CSE2395/CSE3395

Topic 8

Files and directories

Learning Perl 2nd edition
chapter 10, pages 108-115
chapter 12, pages 129-133
chapter 13, pages 134-141
Programming Perl 3rd edition
pages 20-22, 28-29, 97-100, 747-755, 770
Programming Perl 2nd edition
pages 12-14, 19-20, 85-87, 191-195
`perlfunc`, `perlopentut` manpages

1

CSE2395/CSE3395

Last time

Covered in Topic 7

- Subroutines (functions)
- Calling and returning from functions
- Passing arguments to functions
  - the `@_` array
- Localizing variables
  - `my` and `local` keywords

2

CSE2395/CSE3395

To be covered today

- Files
  - opening
  - closing
  - reading from
  - writing to
  - testing
  - renaming, deleting
- Directories
  - scanning
  - creating and removing

3

CSE2395/CSE3395

Standard I/O

Revision

- All programs have three filehandles open by default
  - `STDIN` (standard input)
    - buffered input, defaults to keyboard
  - `STDOUT` (standard output)
    - buffered output, defaults to console
  - `STDERR` (standard error)
    - unbuffered output, defaults to console
- Each may be independently redirected to a file or process by shell redirection

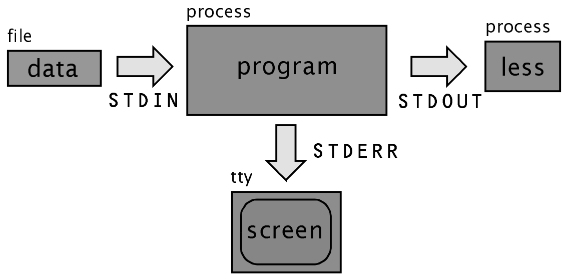
4

CSE2395/CSE3395

Standard I/O

Example

```
% program < data | less
```



5

CSE2395/CSE3395

Writing to `STDERR`

Using filehandles other than `STDOUT`

- `STDOUT` is the default filehandle for `print`
- `print` can use other filehandles
  - put filehandle name immediately after `print`
  - no comma between filehandle and first parameter
  - `print STDERR "Invalid data\n";`
  - same syntax is used for printing to manually opened filehandles
  - `warn` function is another way of printing to `STDERR`

Llama2 page 111; *Camel3* pages 21-22
Camel2 page 14; `perlfunc` manpage

6

Dealing with errors

`warn` and `die`

- `warn` function prints a message to `STDERR`
  - `warn "Near critical mass" if $temp > 2500;`
  - appends the program's name and line number unless the warning message ends in newline
- `die` function prints a message to `STDERR` and then exits the program
  - behaviour is otherwise the same as `warn`
  - `die "Reactor meltdown!" if $temp > 10000;`

Llama2 pages 109-111; *Camel3* pages 827-828, 700-701
Camel2 pages 241, 157; `perlfunc` manpage

Opening files

Going beyond `STDIN`, `STDOUT` and `STDERR`

- `open` function opens a file for reading or writing
  - associates a filehandle variable with the file
  - returns false and sets `$!` if open fails
- `close` function closes the filehandle and commits any changes to disk
  - all open filehandles are closed automatically at end of program
  - many Perl programs don't bother to close files explicitly

Llama2 pages 108-109; *Camel3* pages 20-22, 747-755, 693;
Camel2 pages 12-14, 191-195, 151-152
`perlfunc`, `perlopentut` manpages

Opening files

Anatomy of an `open`

filehandle should be entirely in capitals

```
open HANDLE, filename;
```

filename is a string:
`"file"` (read from file)
`"<file"` (read from file)
`">file"` (write to file)
`">>file"` (append to end of file)
other forms are possible

Llama2 pages 108-109; *Camel3* pages 747-755
Camel2 pages 191-195
`perlfunc`, `perlopentut` manpages

Using filehandles

Reading from and writing to a file

- To write to a filehandle, use `print HANDLE`
  - no comma between filehandle and arguments to `print`
  - `print OUTFILE "Some text\n";`
  - `print STDOUT "Filehandle here is redundant";`
- To read from a filehandle, use `<HANDLE>`
  - `$line = <INFILE>;`
  - `while (<INFILE>) { # Do things with $_ }`
  - `<HANDLE>` returns `undef` at end of file

Llama2 page 111; *Camel3* pages 20-22, 80-83, 765-766
Camel2 pages 13-14; `perlfunc` manpage

Example

Generating a file of prime numbers under 100

```
# "or" keyword is a very-low precedence
# version of the short-circuiting "||" operator.
# $! contains the last error message from a
# failed call to open, close or system.
open OUTFILE, ">primes"
  or die "Cannot open file for writing: $!";

# Sieve of Eratosthenes algorithm.
@is_prime = (1) x 100; # Array of 100 ones.
for ($i = 2; $i < 100; $i++) {
  if ($is_prime[$i]) {
    # Number $i is prime, print it.
    print OUTFILE "$i\n";
    # All multiples of $i are not prime.
    for ($j = $i * 2; $j < 100; $j += $i)
    { $is_prime[$j] = 0; }
  }
}
```

Example

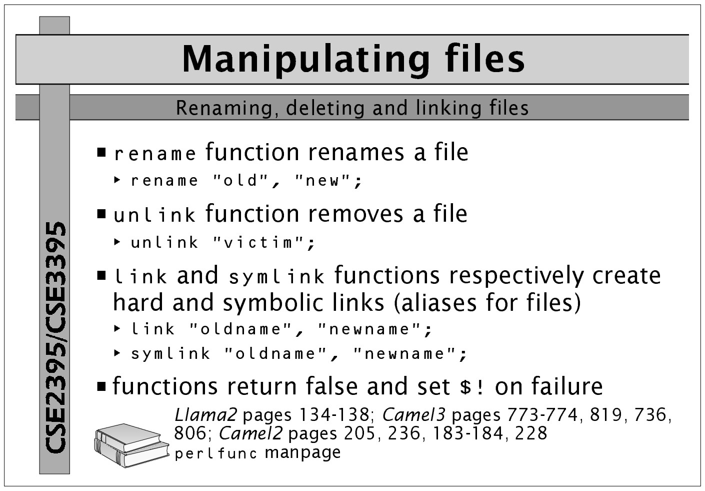
Reading from a file to determine prime factors

```
print "Enter a number to factor: ";
chomp ($number = <STDIN>);

open PRIMES, "<primes"
  or die "Cannot open file for reading: $!\n";

# Assign $_ to each line of primes file in turn.
while (<PRIMES>)
{
  chomp;
  # Apply this factor as often as possible.
  while ($number % $_ == 0) {
    push @factors, $_; $number /= $_;
  }
  last if $_ > $number;
}
close PRIMES; # Rarely done explicitly.

print join ("*", @factors), "\n";
```


CSE2395/CSE3395

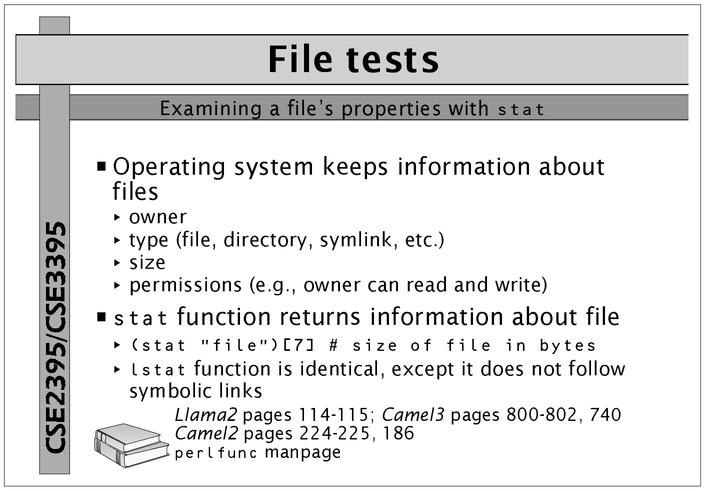
Manipulating files

Renaming, deleting and linking files

- `rename` function renames a file
  - `rename "old", "new";`
- `unlink` function removes a file
  - `unlink "victim";`
- `link` and `symlink` functions respectively create hard and symbolic links (aliases for files)
  - `link "oldname", "newname";`
  - `symlink "oldname", "newname";`
- functions return false and set `$!` on failure

Llama2 pages 134-138; *Camel3* pages 773-774, 819, 736, 806; *Camel2* pages 205, 236, 183-184, 228
`perlfunc` manpage

13


CSE2395/CSE3395

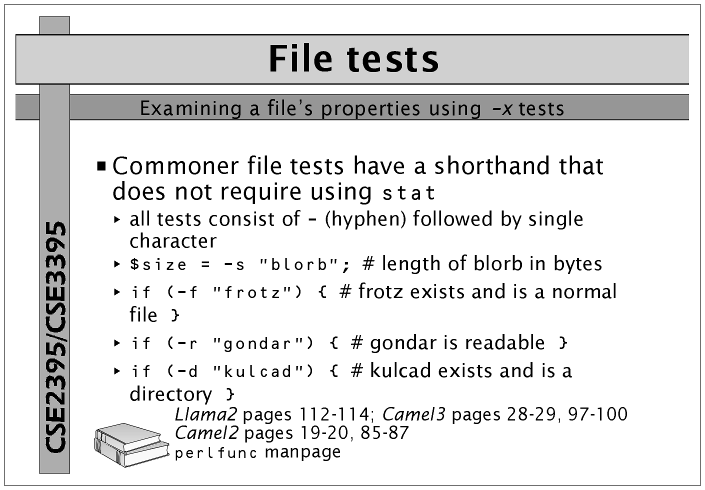
File tests

Examining a file's properties with `stat`

- Operating system keeps information about files
  - owner
  - type (file, directory, symlink, etc.)
  - size
  - permissions (e.g., owner can read and write)
- `stat` function returns information about file
  - `(stat "file")[7] # size of file in bytes`
  - `lstat` function is identical, except it does not follow symbolic links

Llama2 pages 114-115; *Camel3* pages 800-802, 740
Camel2 pages 224-225, 186
`perlfunc` manpage

14


CSE2395/CSE3395

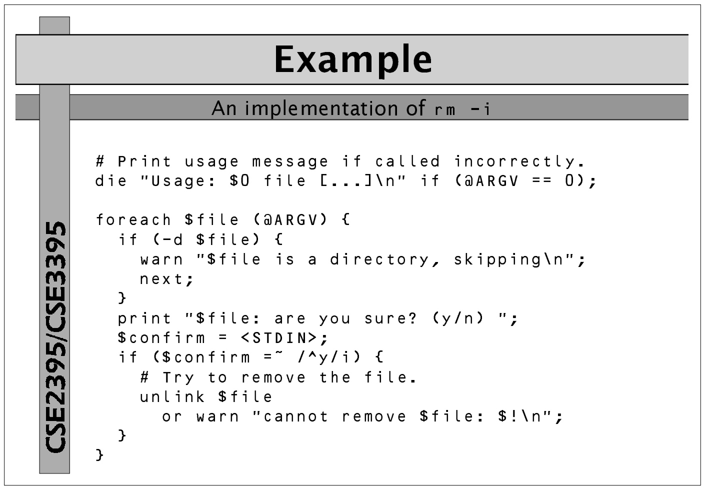
File tests

Examining a file's properties using *-x* tests

- Commoner file tests have a shorthand that does not require using `stat`
  - all tests consist of `-` (hyphen) followed by single character
  - `$size = -s "blorb";` # length of blorb in bytes
  - `if (-f "frotz") {` # frotz exists and is a normal file `}`
  - `if (-r "gondar") {` # gondar is readable `}`
  - `if (-d "kulcad") {` # kulcad exists and is a directory `}`

Llama2 pages 112-114; *Camel3* pages 28-29, 97-100
Camel2 pages 19-20, 85-87
`perlfunc` manpage

15


CSE2395/CSE3395

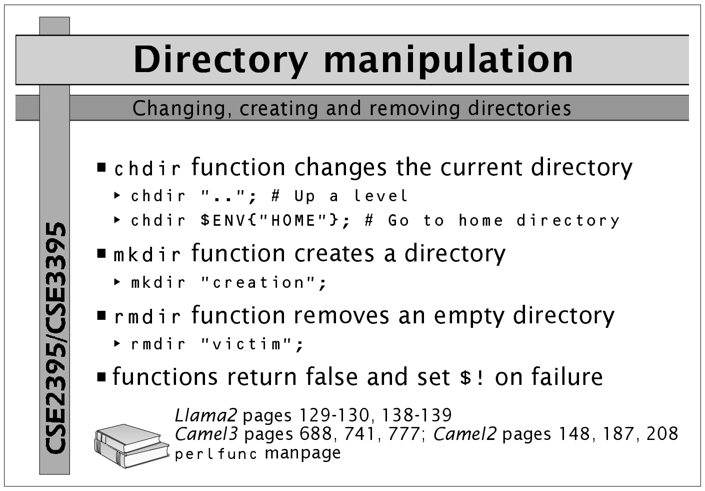
Example

An implementation of `rm -i`

```
# Print usage message if called incorrectly.
die "Usage: $0 file [...]\n" if (@ARGV == 0);

foreach $file (@ARGV) {
  if (-d $file) {
    warn "$file is a directory, skipping\n";
    next;
  }
  print "$file: are you sure? (y/n) ";
  $confirm = <STDIN>;
  if ($confirm =~ /^y/i) {
    # Try to remove the file.
    unlink $file
      or warn "cannot remove $file: $!\n";
  }
}
```

16


CSE2395/CSE3395

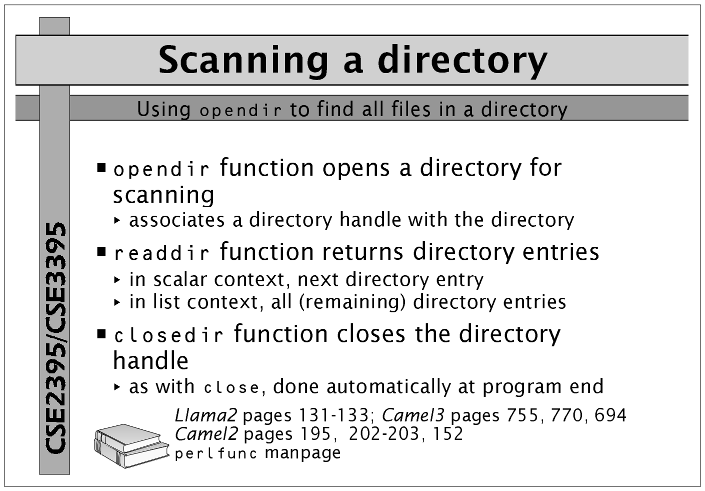
Directory manipulation

Changing, creating and removing directories

- `chdir` function changes the current directory
  - `chdir ".."; # Up a level`
  - `chdir $ENV{"HOME"}; # Go to home directory`
- `mkdir` function creates a directory
  - `mkdir "creation";`
- `rmdir` function removes an empty directory
  - `rmdir "victim";`
- functions return false and set `$!` on failure

Llama2 pages 129-130, 138-139
Camel3 pages 688, 741, 777; *Camel2* pages 148, 187, 208
`perlfunc` manpage

17


CSE2395/CSE3395

Scanning a directory

Using `opendir` to find all files in a directory

- `opendir` function opens a directory for scanning
  - associates a directory handle with the directory
- `readdir` function returns directory entries
  - in scalar context, next directory entry
  - in list context, all (remaining) directory entries
- `closedir` function closes the directory handle
  - as with `close`, done automatically at program end

Llama2 pages 131-133; *Camel3* pages 755, 770, 694
Camel2 pages 195, 202-203, 152
`perlfunc` manpage

18

Example

Renaming all normal files in the current directory

```
# Open the current directory
opendir HERE, "."
  or die "Cannot open directory: $!";

# Read each directory entry (scalar context)
while (defined ($name = readdir(HERE)))
{
  # Skip . (current) and .. (parent) directory
  # entries (redundant here because of -f test)
  next if $name =~ /^\.\.?$/;
  # Skip anything that's not a normal file
  next unless -f "./$name";

  rename "./$name", "./$name.bak"
    or warn "Cannot rename $name: $!";
}
```

Example

The same program using `readdir` in list context

```
opendir HERE, "."
  or die "Cannot open directory: $!";

# Read all files into @names (list context).
@names = readdir(HERE);
closedir HERE;

foreach $name (@names)
{
  # Skip . and .. entries
  next if $name =~ /^\.\.?$/;
  # Skip anything that's not a normal file
  next unless -f "./$name";

  rename "./$name", "./$name.bak"
    or warn "Cannot rename $name: $!";
}
```

Covered today

- Files
  - opening with `open`
  - closing
    - usually not required
  - reading from
    - `<HANDLE>`
  - writing to
    - `print HANDLE`
  - testing
    - `-f`, `-d`, `-x`, etc.
  - renaming, deleting
- Directories
  - reading directory entries
    - `opendir`, `readdir`
  - creating, removing

Going further

More things related to today's topic

- Exceptions
  - catching exceptions with `eval` and `die`
  - *Camel3* pages 700, 705-707; *Camel2* pages 157, 161-163
- `IO::File` and `IO::Dir`
  - object-oriented approach to files and directories
  - `man IO::File`, `man IO:Dir`
- Perl 5.6 three-argument `open`
  - separating the mode from the filename
  - *Camel3* pages 747-755

Next time

To be covered in Topic 9

- Processes
  - cooperating with other programs
- Formats
  - printing nicely-formatted reports

Reading:
Learning Perl 2nd edition chapters 11, 14, pages 116-128, 142-152
Programming Perl 3rd edition pages 234-241, 747-755, 426-428
Programming Perl 2nd edition pages 121-127, 191-195, 341-342
`perlform`, `perlopentut` manpages