Server-Side Programs and Perl 5

Outline

1 Server-Side Includes (SSI)
2 Common Gateway Interface (CGI)
3 Introduction to Perl
String Processing and Regular Expressions
4 Viewing Client/Server Environment Variables
5 Form Processing and Business Logic
6 Verifying a Username and Password
Code
7 Cookies and Perl

Based on material © 2000 Deitel & Associates, Inc.

1 Server-Side Includes

- Web offers ability to track
  - Where client coming from
  - What client views on your site
  - Where client goes after your site
- Tracking Web data important, allows webmasters to
  - Know which sites visited most frequently
  - Know how effective advertisements and products are
- *Server-side includes* (SSIs)
  - Commands embedded in HTML documents
  - Provide for content creation
  - Allow inclusion of current time, date or even contents of different HTML documents

Based on material © 2000 Deitel & Associates, Inc.

1 Server-Side Includes (II)

- *SSI commands*
  - Execute CGI scripts on a server
  - Are capable of connecting to an ODBC data source
    - Use to create customized Web pages depending for certain conditions
  - Document containing SSI commands has **`.shtml`** file extension
- ***`EXEC CGI`*** *command*
  - Issued to execute a Perl script before document sent to client

  Example: **`<!-- #EXEC CGI="cgi-bin/counter.pl" -->`**
  - Executes the Perl script **`counter.pl`**, located in **`/cgi-bin`** directory on server

Based on material © 2000 Deitel & Associates, Inc.

1 Server-Side Includes (III)

- ***ECHO*** *command*
  - Used to display variable information
  - Is followed by the *keyword* ***VAR*** and variable's constant name
    Example: `<!-- #ECHO VAR="DATE_LOCAL" -->`
  - Returns the current local time
- Other variables
  - ***DATE_GMT***
    - Contains current Greenwich Mean Time
  - ***DOCUMENT_NAME***
    - Contains name of current document
  - Many more ← Apache Tutorial

Based on material © 2000 Deitel & Associates, Inc.

1 Server-Side Includes (III)

- ***EXEC CGI*** *command*
  - Used to include CGI program output
  - Example follows
- To see what our servers (at Dal FCS) do see
  - `examples/SSI/test1.shtml` (what the client gets)
  - `examples/SSI/test1.source` (code at the server)

Based on material © 2000 Deitel & Associates, Inc.

```
1<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2<!-- Fig. 22 counter.shtml -->
3
4<HTML>
5   <HEAD>
6      <TITLE>Using Server Side Includes</TITLE>
7   </HEAD>
8
9<BODY>
10   <CENTER>
11      <H3> Using Server Side Includes</H3>
12   </CENTER>
13
14   <!-- #EXEC CGI="/cgi-bin/counter.pl" --><BR>
15   The Greenwich Mean Date is
16   <FONT COLOR = BLUE>
17
18   <!-- #ECHO VAR="DATE_GMT" -->.
19   </FONT><BR>
20   The name of this document is
21   <FONT COLOR = BLUE>
22
23   <!-- #ECHO VAR="DOCUMENT_NAME" -->
24   </FONT><BR>
25   The local date is
26   <FONT COLOR = BLUE>
27
28   <!-- #ECHO VAR="DATE_LOCAL" -->
29   </FONT><BR>
30   This document was last modified on
31   <FONT COLOR = BLUE>
32
```

Outline

14 Execute Perl script `counter.pl` using `EXEC CGI` statement

18 Use `ECHO VAR` statements to display environmental variables

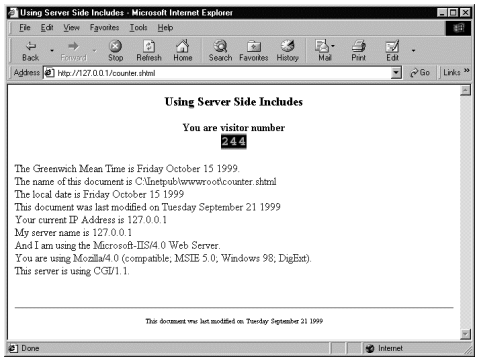
```
33   <!-- #ECHO VAR="LAST_MODIFIED" -->
34   </FONT><BR>
35   Your current IP Address is
36   <FONT COLOR = BLUE>
37
38   <!-- #ECHO VAR="REMOTE ADDR" -->
39   </FONT><BR>
40   My server name is
41   <FONT COLOR = BLUE>
42
43   <!-- #ECHO VAR="SERVER_NAME" -->
44   </FONT><BR>
45   And I am using the
46   <FONT COLOR = BLUE>
47
48   <!-- #ECHO VAR="SERVER SOFTWARE" -->
49   Web Server.</FONT><BR>
50   You are using
51   <FONT COLOR = BLUE>
52
53   <!-- #ECHO VAR="HTTP_USER_AGENT" -->.
54   </FONT><BR>
55   This server is using <FONT COLOR = BLUE>
56
57   <!-- #ECHO VAR="GATEWAY INTERFACE" -->.
58   </FONT><BR>
59   <BR><BR>
60   <CENTER>
61   <HR>
62   <FONT SIZE = -5>This document was last modified on
63
64   <!-- #ECHO VAR="LAST MODIFIED" --></FONT>
65
66   </CENTER>
67</BODY>
68</HTML>
```

Outline

Continue printing environmental variables using `ECHO VAR` statements

8

Script Output



Based on material © 2000 Deitel & Associates, Inc.

```
1  # Counter.pl
2  # Program to track the number of times a web page
3  # has been accessed.
4
5  open(COUNTREAD, "counter.dat"); # should have an error check
6     my $data = <COUNTREAD>; # read 1 line from file
7     $data++;
8  close(COUNTREAD);
9
10 open(COUNTWRITE, ">counter.dat"); # should have an error check
11    print COUNTWRITE $data;
12 close(COUNTWRITE);
13
14 print "<div style='text-align: center'>";
15 print "<strong>You are visitor number</strong><br />";
16
17 for (my $count = 0; $count < length($data); $count++)
18 {
19    my $number = substr( $data, $count, 1 );
20    print '<img src="images/counter/' . $number . '.jpg"';
21    print '     alt="' . $number . '">';
22 }
23 print "</div>";
```

Outline

5. Open `counter.dat`, assign to filehandle `COUNTREAD`

7. Increment data in `COUNTREAD`

8. Close `COUNTREAD`

6. Assign data contained in file `counter.dat` to variable `$data`

17. Use `for` structure to output number of page hits using number images

From material © 2000 Deitel & Associates, Inc. All rights reserved.

10

1 SSI (Perl preview)

- Perl scripts can access and modify other files
  - **`open()`** function
    - Form: **`open(fileHandle, ">fileName");`**
      - **`>`** discards any data in file, creates new file if does not exist
      - **`>>`** append mode
      - Returns false on error
      - File handles do not need type (**`$,@,%`**)
      - While file open, referenced using **`fileHandle`**
  - Close file using the **`close()`** statement
    - Format: **`close(fileHandle);`**
- Error checking:
  - **`open(COUNTREAD, "counter.dat")`**
    `|| die "opening 'counter.dat': $!";`
  - See **`die.pl`** and **`warn.pl`** examples

Based on material © 2000 Deitel & Associates, Inc.

11

1 SSI (Perl preview)

`print` statement can redirect output to a file

```
print COUNTWRITE $data;
```

- Assigns **`$data`** to file pointed to by **`COUNTWRITE`**
- *If* the file is open for writing already

Based on material © 2000 Deitel & Associates, Inc.

12

1 SSI (Perl preview II)

- ***`length()`*** *function*
  - Returns length of string
- ***`substr( expr, len, offset )`*** *function*
  - Similar to JavaScript's **`substr`** function
  - First argument (***`expr`***)
    - Specifies string from which to take a substring
  - Second argument (***`offset`***)
    - Specifies offset in characters from beginning of the string
  - Third argument (***`len`***)
    - Specifies length of substring to return

Based on material © 2000 Deitel & Associates, Inc.

13

2 Common Gateway Interface (CGI)

- *Server-side programming*
  - Process data on the server to increase communication between *clients* and *servers*
  - Create interactive applications
- Client-side scripting
  - Not always sufficient when building truly interactive Web-based applications
- *HyperText Transfer Protocol* (*HTTP*)
  - Used for communication between Web browsers and servers
- *Universal Resource Locator* (*URL*)
  - Used by browsers (clients) to specify name of server from which to request data

Based on material © 2000 Deitel & Associates, Inc.

14

2 Common Gateway Interface (CGI) (II)

- HTTP **GET** command
  - By issuing command, client directs server to send specific data to browser
- CGI
  - Lets HTTP clients interact with programs across a network through a Web server
  - A standard for interfacing applications with a Web server
  - CGI applications
    - Can be written in many different programming languages
    - Often reside in the *directory* **`/cgi-bin`**
- Within Web server
  - Permission granted by *webmaster* to allow specific programs to be executed on the server

Based on material © 2000 Deitel & Associates, Inc.

15

2 Common Gateway Interface (CGI) (III)

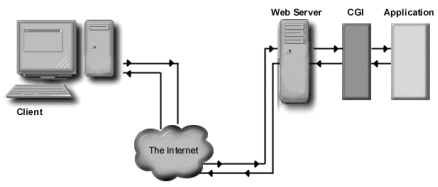
- Interaction methods
  - Standard input (keyboard)
  - Standard output (screen)
- Web browser
  - Take info from user
  - Using HTTP, sends info to a Web server
  - Server-side CGI program executed
  - Standard output from server-side applications or scripts redirected or *piped* to CGI
  - Output sent from CGI over the Internet to client for rendering
- CGI is an interface
  - Cannot be directly programmed
  - Script or executable program must be used to interact with it

Based on material © 2000 Deitel & Associates, Inc.

16

2 Common Gateway Interface (CGI) (IV)

Data path of a typical CGI-based application



Based on material © 2000 Deitel & Associates, Inc.

17

2 CGI Binaries at FCS

- On `borg`
- Must be in ~/`public_html/cgi-bin/` directory
- Must end with `.cgi` no matter what language they're in
- Use `http://`**`borg`**`.cs.dal.ca`

- We run suexec
  - CGI programs are opened by http daemon
  - CGI programs are run by the owner
  - Your CGI programs have your permissions
  - Other options: setuid, run as http (or nobody)
  - See `examples/CGI/about.pl`

Based on material © 2000 Deitel & Associates, Inc.

18

2 Configuring Personal Web Server (PWS) for Perl/CGI

- To run CGI with PWS
  - Several modifications must be made in the *Windows Registry*
    - PWS must be enabled to execute Perl scripts – does not by default

- For detailed instructions on procedure to update Windows Registry to handle Perl scripts
  - See section 3 in *Deitel, et al.* (on reserve in Killam Library)

Based on material © 2000 Deitel & Associates, Inc.

19

3 Introduction to Perl

Perl (***P****ractical* ***E****xtraction and* ***R****eport* ***L****anguage)*

- High-level programming language
- Developed by Larry Wall in 1987
  - Trained as a linguist
  - A systems admin at NASA
- Rich, easy-to-use text-processing capabilities
- Alternative to the tricky C programming language
- Powerful alternative to Unix shell scripts
  - Lots of built-in functionality
  - TMTOWTDI

Based on material © 2000 Deitel & Associates, Inc.

20

3 Introduction to Perl

- Current version: Perl 5.8
  - *Programming Perl* (1st ed.) was about Perl 4
  - Perl 5 is a complete rewrite
  - An entirely new language
- Good choice for programming server side WWW
  - Most popular language for doing so today
  - Is under continuous update by the online Perl community
    - Stays competitive with newer server-side technologies
    - Programmer driven
    - Extensible by modular objects
      - Can even search the online object-base to find newer versions

Based on material © 2000 Deitel & Associates, Inc.

21

3 Introduction to Perl (II)

- Perl initially developed for Unix platform
  - Always intended to be a cross-platform computer language
- *ActivePerl*
  - Version of Perl for Windows
  - Free download at **`http://www.activestate.com`**
  - Includes the *core Perl package*
    - Predefined functionality expected to behave the same across all platforms
    - Perl Interpreter — *perl* — placed in bin directory
      - Loaded into memory each time Perl program invoked
  - Extension of Perl programs is **`.pl`**
    - Associated with Perl interpreter by default
- Perl program execution
  - Type **`perl -w`** followed by filename of Perl source code at command line (Unix or DOS prompt)

Based on material © 2000 Deitel & Associates, Inc.

22

3 Introduction to Perl (III)

Perl command line switches (case sensitive)

Command-line switch	Meaning
`-e 'command'`	Interpret one line of Perl code
`-S`	Search for the specified script using the **PATH** environment variable
`-T`	Turn on taint mode (must be first switch)
`-v`	Print the version of Perl
`-w`	Allow warnings to be displayed on compilation of the script
`-h`	Display all options for `perl`

Based on material © 2000 Deitel & Associates, Inc.

23

3 Introduction to Perl (IV)

- *Comment character* `#`
  - Goes at beginning of every line with comment
- Function **`print`**
  - Outputs text indicated by quotation marks (**"…"**)
- Escape sequences
  - E.g. `\n, \t, \a`
  - Newline, tab, alert
- Statements terminated with semicolons (`;`)
  - Exception: where braces (`{}`) used to denote block of code

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig. 4: first.pl
2  # A first program in Perl.
3
4  print "Welcome to Perl!\n";
```

Outline

1.1 Print Statement

```
Welcome to Perl!
```

From material © 2000 Deitel & Associates, Inc. All rights reserved.

3 Introduction to Perl (V)

- Perl contains set of data types
  - Represent different kinds of information
  - Each variable name has special character preceding it
    - **$** - variable contains scalar value
      - Strings, integer numbers and floating-point numbers
    - **@** - indexed array
      - Uses an integer (called an index) to reference array elements
    - **%** - hash (associative array)
      - Uses keys that are strings to reference individual array elements
  - Variables should be initialized before being used
- Variable names in strings
  - Serve as place-holders for values they represent
  - If have no declared value – set to *undef* (empty) value

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig. 6: variable.pl
2  # Program to illustrate the use of scalar variables.
3
4  # using a variable in the context of a string
5  print "Using a variable before initializing: $var\n";
6
7  # using a variable in a numeric context
8  $test = $num + 5;
9  print "Adding uninitialized variable num to 5 yields: $test.\n";
10
11 $a = 5;
12 print "The value of variable a is: $a\n";
13
14 $a = $a + 5;
15 print "Variable a after adding 5 is $a.\n";
16
17 $b = "A string value";
18 $a = $a + $b;
19
20 print "Adding a string to an integer yields: $a\n";
21
22 $number = 7;
23 $b = $b + $number;
24
25 print "Adding an integer to a string yields: $b\n";
```

```
Using a variable before initializing:
Adding uninitialized variable num to 5 yields: 5.
The value of variable a is: 5
Variable a after adding 5 is 10.
Adding a string to an integer yields: 10
Adding an integer to a string yields: 7
```

Outline

1.1 Demonstrate variable in string before initialization

1.2 Demonstrate addition involving variable using print statements

1.3 Add integer to string and print result

Add integer to string and print result

3 Introduction to Perl (VI)

- Perl can store arrays
  - Arrays divided into elements
    - Each can contain an individual scalar variable
- Array definition

  ```
  @arrayName = ("element1", "element2", …,
    "elementN");
  ```
- First array element is **[0]**
  - Just like C, C++, etc.
  - Could be changed in Perl 4 but should not in Perl 5

Based on material © 2000 Deitel & Associates, Inc.

28

3 Introduction to Perl (VII)

- Arrays
  - Elements are referenced as scalar values with element number in square brackets (**[]**)
    - **@** refers to array as a whole, **$** refers to elements
      Example: **$array[2]**
    - Refers to the third element in **@array**
- Range Operator – "**..**"
  - Used to store all values between given arguments
    Example: **@array2 = (A..Z);**
  - Creates array **@array2** containing all capital letters in alphabet (all letters between **A** and **Z**)

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig. 7: arrays.pl
2  # Program to demonstrate arrays in Perl
3
4  @array = ("Bill", "Bobby", "Sue", "Michelle");
5
6  print "The array contains:\n\n";
7  print "@array \n\n";
8  print "Third element: $array[2]\n\n";
9
10 @array2 = (A..Z);
11
12 print "The range operator is used to store all\n";
13 print "letters from capital A to Z:\n\n";
14 print "@array2 \n";
```

```
The array contains:

Bill Bobby Sue Michelle

Third element: Sue

The range operator is used to store all
letters from capital A to Z:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Outline

1.1 Define array **@array**

2.1 Print contents of **@array**

2.2 Print third element of **@array**

3.1 Define array **@array2**

3.2 Explain and print contents of **@array2**

From material © 2000 Deitel & Associates, Inc. All rights reserved.

30

3 Introduction to Perl (VIII)

- In addition to core Perl package
  - Add-ons called *packages* provide additional functionality
- Packages
  - Often provide platform specific features
  - Are available at
    http://www.cpan.org
    http://www.activestate.com/packages

Based on material © 2000 Deitel & Associates, Inc.

3 String Processing and Regular Expressions

- Processing textual data easily and efficiently
  - One of Perl's most powerful capabilities
  - Usually done through use of *regular* expressions
    - Patterns of characters used to search through text files and databases
    - Allows large amounts of text to be searched using relatively simple expressions
- **eq** equality operator
  - Tests whether two strings are equivalent

  example: **`if ($hello eq "Good Morning")`**…
- Keyword **my**
  - Designates variable only valid for block of code in which it is declared

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig. 16: equals.pl
2  # Program to demonstrate the eq operator
3
4  my $stringa = "Test";
5  my $stringb = "Testing";
6
7  if ($stringa eq "Test")
8  {
9     print "$stringa matches Test.\n";
10 }
11 else
12 {
13    print "$stringa does not match Test.\n";
14 }
15
16 if ($stringb eq "Test")
17 {
18    print "$stringb matches Test.\n";
19 }
20 else
21 {
22    print "$stringb does not match Test.\n";
23 }
```

```
Test matches Test.
Testing does not match Test.
```

Outline

1.1 Declare variables using my

2.1 Test string variable-string equality

2.2 Print appropriate result

3.1 Test second variable

3.2 Print appropriate result

From material © 2000 Deitel & Associates, Inc. All rights reserved.

3 my and local

- Keyword **my**
  - Designates variable only valid for block of code in which it is declared
  - In Perl 4 was done by `local`
- `my` creates local variables
- `local` creates local copy & then restores it on exit
- See following program …

Based on material © 2000 Deitel & Associates, Inc.

3 my and local (program)

```
$lo = 'global';
$m  = 'global';
A();
sub A {
  local $lo = 'string';
  my    $m  = 'string';
  B();
}
sub B {
  print "B ", ($lo eq 'string' ?'can' :'cannot'),
        " see the value of lo set by A.\n";
  print "B ", ($m eq 'string' ?'can' :'cannot'),
        " see the value of m set by A.\n";
}
------------------------------------------------------------
B can see the value of lo set by A.
B cannot see the value of m set by A.
```

Based on material © 2000 Deitel & Associates, Inc.

3 String Processing and Regular Expressions (II)

- **eq** operator
  - Cannot be used to search through a series of words
- String binding 'operator' **=~**
  - Tests whether match for a string is found within a single string or series of words
- Example

  `$search =~ /Test/;`

  Searches for word test within indicated string

  `$string =~ s/Regular/regular/g;`

  Makes the substitution operation work on **`$string`**, instead of **`$_`**

Based on material © 2000 Deitel & Associates, Inc.

3 String Processing and Regular Expressions (III)

- Some *meta/modifying characters*
  - **^** – indicates beginning of a line
  - **$** – indicates end of a line (matches **\n**)
  - **\b** – indicates word boundary
  - **\w** – matches any *alphanumeric character* and underscore `[a-z_A-Z0-9]`
- Other modifying characters

Modifying Character	Meaning
/g	Search everywhere for the expression (global search).
/i	Ignores the case of the search string.
/m	The string is evaluated as if it had multiple lines (i.e., contains multiple newline characters) of text. (^ and $ work differently, Use \A for start of string, and \Z for end of string)
/s	Ignore the newline character and treat it as whitespace. The text is seen as a single line.
/x	All whitespace characters are ignored when searching the string.

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig 17: expression1.pl
2  # searches using the matching operator and regular expressions
3
4  $search = "Testing pattern matches";
5
6  if ( $search =~ /Test/ )
7  {
8     print "Test was found.\n";
9  }
10
11 if ( $search =~ /^Test/ )
12 {
13    print "Test was found at the beginning of the line.\n";
14 }
15
16 if ( $search =~ /Test$/ )
17 {
18    print "Test was found at the end of the line.\n";
19 }
20
21 if ( $search =~ / \b ( \w+ es ) \b /x )
22 {
23    print "Word ending in es: $1 \n";
24 }
```

```
Test was found.
Test was found at the beginning of the line.
Word ending in es: matches
```

Outline

1.1 Test for word 'Test' in string, print result

2.1 Test for word 'Test' at beginning on string, print result

3.1 Test for word 'Test' at end of string, print result

4.1 Test for word in string ending with letters 'es', print result

From material © 2000 Deitel & Associates, Inc. All rights reserved.

4 Viewing Client/Server Environment Variables

- Knowing info about client very useful to system administrators
- *CGI environment variables*
  - Contains info about client
    - Web browser being used
    - Version of CGI server running
    - HTTP host, HTTP connection
    - Much more (we'll see example shortly)
- ***use*** statement
  - Includes predefined library packages in programs

Based on material © 2000 Deitel & Associates, Inc.

4 Viewing Client/Server Environment Variables (II)

- *CGI Library*
  - Included to provide functionality that makes it easier to write HTML sent to Web browser
  - Contains keywords that represent HTML tags
- ***foreach*** *loop*
  - Iterates through keys in given *hashtable*, performs indicated actions
    `foreach $key (sort keys %ENV)`
  - Iterates through **`%ENV`** hashtable
    - Built-in table in Perl that contains names and values of all CGI environment variables
  - **`sort`** function
    - returns list in lexographical order
  - Assigns current key to **`$key`** and performs indicated actions

Based on material © 2000 Deitel & Associates, Inc.

42

4 `env.cgi`

- Source: .../examples/perl/env.pl.source
- Execute

Based on material © 2000 Deitel & Associates, Inc.

43

4 Taint mode

- When in *taint mode* perl won't let you user input to open files, etc.
- Taint mode on when running as CGI or with `-T` switch
  `-T` must be first switch, use `-Tw` to get both `T` and `w`
- To remove taint from variables
  - Use regular expression backreferences

```
$file = param("filename"); # input from CGI form
if ( $file !~ /^([\w.-]+)$/ ) {
   die "filename `$file´ has invalid characters\n";
} else {
   $file = $1;
}
```

Based on material © 2000 Deitel & Associates, Inc.

44

4 CGI Binaries at FCS

- On `borg`
- Must be in ~/`public_html/cgi-bin/` directory
- Must end with **`.cgi`** no matter what language they're in
- Use `http://`**`borg`**`.cs.dal.ca`

Based on material © 2000 Deitel & Associates, Inc.

45

5 Form Processing and Business Logic

- HTML **FORM**s
  1. Allow users to enter data
  2. Data sent to Web server for processing
  3. Program processes data
  - Allows users to interact with server
  - Vital to electronic commerce
- **FORM** element
  - Indicates what action should occur when user submits form
  - Attribute: **ACTION = "cgi-bin/form.pl"**
    - Directs server to execute **form.pl** Perl script
    - Example

Based on material © 2000 Deitel & Associates, Inc.

49

5 Form Processing and Business Logic (II)

- Retrieving data from form output
  - Assign to variables
  - Example: Assign data from form **INPUT OS** to variable **$os**

```
$os = param(OS);
```

- Testing for correct form input
  - Example: Make sure phone number in format (555)555-5555

```
if ( $phone =~ / \( \d{3} \) \d{3} - \d{3} /x ) { actions }
```

  - **d{n}** tests for **n** characters
  - **** is escape character
    - Close-bracket (‘**)**’) character is used in Perl statements, needs escape character ‘****’ to appear as part of search test string

Based on material © 2000 Deitel & Associates, Inc.

54

6 Verifying Username & Password

- Often desirable to have private Web site
  - Developers often employ username and password authentication to implement privacy
  - In reality we would use the server software to do this
  - We'll see an example with perl
- Upcoming files
  - **verify.html** – HTML document client browser displays
  - **password.pl** – Perl script that verifies username and password inputted by client and performs appropriate actions
  - **data.txt** – Text file containing username and password combinations (unencrypted for simplicity)

Based on material © 2000 Deitel & Associates, Inc.

6 Verifying Username & Password (II)

- If file cannot be opened
  - Use function **`die`** to exit program and print message
- **`while <fileHandle>`**
  - Executes structure while still information in ***`fileHandle`***
  - Assigns a line at a time to **`$_`**
- **`split`** function
  - Read contents of a file into an array
    `@arrayName = split(/\n/)`
  - Creates array ***`arrayName`***, creates new array entry after every **`\n`** character
- Access array elements and split into two parts
  `foreach $entry (@data) {…}`
  - Performs indicated action on every entry in array **`@data`**
  - Subsequently assigns entry information to **`$entry`**

Based on material © 2000 Deitel & Associates, Inc.

6 Verifying a Username and Password (III)

- **`split`** array into two parts
  `($name, $pass) = split(/,/, $entry)`
  - Assigns username string of current entry to **`$name`**
  - Assigns password string of current entry to **`$pass`**

Based on material © 2000 Deitel & Associates, Inc.

6 Verifying a Username and Password (III)

- Perl has logical *and* (**`&&`**) and *or* (**`||`**) operators
  - Same format as other languages

  Example:
  `if ($userverified && $passwordverified) {…}`
  - Evaluates to true if both variable values are true
  - Short-circuit evaluation
    - String context: true is any non-empty string
    - Numeric context: true is any non-zero number
    - String `"0"` is false!
    - String `"00"` is true!

Based on material © 2000 Deitel & Associates, Inc.

6 Verifying a Username and Password (III)

sub *functionName* {…}

- Sets actions of user-defined function ***functionName***
- User-defined functions accessed:
  - ***&functionName*** — old style, not used much
  - ***functionName()*** — preferred form, allows for extras

Based on material © 2000 Deitel & Associates, Inc.

```
1<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2<!-- Fig. 24: verify.html -->
3
4<HTML>
5<HEAD>
6<TITLE>Verifying a username and a password.</TITLE>
7</HEAD>
8
9<BODY BACKGROUND = "images/back.gif">
10   <P>
11      <FONT FACE = Arial>
12      Type in your username and password below.
13      </FONT><BR>
14      <FONT COLOR = #0000FF FACE = Arial SIZE = 1>
15      <STRONG>
16      Note that password will be sent as plain text
17      </STRONG>
18      </FONT>
19   </P>
20
21   <FORM ACTION = "/cgi-bin/password.pl" METHOD = "post">
22   <BR>
23
24   <TABLE BORDER = "0" CELLSPACING = "0" STYLE = "HEIGHT: 90px;
25          WIDTH: 123px" CELLPADING = "0">
26      <TR>
27         <TD BGCOLOR = #DDDDDD COLSPAN = 3>
28         <FONT FACE = Arial SIZE = 2>
29         <STRONG>Username:</STRONG>
30         </FONT>
31         </TD>
32      </TR>
```

Outline

1.1 Print instructions

2.1 Open `FORM` and define `ACTION` attribute

3.1 Open HTML `TABLE`

```
33      <TR>
34         <TD BGCOLOR = #DDDDDD COLSPAN = 3>
35         <INPUT SIZE = "40" NAME = "USERNAME"
36                STYLE = "HEIGHT: 22px; WIDTH: 115px">
37         </TD>
38      </TR>
39      <TR>
40         <TD BGCOLOR = #DDDDDD COLSPAN = 3>
41         <FONT FACE = Arial SIZE = 2>
42         <STRONG>Password:</STRONG>
43         </FONT></TD>
44      </TR>
45      <TR>
46         <TD BGCOLOR = #DDDDDD COLSPAN = 3>
47         <INPUT SIZE = "40" NAME = "PASSWORD"
48                STYLE = "HEIGHT: 22px; WIDTH: 115px"
49                TYPE = PASSWORD>
50         <BR></TD>
51      </TR>
52      <TR>
53         <TD COLSPAN = 3>
54         <INPUT TYPE = "submit" VALUE = "Enter"
55                STYLE = "HEIGHT: 23px; WIDTH: 47px">
56         </TD>
57      </TR>
58   </TABLE>
59   </FORM>
60</BODY>
61</HTML>
```

Outline

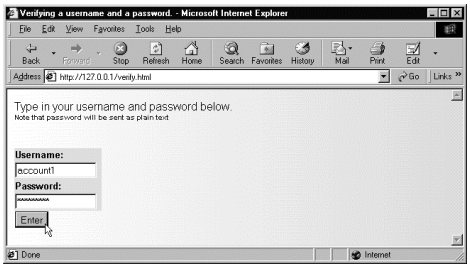
3.2 Insert and define `INPUT` elements for username and password

3.3 Insert `INPUT` `submit` button

3.4 Close `TABLE` and `FORM` elements

61

Script Output



Based on material © 2000 Deitel & Associates, Inc.

```
1  account1,password1
2  account2,password2
3  account3,password3
4  account4,password4
5  account5,password5
6  account6,password6
7  account7,password7
8  account8,password8
9  account9,password9
10 account10,password10
```

Outline

Data.txt

1.1 Input username and password combinations using format:

`username,password/n`

From material © 2000 Deitel & Associates, Inc. All rights reserved.

66

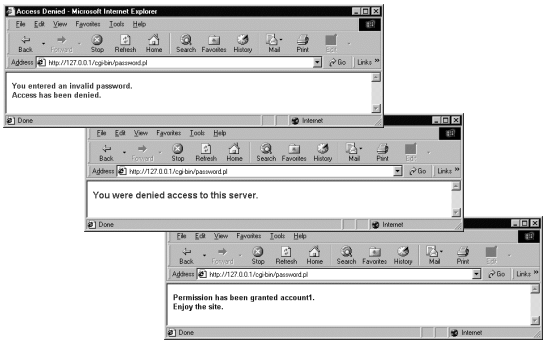
6 Verifying a Username and Password (IV)

- See example `Fig_27_25.pl`

Based on material © 2000 Deitel & Associates, Inc.

67

Script Output



Based on material © 2000 Deitel & Associates, Inc.

75

7 Cookies

- *What?* Client-side storage for server-side use
- *Why?* To save state information
- *How?*
  - When server sends document is can also send a cookie
  - When client requests document it can also send back cookie with request

Based on material © 2000 Deitel & Associates, Inc.

76

7 Cookies

Some Details

- Server sends '`Set-Cookie:`' header
  - `NAME = VALUE` is required
- Parameters separated by semicolons (;)
- Optional parameters
  - `Expires=`
    - When the cookie ceases to be (crumbles)
    - If not set then expiry is end of browser process
  - `Domain=`
    Site to send cookie back to
  - `Path=`
    What file (directory) it applies to
  - `Secure=`
    Do not send with unsecured protocol

Based on material © 2000 Deitel & Associates, Inc.

77

7 Cookies

Some More Details

- Multiple set-cookie headers allowed
- Cookies can overwrite each other
- Expires times in the past are used to delete cookies

- Limits:
  - 300 cookies
  - 4 Kb per cookie
  - 20 cookies per server or domain

Based on material © 2000 Deitel & Associates, Inc.

78

7 Cookies

Examples from the draft specification

Based on material © 2000 Deitel & Associates, Inc.

79

7 Cookies

•Pros •Cons

Based on material © 2000 Deitel & Associates, Inc.

80

7 Cookies and Perl (II)

- To set a cookie using plain Perl
  - Set variable values to user input strings
  - Set cookie setup info
    - **`$expires`** – expiration date of cookie
    - **`$path`** – location on clients computer to store cookie
    - **`$server_domain`** – IP address of your server
  - **`print "set-cookie: ";`** …
    set information to be stored in cookie using print statement
  - Repeat as needed to store all information in cookie

Based on material © 2000 Deitel & Associates, Inc.

81

7 Cookies and Perl (III)

- Internet Explorer stores cookies
  - Text file added to **Temporary Internet Files** directory
    - Filename: **`Cookie:administrator@`*`ip.number`***

Based on material © 2000 Deitel & Associates, Inc.

```
1<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2<!-- Fig. 32: cookies.html -->
3
4<HTML>
5   <HEAD>
6      <TITLE>Writing a cookie to the client computer</TITLE>
7   </HEAD>
8
9<BODY BACKGROUND = "images/back.gif">
10<BASEFONT FACE = "ARIAL,SANS-SERIF" SIZE = 2>
11
12   <FONT SIZE = +2>
13      <B>Click Write Cookie to save your cookie data.</B>
14   </FONT><BR>
15
16   <FORM METHOD = "POST" ACTION = "cgi-bin/cookies.pl">
17      <STRONG>Name:</STRONG><BR>
18      <INPUT TYPE = "TEXT" NAME = "NAME"><BR>
19      <STRONG>Height:</STRONG><BR>
20      <INPUT TYPE = "TEXT" NAME = "HEIGHT"><BR>
21      <STRONG>Favorite Color</STRONG><BR>
22      <INPUT TYPE = "TEXT" NAME = "COLOR"><BR>
23      <INPUT TYPE = "SUBMIT" VALUE = "Write Cookie">
24   </FORM>
25</BODY>
26</HTML>
```

Outline

1.1 Enter text instructions

2.1 Open `FORM` and define `ACTION` attribute

2.2 Insert and define `INPUT` fields

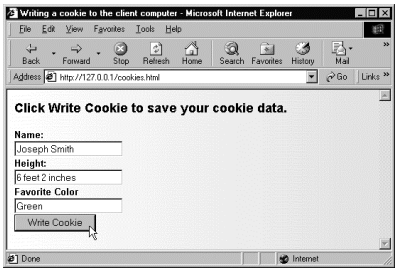
2.3 Insert `INPUT` `submit` button

2.4 Close `FORM` area

From material © 2000 Deitel & Associates, Inc. All rights reserved.

83

Script Output



Based on material © 2000 Deitel & Associates, Inc.

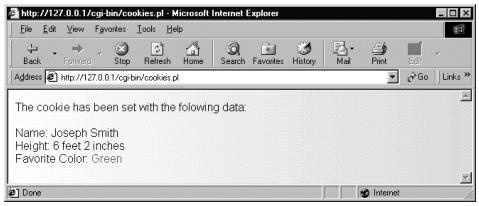
Outline

```
1  # Fig. 33: cookies.pl
2  # Program to write a cookie to a client's machine
3
4  use CGI qw/:standard/;
5
6  my $name = param(NAME);
7  my $height = param(HEIGHT);
8  my $color = param(COLOR);
9
10 $expires = "Monday, 20-Dec-99 16:00:00 GMT";
11 $path = "";
12 $server_domain = "10.0.1";
13
14 print "Set-Cookie: ";
15 print "Name", "=", $name, "; expires=", $expires,
16       "; path=", $path, "; domain=", $server_domain, "\n";
17
18 print "Set-Cookie: ";
19 print "Height", "=", $height, "; expires=", $expires,
20       "; path=", $path, "; domain=", $server_domain, "\n";
21
22 print "Set-Cookie: ";
23 print "Color", "=", $color, "; expires=", $expires,
24       "; path=", $path, "; domain=", $server_domain, "\n";
25
26 print header;
27 print "<BODY BACKGROUND = \"/images/back.gif\">";
28 print "<BASEFONT FACE = \"ARIAL,SANS-SERIF\" SIZE = 3>";
29 print "The cookie has been set with the folowing data:";
30 print "<BR><BR>";
31 print "<FONT COLOR=BLUE>Name:</FONT> $name <BR>";
32 print "<FONT COLOR = BLUE>Height:</FONT> $height<BR>";
33 print "<FONT COLOR = BLUE>Favorite Color:</FONT> ";
34 print "<FONT COLOR = $color> $color<BR>";
```

From material © 2000 Deitel & Associates, Inc. All rights reserved.

85

Script Output



Based on material © 2000 Deitel & Associates, Inc.

7 Cookies and Perl (IV)

- Cookies are read from client machine using Perl
  - Subroutine **`readCookies`** returns the information stored in cookies sent to client from server ip address
    - Information read with statement
      `$ENV{'HTTP_COOKIE'}`
  - Cookie information can be read by
    - Storing information in hash array
    - Splitting fields
    - Displaying information
- Display cookie output in table for organization

Based on material © 2000 Deitel & Associates, Inc.

```
1  # Fig. 36: read_cookies.pl
2  # Program to read cookies from the client's computer
3
4  use CGI qw/:standard/;
5
6  print header;
7  print "<BODY BACKGROUND = \"/images/back.gif\">";
8  print "<BASEFONT FACE = \"ARIAL,SANS-SERIF\" SIZE = 3>";
9  print "<STRONG>The following data is saved in a cookie on your ";
10 print "computer.</STRONG><BR><BR>";
11
12 my %cookie = &readCookies;
13
14 print ("<TABLE ",
15        "BORDER = \"5\" ",
16        "CELLSPACING = \"0\" ",
17        "CELLPADDING = \"10\">");
18
19 foreach $cookie_name (keys %cookie)
20 {
21    print "<TR>";
22    print "   <TD BGCOLOR=#AAAAFF>$cookie_name</TD>";
23    print "   <TD BGCOLOR=#AAAAAA>$cookie{$cookie_name}</TD>";
24    print "</TR>";
25 }
26 print "</TABLE>";
27
28 sub readCookies
29 {
30    my @cookie_values = split (/; /,$ENV{'HTTP_COOKIE'});
31
32    foreach (@cookie_values)
```

Outline

1.1 `use CGI standard` library

1.2 `print header`

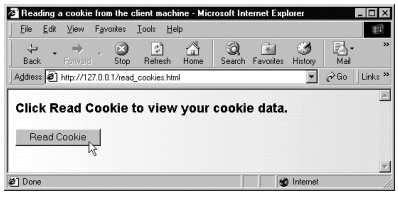
2.1 Call function `readCookies` to and store info in `%cookie`

3.1 Use `foreach` structure to output cookie info

4.1 Define function readCookies

4.2 Put cookie information into an array

```
33    {
34       my ($cookie_name, $cookie_value) = split ( /=/, $_ );
35       $cookies{$cookie_name} = $cookie_value;
36    }
37
38    return %cookies;
39 }
```



Outline

4.3 Split cookie entry names and values

4.4 Return information for output

From material © 2000 Deitel & Associates, Inc. All rights reserved.

89

Script Output

Based on material © 2000 Deitel & Associates, Inc.

90

7 Cookies and `CGI.pm`

```
use CGI qw(:standard);
my $cookie = cookie(-name=>'regular',
                    -value=>'chip');
print header(-cookie=>$cookie);

----------------------------------------

Set-cookie: regular=chip
Content-type: text/html
```

Examples

Based on material © 2000 Deitel & Associates, Inc.