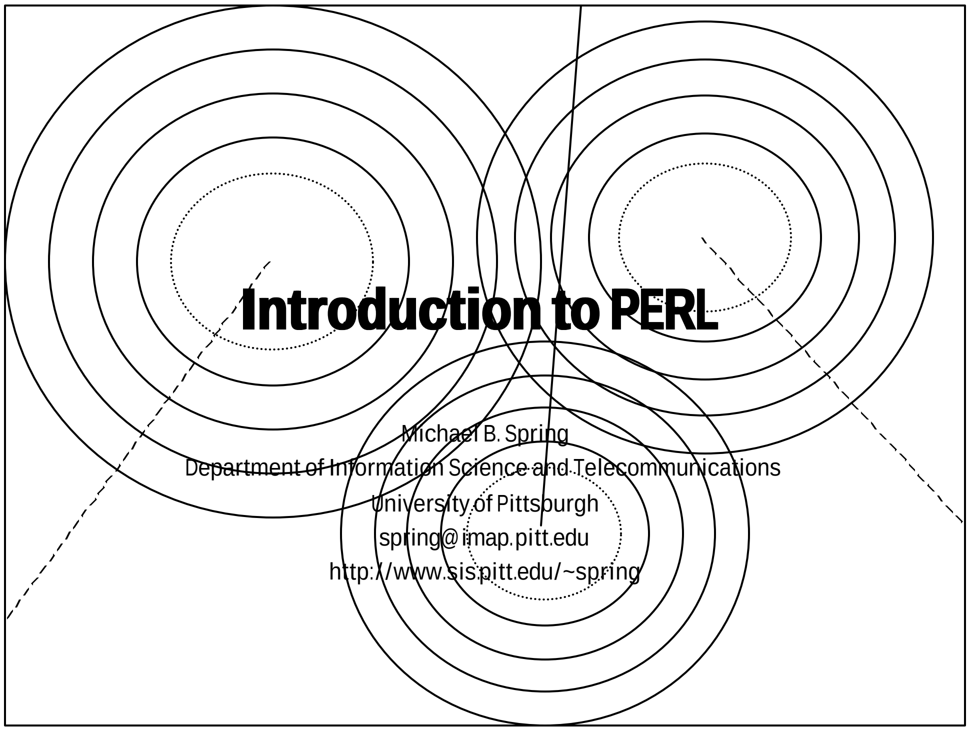
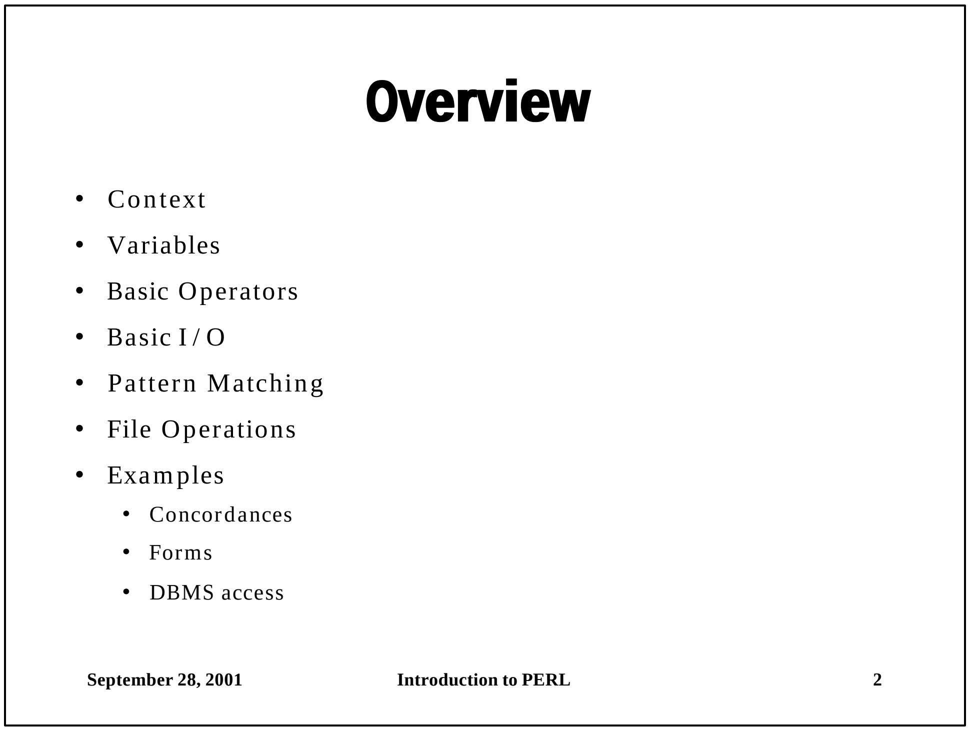


Introduction to PERL

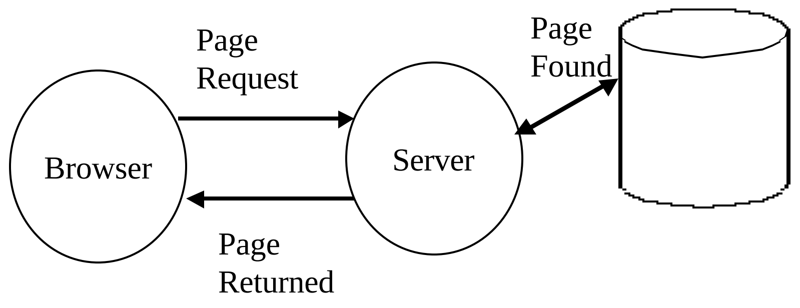
Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



Overview

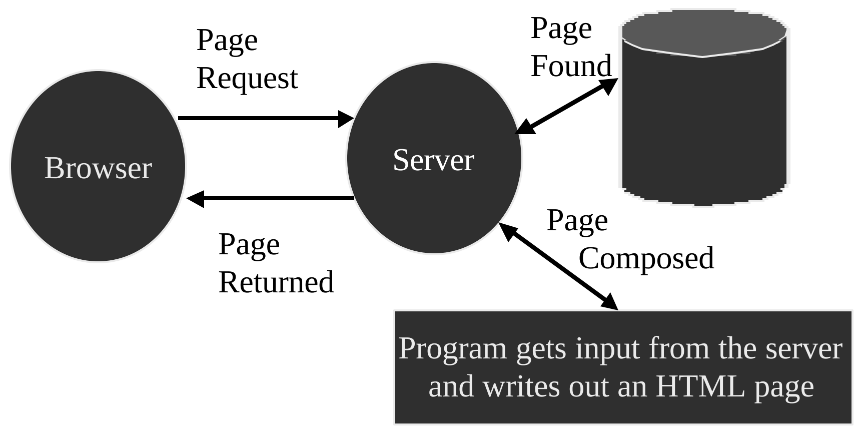
- Context
- Variables
- Basic Operators
- Basic I/O
- Pattern Matching
- File Operations
- Examples
  - Concordances
  - Forms
  - DBMS access

Context
Producing Pages Statically



September 28, 2001 Introduction to PERL 3

Context
Producing Pages Dynamically



Program gets input from the server and writes out an HTML page

September 28, 2001 Introduction to PERL 4

Why PERL

- PERL, “Practical Extraction and Report Language” is a interpreted language
- The strengths of PERL include:
  - String handling
  - Regular expression capabilities
  - File handling
- It was a natural choice of Unix System Administrators as a familiar rapid prototyping tool to produce reports or pages of HTML
- It is now being replaced by more object oriented approaches

Variables

- Variables in PERL are not declared in advance
- They are assigned and defined in context
- There are three basic variable types
  - $name ::= a simple scalar; type may be real, integer, string, etc.
  - @name ::= an array of objects -- may be scalar, array, or other
  - %name ::= a hashed array -- items are identified by a name.
- Once array (and hash) variables are defined, some other things may be assumed:
  - $name[n] ::= is the n+1th element of @name
  - $#name ::= the number of items in @name
  - @name[n-m] or %name{'hash', 'hash2'} ::= slice of array
  - $name[-1]::= the last element of the array @name

General “Special” Variables

- There are dozens of special variables defined in PERL.
- The most ubiquitous and important of these is $_
  - the default input and the default pattern search space
    - The default variable for all loops
    - The default variable for file I/O
    - The default space searched in a pattern match
- @_ is the parameter array for subroutines
- Other useful special variables include:
  - $. Is the current input line number of last file read
  - @ARGV are the arguments to the script
  - %ENV are the environment variables – in a hash

“Special” Variables Related to Pattern Matching

- $& is the string matched by the last successful pattern match
- $’, $&, $` is the pre, match, and post string of the last successful pattern match
- $1--$9 the subpatterns in the last pattern match

Basic Operators

- PERL has most of the operators one would find in c.
- These include
  - + - * / %
  - +=, etc
  - ++ --
  - < >= <= etc.
- you also have
  - x -- string multiplication
  - . -- string concatenation
  - lt gt le ge -- for string comparison
  - eq ne cmp -- for string equality checking
  - =~ binds a pattern match to a scalar
  - !~ same but negates a result.

Basic Statements -- flow control

- if (expression) {block} else {block}
- until (expression {block} continue {block}
- while (expression) {block} continue {block}
- For, while, and other blocks have
  - Redo
  - Next
  - Last
- for (expression) {block}
- foreach var (list) {block}

Basic I/O

- To open a file:
  open (FILEHNDLE, $accessmode.$filenamestr)
  - accessmodes are <=read >=write +>= create >>=append
- Context is important in input
  @var = <FILEHNDLE> reads an entire file (“array context”)
  $var = <FILEHNDLE> reads one line (“scalar context”)
  <> STDIN or the files in ARGV
  getc FILEHNDLE or STDIN
- To write a file, or STDOUT
  Print FILEHNDLE list
  - There is also binary reads and writes
- File positioning
  seek FILEHNDLE position whence
  tell FILEHNDLE

File Operations

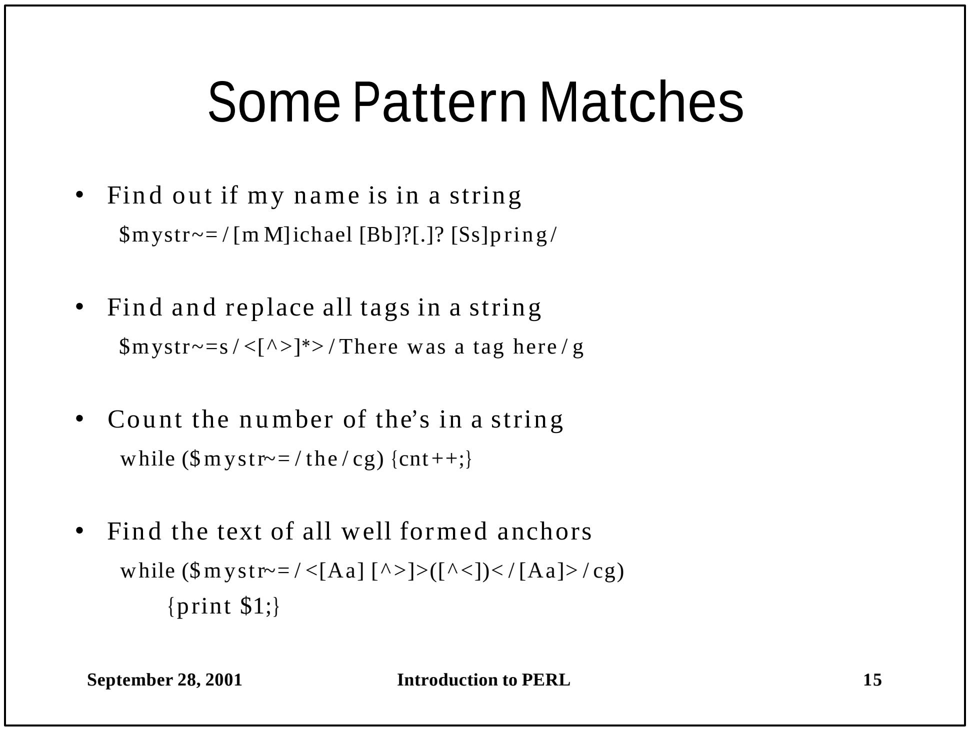
- Several operators return information about a file. The most useful include:
  - -r -w -x File is readable/writable/executable
  - -e -z File exists / has zero size.
  - -f -d File is a plain file, a directory.
  - -s File exists and has non-zero size. Returns the size.
  - -M -A -C File modification/access/inode-change time (days)
- The –s option allows things like
  Read FHNDL, -s FHNDL
- Other interesting functions include the ability to change mode (chmod), change owner (chown), truncate a file, make directories, and remove directories

Pattern Matching

- Pattern matching makes use of patterns and replacements to do several things:
  - Options when searching and replacing include:
    - c =continue from previous match
    - g = global
    - i = case insensitive
    - m = accept embedded newlines
    - o = interpolates variables only once
    - s = include newline as a . Character
    - x = allow for extensions
  - Search a string -- / pattern / option
  - Search and replace -- s / pattern / replace / option
- In the pattern, []s define a single character and ()s enclose a subpattern to be remembered

Patterns

- between the / 's include things like the following:
  - XYZ ::= the characters themselves
  - XYZ$ ::= the characters themselves, at the end of a line
  - ^XYZ ::= the characters themselves, at the start of a line
  - x(a | b)f ::= x followed by a or b followed by f
  - x.y.z ::= x followed by any character followed by y followed by any character by z
  - x* ::= 0 or more x's
  - y+ ::= one or more y‘s
  - z? ::= 0 or one z's



Some Pattern Matches

- Find out if my name is in a string

```
$mystr~=/[mM]ichael [Bb]?[.]? [Ss]pring/
```

- Find and replace all tags in a string

```
$mystr~=s/<[^>]*>/There was a tag here/g
```

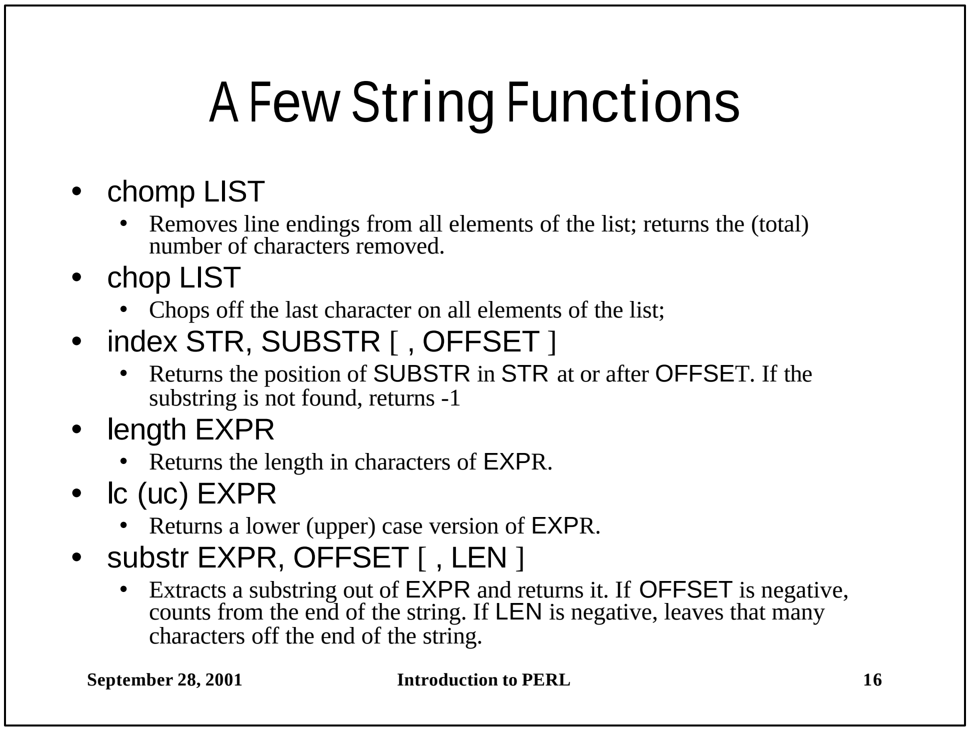
- Count the number of the’s in a string

```
while ($mystr~=/the/cg) {cnt++;}
```

- Find the text of all well formed anchors

```
while ($mystr~=/<[Aa] [^>]>([^<])</[Aa]>/cg)
      {print $1;}
```

September 28, 2001 **Introduction to PERL** **15**



A Few String Functions

- chomp LIST
  - Removes line endings from all elements of the list; returns the (total) number of characters removed.
- chop LIST
  - Chops off the last character on all elements of the list;
- index STR, SUBSTR [, OFFSET]
  - Returns the position of SUBSTR in STR at or after OFFSET. If the substring is not found, returns -1
- length EXPR
  - Returns the length in characters of EXPR.
- lc (uc) EXPR
  - Returns a lower (upper) case version of EXPR.
- substr EXPR, OFFSET [, LEN]
  - Extracts a substring out of EXPR and returns it. If OFFSET is negative, counts from the end of the string. If LEN is negative, leaves that many characters off the end of the string.

September 28, 2001 **Introduction to PERL** **16**

A Few Array Functions

- pop [@ARRAY], push @ARRAY, LIST
  - Pops off and returns the last value of the array.
    Pushes the values of the list onto the end of the array.
- reverse LIST
  - In array context: returns the LIST in reverse order.
- split [PATTERN [, EXPRy [, LIMIT]]]
  - Splits a string into an array of strings, and returns it. LIMIT specifies the max number of fields. If PATTERN is omitted, splits on whitespace.
- join EXPR, LIST
  - Joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string.
- sort [SUBROUTINE] LIST
  - Sorts the LIST and returns the sorted array value. If SUBROUTINE is specified, gives the name of a subroutine that returns less than zero, zero, or greater than zero, depending on how the elements of the array, available to the routine as variables $a and $b.

A Couple Hash Functions

- values %HASH
  - Returns a normal array consisting of all the values of the named hash keys
- %HASH
  - Returns an array of all the keys of the named hash.
- each %HASH
  - Returns a 2-element array consisting of the key and value for the next value of the hash. After all values of the hash have been returned, an empty list is returned. The next call to each after that will start iterating again.

Four Sample Programs

- Concordance: A simple perl program that simply counts the occurrence of words in a file – it is not cgi script related, but it could be used to index documents on a web site. It demonstrates file I/O, string manipulation, and hashes
- HTML Page Analysis: Shows the use of pattern matching to remove tags from a page and to analyze the tags on a page
- HTML Form Porcessing: Shows a page with a form and a CGI perl script using cgi-lib.pl to parse the values
- DBMS Access: A simple perl program to access an access DBMS

Program to Produce a Concordance

- produce a count of all the words in a file

```
#! /usr/bin/perl
open DATA, "testparse.dat";
while(<DATA>)
{
  foreach $term (split)
   {
   $aw{$term}++;
   $tw++;
   }            }
}
```

Concordance Program Output 1

# a printout of the results sorted in alphabetical order

```
$i=0;
print  "\nHere is the list alphabetically\n";
@list = sort {(uc $a) cmp (uc $b) } keys %aw;
foreach (@list)
{
if ($i++%3==0) {print "\n";}
$a=substr $_ . "                ", 0, 16;
print  $a . ": " . $aw{$_} . "\t";
}
print "\n\n";
```

Concordance Program Output 2

# a printout of the results sorted in order of occurrence

```
$i=0;
print  "\nHere is the list in terms of frequency\n";
@list = sort {$aw{$a} <=> $aw{$b} } keys %aw;
foreach (@list)
  {
  $i++;
  if ($i%3==0) {print "\n";}
  $a = substr $_ . "                ", 0, 16;
  print  $a . ": " . $aw{$_} . "\t";
  }
print "\n\n";
print  "\nThere were ". $tw . " words in the file\n";
```

HTML Page Analysis: Text without Tags

# for an html file with tags

```
#! /usr/bin/perl
open DATA, “page.html";
read DATA, $filestring –s DATA
$cleanstr ~=s/<[^>]*>//g;
@tl = split /[ \n\t\.\,\d]+/, cleanstr;
foreach $term (@tl)
          {
          $aw{$term}++;
          $tw++;
          }
}
```

HTML Page Analysis: Tags

#for an html file with tags

```
#! /usr/bin/perl
open DATA, “page.html";
read DATA, $filestring –s DATA
while($filestring ~=s/(<[^>]*>)//cg)
    {
    push @tags, $1;
    }
foreach $tag (@tags)
    {$aw{$tag}++;
    $tw++;
    }
```

A Form

```
<form method="POST"
action="http://augment.sis.pitt.edu/cgi-bin/comm_form.cgi">
<P>Name:
<input type="text" SIZE = "40" MAXLENGTH="80" name="name" value =
   "anonymous”>
<P>Subject:
<input type="text"SIZE = "40" MAXLENGTH="80"  name="subject" value =
   "None”>
<P>Comments:
<textarea NAME="comment" ROWS=8 COLS=40>
type your comment or poem here</textarea>
<input type="submit" name="SSC" value="Send Comment">
<input type="reset" value="Clear Comment">
</form>
```

Perl script to process form (part 1)

```
#!/opt/bin/perl
require "cgi-lib.pl";
$COMMENT_DIR = "insert an absolute path name here";
$COMM_FILE = "testcomm.dat";
$COMM_LOG = "testcomm.log";
print "Content-type:
  text/html\n\n<html>\n<head>\n<title>".
"COMMENT</title>\n</head>\n";
#Parse the form arguments and exit if error
&ReadParse(*values);
#Use the information
#If any field is missing, complain!
if ((! $values{"name"}) || (! $values{"subject"}) ||
  (! $values{"comment"}))
{ print "<body>message\n</body></html>\n";
  exit 0;}
```

Perl script to process form (part 2)

```
#OK, we have all the data. Write it to a file
$lf_name = ">>" . $COMMENT_DIR . $COMM_LOG;
$cf_name = ">>" . $COMMENT_DIR . $COMM_FILE;
$time = localtime();
$values{"subject"}=~s/\t/ /g;
$values{"name"}=~s/\t/ /g;
$startpos = (-s $COMMENT_DIR . $COMM_FILE);
if (!$startpos) {$startpos=0;}
open(OUT, $cf_name);
print OUT "DATE:\t", $time, "\n";
print OUT "AUTHOR:\t", $values{"name"}, "\n";
print OUT "SUBJECT:\t ", $values{"subject"}, "\n";
print OUT "MESSAGE:\n", $values{"comment"}, "\n\n";
close(OUT);
```

Perl script to process form (part 3)

```
$length = (-s $COMMENT_DIR . $COMM_FILE) - $startpos;
open(LOG, $lf_name);
seek LOG, 0, 2;
print LOG  $time , "\t", $values{"name"}, "\t",
$values{"subject"}, "\t",$startpos, "\t", $length, "\n";
close (LOG);
print "<body><h1>Thank you, ", $values{"name"},
  "</h1>\n";
print "Thank you for your comments.\n";
print "<P><A HREF=\"http:\/\/augment.sis.pitt.edu\/cgi-
  bin\/comm_view.cgi\?start=",
  0, "&length=", 0,"\" >Click here to see a list of
  comments</A></LI>\n";
print "</body></html>\n";
exit 0;
```