop
```

The `pop` function deletes the last element of an array (only of an array not of a list) and returns it. The length of the array is reduced by 1. If `@array` is omitted, `pop` shortens `@ARGV` in the main program and `@_` in a subroutine.

print

```
print       print list       print filehandle list
```

The `print` function prints a string or a list of strings to the specified filehandle. If `filehandle` is omitted, `print` prints to `STDOUT`. If no argument is given, `$_` is printed. It returns 1 if successful or 0 otherwise.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

printf

```
printf filehandle format, list       printf format, list
```

The printf function prints a formatted string or list of strings to a filehandle or to STDOUT.

push

```
push @array, list
```

The push function pushes a given list of values onto the end of an array. It returns the new length of the array.

quotemeta

```
quotemeta string
```

The quotemeta function returns the string with backslashes before every metacharacter (non-alphanumeric characters). This function is appropriate for preparing complicated strings that should be used literally in regular expressions.

rand

```
rand value       rand
```

The rand function determines a random fractional number between 0 (inclusive) and value (exclusive) and returns it. If value is omitted, the upper border is set to 1. To get really random numbers, you have to use srand as well.

read

```
read filehandle, $buf, length [, offset]
```

The read function reads length bytes from filehandle and puts them into the variable $buf (at position offset, if specified). It returns the number of bytes read if successful or returns 0 at eof or undef if an error occurs. Use this function when reading from direct-access databases. The position within the file where reading starts can be set initially by seek. To write data to a determined position within a file, use print.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

readdir

```
readdir, dirhandle
```

The readdir function reads entries of a directory. In scalar context, the next entry is returned (or undef, if all are read). In list context, it returns a list of all remaining entries (or an empty list, if nothing is left). The directory must be opened previously by opendir.

readlink

```
readlink linkname       readlink
```

The readlink function returns the name of the file pointed to by the specified link (linkname or $_). It returns undef if not successful.

ref

```
ref $reference       ref
```

The ref function returns the type of variable that the specified reference ($reference or $_) points to. The values returned might be of type REF, SCALAR, ARRAY, HASH, CODE or GLOB. If the referenced object has been attached to a package by bless, the package name is returned instead.

rename

```
rename oldname, newname
```

The rename function renames the file specified by oldname. It returns 1 if successful or 0 otherwise.

require

```
require module       require version
```

The require function is a conservative method of loading a Perl module. It checks all directories of @INC for a file that has the same name as module (and the suffix .pm if module is specified as bareword). When the appropriate file is

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@atlesson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

found, Perl inserts its code into the current program or module and executes it. To import variable and function names, you have explicitly to call `import`, because `require` does not automatically import.

If the argument is a numeric value, `require` checks the current version of Perl against the specified argument. It immediately terminates the program, if the required version is newer than the current one.

reset

```
reset expr
```

The `reset` function resets all variables (scalars, arrays, and hashes) to their initial values. Not the name of a variable, `expr` is a list of letters. All variables inside the current block or subroutine that start with such letters are reset.

return

```
return value        return list      return
```

The `return` function terminates the actual subroutine and returns the specified value or list. If no values are specified, it returns `undef` in a scalar context and an empty list in list context.

reverse

```
reverse @array        reverse string
```

In list context, the `reverse` function returns the given list in reverse order. In scalar context, it concatenates all elements of the array and returns the resulting string in reverse order. If a string is given as argument, there is nothing to concatenate and the reversed string is returned.

rewinddir

```
rewinddir dirhandle
```

The `rewinddir` function sets the directory pointer to the beginning of the directory, so that the next call to `readdir` yields the first entry.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

rindex

```
rindex string, substr, [offset]
```

The `rindex` function returns the position of the last occurrence of a substring within a string. Positions are counted from 0. If `offset` is specified, a search is performed from 0 to that position. If the substring is not found, `rindex` returns -1.

rmdir

```
rmdir dirname        rmdir
```

The `rmdir` function deletes the specified directory (or `$_`) if it is empty. It returns 1 if successful; otherwise, it returns 0.

scalar

```
scalar expr
```

The `scalar` function forces an expression to be evaluated in scalar context. So `scalar @array` returns the list of elements of the array.

seek

```
seek filehandle, position, start
```

The `seek` function sets the file pointer of `filehandle` to a new position. The position is counted in bytes (starting at 0). The argument `start` determines from which point the parameter `position` is counted: 0 indicates the beginning, 1 the actual position, and 2 the end of the file. If 2 is specified, `position` is counted back from the end. It returns 1 if successful or 0 otherwise.

seekdir

```
seekdir dirhandle, position
```

The `seekdir` function sets the pointer of a directory handle to a new position.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

select

```
select filehandle       select
```

The `select` function returns the currently selected output filehandle. If a filehandle is specified, the default output filehandle is changed to it. For example, `print` sends its output to that filehandle instead of `STDOUT` when called without a filehandle argument.

select

```
select rbits, wbits, ebits, timeout
```

This is a completely different function than the previously mentioned one and has the same name only for historical reasons. This function is an interface to the `select(2)` system call, which can examine file descriptors.

setpgrp

```
setpgrp pid, pgrp
```

The `setpgrp` function sets the process group for the process `pid` to `pgrp`. A `pid` of 0 addresses the current process.

setpriority

```
setpriority which, who, priority
```

The `setpriority` function sets the priority for a process, a process group, or a user.

shift

```
shift @array        shift
```

The `shift` function deletes the first element of an array (only of an array, not of a list) and returns it. The length of the array is reduced by one and all elements are moved left. If `@array` is omitted, `shift` shortens `@ARGV` in the main program and `@_` in a subroutine.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

sin

```
sin value     sin
```

The `sin` function returns the sine of the specified value or of `$_`. Indicate the `value` variable in radians.

sleep

```
sleep seconds     sleep
```

The `sleep` function causes the program to sleep the specified amount of seconds or forever, if nothing is specified. It returns the number of seconds already slept.

sort

```
sort @array sort block @array sort subname @array
```

The `sort` function returns the sorted list. By default, the elements of the array are sorted alphabetically. To sort in numeric or any other order, respective comparison commands have to be specified in a block or in a subroutine. The `sort` function iterates through the array and repeatedly assigns two values to the block or subroutine that have to be compared. The comparison must return a value less than, equal to, or greater than zero. Two special variables are used for the comparison, `$a` and `$b`. You can use two special comparison operators with `$a` and `$b`: `cmp` compares two values alphabetically, and `<=>` compares them numerically.

splice

```
splice @array, startpos [, length] [, replacelist]
```

The `splice` function removes `length` elements of the array (only of an array, not of a list) starting at `startpos` and possibly replaces them with the specified list. The removed sublist is returned. If `length` is omitted, all elements from `startpos` through the last element are removed and replaced. If length is negative, the end position is counted from the end of the array.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@oracle.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

split

```
split /pattern/, string [, limit]
split /pattern/ @array
split
```

The split function splits a string into separated substrings by scanning it for /pattern/ as delimiters. /pattern/ has to be a regular expression. In list context, split returns the substrings without delimiters as a list. In scalar context, it returns the number of split substrings. If limit is specified, that number of fields are split. If string is omitted, it splits $_. Without any arguments, split uses $_ as the string and white space as the delimiter (as in /\s+/) but skips all leading white space.

sprintf

```
sprintf format, list
```

The sprintf function takes a list of scalars and returns them to a variable as a formatted string.

sqrt

```
sqrt value        sqrt
```

The sqrt function returns the square root of value or of $_.

srand

```
srand value        srand
```

The srand function initializes the starting point for the rand operator with value. If value is omitted, srand(time) is performed instead. Without calling srand, rand always returns the same list of random numbers.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

stat

```
stat filename       stat filehandle       stat _
```

The stat function reads out the inode information of a file. Returns the following list of 13 elements:

```
( $dev, # device number
$ino,  # inode number
$mode, # file mode
$nlink, # link count
$uid,  # numeric user ID
$gid,  # numeric group ID
$rdev, # special device identifier
$size, # filesize in bytes
$atime,# time last accessed
$mtime,# time last modified
$ctime,# time inode was last changed
$blksize,# blocksize
$blocks )# number of allocated blocks
```

The argument can be a file name or a filehandle. If _ is specified as argument, stat uses information gathered with the last call of stat or the last file test operation. If not successful, it returns an empty list.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

substr

```
substr string, startpos, [, len]
```

In its first form (`$sub = substr string,startpos,len`) the `substr` function returns the part of a string that starts at position `startpos` and has a length of `len` bytes. If `len` is omitted, the substring is taken up to the end of the string. If `len` is negative, the substring ends at `abs(len)` characters before the end of the string. If `startpos` is negative, the start position is counted from the end of the string.

In its second form (`substr(string,startpos,len)=$replace`), the specified substring is replaced by another string.

symlink

```
symlink oldname, newname
```

The `symlink` function creates a new file name that is symbolically linked to the old file. It returns 1 if successful, or 0 otherwise.

syscall

```
syscall command, arglist
```

The `syscall` function executes the system call specified in `command`. The argument list is `arglist` for the system call to which numbers are handed over as C integers and to which other values are handed over as pointers to strings.

sysopen

```
sysopen filehandle, filename, mode [, perms]
```

To bypass the `stdio` C library, on which the Perl standard I/O functions are based (to avoid buffering, for example), you can use system calls directly. The `sysopen` function is an interface to the `open(2)` system call. It opens the specified file name in the given mode and attaches it to a filehandle. If the file does not exist, it can be created with the permissions indicated in `perms`. (Use four-digit numeric format.) The possible values for `mode` are given in the standard module `Fcntl`; for example: 0 for read-only, 1 for write-only, and 2 for read-write.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

sysread

```
sysread filehandle, $buf, length [, offset]
```

The sysread function is the interface to the `read(2)` system call. Reads `length` bytes from `filehandle` and puts them into the variable `$buf` (at position `offset`, if specified). It returns the number of bytes read if successful, returns 0 if `eof`, and returns `undef` if an error occurred. The position within the file where reading starts can be set by `seek` initially.

system

```
system command, arglist
```

The `system` function executes the specified `command` (to which `arglist` is given as an argument list) without terminating it. It waits for the program to terminate and returns the exit status of the executed program. Because `SIGINT` and `SIGQUIT` are passed to the child process, not to your Perl program, the called `command` is interrupted with Control-C

syswrite

```
syswrite filehandle, $buf, length [, offset]
```

The `syswrite` function is the Interface to the `write(2)` system call. It writes `length` bytes from `$buf` (from position `offset`, if specified) to `filehandle`. It returns the number of bytes written or `undef` if an error occurred. The position within the file where writing starts can be set by `seek` initially.

tell

```
tell filehandle       tell
```

The `tell` function returns the current position of the specified filehandle or of the file last read.

telldir

```
telldir dirhandle
```

The `telldir` function returns the current position of the directory pointer. Returned values can be used by the `seekdir` function.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

tie

```
tie variable, classname [, list]
```

The `tie` function binds a variable (which can be a scalar, an array, or a hash) to a package class. Any further arguments are passed to the `new` method of the class. The best known example for `tie` is the `dbmopen` command, which attaches a hash to the module `AnyDBM_File`.

tied

```
tied variable
```

The `tied` function returns the object to which the variable is tied, which is the same as was returned by the `tie` function. It returns `undef` if the variable is not tied at all.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

time

```
time
```

The `time` function returns the number of seconds since January 1, 1970 in Greenwich Mean Time (GMT).

times

```
times
```

The `times` function measures the time a process consumed and returns a list of elements: (`$user, $system, $cuser, $csystem`). `$user` and `$system` give the time for the actual process in seconds, and `$cuser` and `$csystem` give the time for its child processes in seconds.

truncate

```
truncate filename, size       truncate filehandle, size
```

The `truncate` function truncates a file, specified by `filename` or `filehandle`, to `size` bytes. Returns `undef` if not successful.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

uc

```
uc string       uc
```

The uc function returns string or $_ in uppercase.

ucfirst

```
ucfirst string       ucfirst
```

The ucfirst function returns string or $_ with only the first letter in uppercase.

umask

```
umask value       umask
```

The umask function without an argument gets the current umask. The umask function with an argument sets a new umask to value and returns the old umask.

undef

```
undef $scalar      undef %array       undef %hash
undef &subroutine      $x = undef
```

The undef function removes a scalar, a whole array, a hash, or a subroutine from memory. Storage is deblocked.

unlink

```
unlink filelist       unlink
```

The unlink function removes a list of files or the file stored in $_. It returns the number of files successfully deleted.

unpack

```
unpack template, string
```

The unpack function interprets string as a packed data structure (as returned by pack) and converts it into a list of separate values using template. It is usually the same template used with pack can be used.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

unshift

```
unshift @array, list
```

The unshift function pushes a given list to the front of an array (only of an array, not of a list) and returns the number of elements in the new array. All previous elements are moved right.

untie

```
untie variable
```

The untie function breaks the binding of a variable, which was established with tie.

use

```
use module       use module()      use module symbollist
```

The use function looks in all directories of @INC for a file that has the same name as the specified module (and the suffix .pm if module is specified as a bareword). If the appropriate file is found, Perl inserts its code into the current program or module.

In its first form, use module, Perl automatically imports all variable and function names from that module into the actual namespace. In the form use module symbollist, only the names in symbollist are imported. With use module(), nothing is imported. Because use is executed at compile time, the statement use module list; is exactly equivalent to the following:

```
BEGIN { require module; import module list }.
# BEGIN causes the commands to be executed at compile
# time
```

utime

```
utime atime, mtime, filelist
```

The utime function changes time stamps for a list of files. Access time and modification time are set to the values given in atime and mtime. The inode change time is set to the current time. All specified time values have to be in the format returned by time or stat.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@arhs-es.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

values

```
values %hash
```

The values function returns a list of all the values of the hash when evaluated in list context. The order of the values is the same as for `each` and `keys` and depends on the memory position of the values. In scalar context, the number of values is returned.

vec

```
vec string, offset, bits
```

The `vec` function interprets `string` as a vector of unsigned integers and returns the value stored at position `offset` (counted in bits). The argument `bits` indicates the width of each value. If the values are only 0 and 1, only one bit is needed. If the values are from 0 to 3, two bits are needed, and so on. In this way a very efficient storage of small integers is obtained. `vec($string, 0,1)` function returns the first bit of the first byte of string. Values can also be assigned to `vec`.

wait

```
wait
```

The `wait` function waits for a child process to terminate. When successful, it returns the process ID of the former child; otherwise, it returns -1.

waitpid

```
waitpid pid, flags
```

The `waitpid` function waits for the child process (specified by `pid`) to terminate and returns its process ID. It returns -1 if `pid` is not a child or `pid` does not exist. Refer to the man page `waitpid(2)` for details about available flags.

warn

```
warn string       warn list       warn
```

The `warn` function prints the specified string, list, or the default "`Warning: something's wrong`" to `STDERR`. The specified message is treated in exactly the same way as with `die`. Unlike `die`, `warn` does not terminate the program.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

write

```
write filehandle       write
```

The `write` function writes to the specified filehandle or, if `filehandle` is omitted, to the currently selected output filehandle using the format which is attached to this filehandle. The data to be to written is specified in the format itself.

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Appendix G

Perl Modules

Standard Modules

Standard modules are available in every Perl distribution. Some of these modules are described in Table G-1 through Table G-12.

Table G-1 Pragma Modules

Module	Description
`attrs`	Gets or sets attributes of a subroutine
`autouse`	Defers loading modules until they are needed
`base`	Creates an "is-a" relationship to base classes at compile time
`blib`	Uses MakeMaker's uninstalled version of a package
`constant`	Declares constants
`diagnostics`	Forces verbose warning diagnostics
`fields`	Declares class fields at compile time
`integer`	Uses integer arithmetic instead of double
`less`	Requests less of something from the compiler (not implemented)
`lib`	Manipulates `@INC` at compile time
`locale`	Uses or ignores current locale settings
`ops`	Restricts unsafe opcodes when compiling
`overload`	Overloads basic Perl operations
`re`	Alters behavior of regular expressions

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kutik (laszlo.kutik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-1 Pragma Modules (Continued)

Module	Description
`sigtrap`	Enables simple signal handling
`strict`	Restricts unsafe Perl constructions
`subs`	Predeclares names of subroutines
`utf8`	Turns on and off UTF-8 and unicode support
`vmsish`	Specifies VMS specific features
`vars`	Predeclares global variable names
`warning`	Turns on or off several warnings

Table G-2 Miscellaneous Modules

Module	Description
`Benchmark`	Provides benchmarks running times of Perl code
`CPAN`	Provides an interface for loading CPAN modules
`CPAN::FirstTime`	Creates a CPAN configuration file
`CPAN::Nox`	Uses CPAN without compiled extensions
`Carp`	Warns of errors
`Config`	Accesses Perl configuration information
`English`	Uses long, understandable names for special variables
`Env`	Imports environment variables
`Fatal`	Makes all errors in built-ins fatal
`Getopt::Long`	Extended processing of command-line options
`Getopt::Std`	Processes single-character switches with switch clustering
`I18N::Collate`	Compares data according to locale settings
`Shell`	Executes shell commands transparently
`Symbol`	Manipulates Perl symbols and names
`Sys::Syslog`	Uses UNIX `syslog(3)` calls

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kun (laszlo.kun@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-3 Time and Locale Modules

Module	Description
`Time::Local`	Calculates `time` from local and GMT time
`Time::gmtime`	Calls Perl's built-in function `gmtime` by name
`Time::localtime`	Calls Perl's built-in function `localtime` by name
`Time::tm`	Is an internal object used by `Time::*` modules; do not use directly

Table G-4 File Access and I/O Modules

Module	Description
`Cwd`	Gets the current working directory
`DirHandle`	Provides object-oriented (OO) methods for directory handles
`FileCache`	Keeps more files open than the system permits
`FileHandle`	Provides OO methods for filehandles
`File::Basename`	Parses pathnames for basename and file name
`File::CheckTree`	Tests files on a whole directory tree
`File::Compare`	Compares files or filehandles
`File::Copy`	Copies files or filehandles
`File::DosGlob`	Provides DOS-like file globbing
`File::Find`	Traverses a complete file tree
`File::Path`	Creates or deletes one or more directories
`File::Spec`	Performs portable operations on filenames
`File::Spec::Mac`	Provides `File::Spec` methods for MacIntosh; exists also for OS/2, UNIX, VMS, and Microsoft Windows 32
`File::stat`	Calls Perl's built-in function `stat` by name
`FindBin`	Looks for the directory of the original Perl script
`IO`	Loads several other I/O modules

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-4 File Access and I/O Modules (Continued)

Module	Description
`IO::File`	Provides OO methods for filehandles
`IO::Handle`	Provides OO methods for I/O handles
`IO::Pipe`	Provides OO methods for pipes
`IO::Seekable`	Provides OO methods for seek-based I/O operations
`IO::Select`	Provides OO interface to the `select` system call
`IO::Socket`	Provides OO interface to socket communications
`SelectSaver`	Saves and restores a selected filehandle

Table G-5 Text Processing Modules

Module	Description
`Data::Dumper`	Converts data structures to strings
`Pod::Html`	Converts POD files to HTML
`Pod::Functions`	Lists available Perl functions
`Pod::Text`	Converts POD files to ASCII text
`Search::Dict`	Looks for a key in a dictionary file
`Term::Cap`	Accesses `termcap` database
`Term::Complete`	Provides automatic completion of words
`Term::ReadLine`	Accesses `readline(3)` libraries
`Text::Abbrev`	Creates an abbreviation table from a word list
`Text::ParseWords`	Parses text into an array of tokens
`Text::Soundex`	Converts a word to soundex code
`Text::Tabs`	Expands and unexpands tabs
`Text::Wrap`	Provides line wrapping of long lines

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-6 DBM Modules

Module	Description
`AnyDBM_File`	Provides framework for multiple DBMs
`DB_File`	AccessesBerkeley DB files
`GDBM_File`	Accesses GDBM files
`NDBM_File`	Accesses NDBM files
`ODMB_File`	Accesses ODBM files
`SDBM_File`	Accesses SDBM files

Table G-7 Math Modules

Module	Description
`Math::BigFloat`	Arbitrary-length float math package
`Math::BigInt`	Arbitrary-size integer math package
`Math::Complex`	Complex numbers math package
`Math::Trig`	Trigonometric functions
`POSIX`	Interface to IEEE Standard 1003.1 with a lot of mathematical functions

Table G-8 Networking and IPC Modules

Module	Description
`IPC::Msg`	Provides the UNIX System V message queues
`IPC::Semaphore`	Provides the UNIX System V semaphores
`IPC::SysV`	Provides the UNIX System V object class
`IPC::Open2`	Opens a process for both reading and writing
`IPC::Open3`	Opens a process for reading, writing, and error handling
`Net::Ping`	Checks whether a host is reachable

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kutik (laszlo.kutik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-8 Networking and IPC Modules (Continued)

Module	Description
`Net::hostent`	Calls Perl's built-in `gethost*` functions by name
`Net::netent`	Calls Perl's built-in `getnet*` functions by name
`Net::protoent`	Calls Perl's built-in `getprotoent*` functions by name
`Net::servent`	Calls Perl's built-in `getserv*` functions by name
`Socket`	Provides C `socket.h` functions and structure manipulators
`Sys::Hostname`	Gets the host name in every conceivable way
`User::grent`	Calls Perl's built-in `getgr*` functions by name
`User::pwent`	Calls Perl's built-in `getpw*` functions by name

Table G-9 CGI Modules

Module	Description
`CGI`	Many CGI methods
`CGI::Apache`	CGI addition for Apache HTTP server
`CGI::Carp`	Treatment of error messages
`CGI::Cookie`	Use Netscape cookies
`CGI::Fast`	Interface to FastCGI
`CGI::Push`	Interface to Server Push
`CGI::Switch`	Load several CGI modules

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-10 Object-Oriented Programming Modules

Module	Description
`Class::Struct`	Declares struct-like datatypes as classes
`Exporter`	Default import method for modules
`Tie::Array`	Base class for tied arrays
`Tie::Hash`	Base class for tied hashes
`Tie::Handle`	Base class for tied handles
`Tie::RefHash`	Base class for tied hashes with references as keys
`Tie::Scalar`	Base class for tied scalars
`Tie::StdArray`	Basic methods for tied arrays
`Tie::StdHash`	Basic methods for tied hashes
`Tie::StdScalar`	Basic methods for tied scalars
`Tie::SubstrHash`	Fixed-table-size, fixed-key-length hashing
`UNIVERSAL`	Base class for ALL classes

Table G-11 Dynamic Function Loading Modules

Module	Description
`AutoLoader`	Loads functions only on demand
`AutoSplit`	Splits a package for autoloading
`Devel::SelfStubber`	Provides a stub generator for SelfLoading modules
`DynaLoader`	Provides dynamic loading of C libraries into Perl
`SelfLoader`	Loads functions only on demand

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-12 Extensions and Development Support Modules

Module	Description
`B*`	Experimental packages (byte code creation, Perl-to-C translater)
`ExtUtils::Command`	Replaces common UNIX commands in Makefiles
`ExtUtils::Embed`	Embedding Perl in C or C++ applications
`ExtUtils::Install`	Installation of files
`ExtUtils::Installed`	Management of installed modules
`ExtUtils::Liblist`	Determines which libraries to use and how to use them
`ExtUtils::MM_OS2`	OS/2 methods for `ExtUtils::MakeMaker`
`ExtUtils::MM_Unix`	UNIX methods for `ExtUtils::MakeMaker`
`ExtUtils::MM_VMS`	VMS methods for `ExtUtils::MakeMaker`
`ExtUtils::MM_Win32`	Win32 methods for `ExtUtils::MakeMaker`
`ExtUtils::MakeMaker`	Create a Makefile
`ExtUtils::Manifest`	Write and check MANIFEST files
`ExtUtils::Miniperl`	Writes the C code for `perlmain.c`
`ExtUtils::Mkbootstrap`	Creates bootstrap file for the DynaLoader
`ExtUtils::Mksymlists`	Creates linker options files for dynamic extension
`ExtUtils::Packlist`	Manages `.packlist` files
`ExtUtils::testlib`	Adds blib directories to `@INC`
`Fcntl`	Loads the C `Fcntl.h` definitions
`Opcode`	Disables named opcodes when compiling Perl code
`POSIX`	Interfaces to IEEE Standard 1003.1
`Safe`	Compiles and executes code in restricted compartments
`Test::Harness`	Runs Perl standard test scripts with statistics

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kvik (laszlo.kvik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

CPAN Modules

In addition to the Perl standard modules, there are 1500 Comprehensive Perl Archive Network (CPAN) modules. These freely available modules reflect the work and ideas of many Perl programmers. The archive is available on the Web at `http://www.cpan.org/`.

Due to the large size of the collection, only the list of categories is shown in Table G-13.

Table G-13 CPAN Module Categories

Perl Core Modules
Development Support
Operating System Interfaces
Networking Devices, IPC
Data Type Utilities
Database Interfaces
User Interfaces
Language Interfaces
File Names, File Systems, and File Locking
String Processing and Language Text Processing
Option, Argument, Parameter, and Configuration File Processing
Internationalization and Localization
Security and Encryption
World Wide Web, HTML, HTTP, and CGI
Server and Daemon Utilities
Archiving and Compression
Images, Pixmaps, and Bitmaps
Mail and Usenet News
Control Flow Utilities
Filehandle and Input/Output Stream Utilities
Microsoft Windows Modules

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C

Table G-13 CPAN Module Categories (Continued)

Miscellaneous Modules
Commercial Software Interfaces

Unauthorized reproduction or distribution prohibited. Copyright© 2011, Oracle and/or its affiliates.

László Kuik (laszlo.kuik@ericsson.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision C