CIS 232 – Hour 10

More File and Directory Functions

Directory Functions

- Opening a directory equivalent to opening a file:
  opendir (directoryfilehandle, ‘directory name’)

- Same with closing a directory:
  closedir (directoryfilehandle);

- However, reading a directory has a specific command:
  readdir (directoryfilehandle);
  and returns FALSE at the end of the directory meaning we can use readdir in a foreach loop (or any other loop construct).

- Any running perl script has a "working directory" value, generally the directory where the Unix shell executed the script. The chdir function changes this working directory to the path specified as it's argument.

- **chdir(‘/directorypath’);**

Directory Functions

- Example: Directory listing
- **Exercise61:**

```
#!/usr/bin/perl
use strict;
use warnings
chdir($ENV{HOME});     my $directory = '.';
opendir (DIR, $directory) || die $!;
while (my $file = readdir(DIR)) {
# Use a regular expression to ignore files beginning with a period
        next if ($file =~ m/^\./);        print "$file\n";     }
closedir(DIR);
```

- Note use of the pragma **use warnings**; same as –w on the perl command.

Directory Functions

- • Finding directories
- Example: Directory listing of directories only using file test operators: (here’s a more complete list)
- **Exercise62**

```
#!/usr/bin/perl
use strict;
use warnings;
chdir($ENV{HOME});     my $dir = '.';
opendir(DIR, $dir) or die $!;
while (my $file = readdir(DIR)) {
# A file test to check that it is a directory (-d)  or to test for a file (-f)
        next unless (-d "$dir/$file");
        print "$file\n"; }
closedir(DIR);
```

Directory Functions

- Find files ending in...
- Example: Looking for files ending it .txt
- **Exercise63:**

```
#!/usr/bin/perl
use strict;
use warnings;
chdir($ENV{HOME});
my $dir = '.';
opendir(DIR, $dir) or die $!;
while (my $file = readdir(DIR)) {
    # We only want files
    next unless (-f "$dir/$file");
    # Use a regular expression to find files ending in .txt
    next unless ($file =~ m/\.txt$/);
    print "$file\n";  }
closedir(DIR);
```

Directory Functions

- Finding specific files
- Example: use grep to filter out the files you want. (not directories) beginning with a period from the open directory. The filenames are found in the array @dots.
- **Exercise64:**

```
#!/usr/bin/perl
use strict;
use warnings;
chdir($ENV{HOME});
my $dir = '.';
opendir(DIR, $dir) or die $!;
my @dots = grep { /^\./            # Begins with a period
          && -f "$dir/$_"          # and is a file
          } readdir(DIR);
foreach my $file (@dots) {
          print "$file\n"; }
closedir(DIR);
```

Directory Functions

- • Filename “globbing”
- Another way of getting a directory listing is the “glob” function; if you're only interested in the current directory and not in any sub-directories. You pass glob a pattern (or patterns) to match and it will return any files that match in the current directory:
- **Exercise65:**

```
#!/usr/bin/perl
use strict;
use warnings;
chdir($ENV{HOME});
my $dir = '.';
my @files = glob("*.txt");
foreach my $file (@files) {
        print "$file\n";  }
```

File Functions

- **Copy a File – copy()**

- Using the copy function we can duplicate the file. Copy takes two arguments, the name of the file to be copied and the name of the new file/directory to which the file is to be copied.. If the same file name is used , PERL rewrites over the file if permissions is allowed. Note the use of the File::Copy Perl module.

- **Exercise66:**

```
#!/usr/bin/perl
use File::Copy;
$filetobecopied = "exercise3";
$newfile = "execrcise3y";
copy($filetobecopied, $newfile) or die "File cannot be copied: $!.";
```

File Functions

- **Move a file – move() or rename()**

- Moving the file requires the use of the "move" function. This function works similarly to the copy function from above and we send the same module to PERL. The difference is here is, instead of copying we just 'cut' the file and send it to a new location. In UNIX, this is the same as a DOS/Windows RENAME command.

- **Exercise67:**

```
#!/usr/bin/perl
use File::Copy;
$oldlocation = "exercise3y";
$newlocation = ”exercise3z";
move($oldlocation, $newlocation) or die "File cannot be renamed: $!”;
```

- Note alternative syntax using the rename function:

```
rename(oldlocation, $newlocation);
```

File Functions

- **Deleting files - unlink()**

  To delete specific files use the "unlink" function. The best way is often to set a variable equal to name of the file you wish to delete.

- **Exercise68:**

  ```
  #!/usr/bin/perl -w
  $file = "exercise3z";
  if (unlink($file) == 0) {
  print "File deleted successfully.";
  } else { print "File was not deleted.";  }
  ```

- **Removing Multiple Files at Once**

  To remove multiple files at once, create an array of files that has to be deleted and then loop through each one. There are many other ways to go about this process. Notice possibility of combining with the “glob” function.

- **Exercise69:**

  ```
  #!/usr/bin/perl -w
  @files = ("source.txt","destination.txt");
  foreach $file (@files) { unlink($file); }
  ```

File Functions

- **File statistics – stat()**

- The stat function in perl returns an array with information about the specified file or file handle.

[0] dev	device number of filesystem
[1] inode	inode number
[2] mode	file mode (type and permissions)
[3] nlink	number of (hard) links to the file
[4] uid	numeric user ID of file's owner
[5] gid	numeric group ID of file's owner
[6] rdev	the device identifier (special files only)
[7] size	total size of file, in bytes
[8] atime	last access time in seconds since the epoch
[9] mtime	last modify time in seconds since the epoch
[10] ctime	inode change time in seconds since the epoch (*)
[11] blksize	preferred block size for file system I/O
[12] blocks	actual number of blocks allocated

File Functions

- **Exercise70:**

```
#!/usr/bin/perl -w
$testfile='exercise3';
@file_info=stat($testfile);
print @file_info;
print "\n";
$fsdev = $file_info[0];         # device number of filesystem
$inode = $file_info[1];         # inode number
$mode  = $file_info[2];         # file mode  (type and permissions)
$nlink = $file_info[3];         # number of (hard) links to the file
$uid   = $file_info[4];                 # numeric user ID of file's owner
$gid   = $file_info[5];                 # numeric group ID of file's owner
$rdev  = $file_info[6];         # the device identifier (special files only)
$size  = $file_info[7];         # total size of file, in bytes
$atime = $file_info[8];         # last access time in seconds since the epoch (1970)
$mtime = $file_info[9];         # last modify time in seconds since the epoch (1970)
$ctime = $file_info[10];        # inode change time in seconds since the epoch (*)
$blksize = $file_info[11];      # preferred block size for file system I/O
$blocks = $file_info[12];       # actual number of blocks allocated
print "File size for $testfile is $size bytes\n";
# another method for showing the file size
print "File size for $testfile is ". (stat($testfile))[7]." bytes\n";
```

Finally

- Note use of pragma and Perl modules from examples.
- File operations are limited to the permissions you have on the system scripts are executed.
- Be careful combining examples of file “glob” and “unlink” commands - especially in looping structures.
- **Lab 9:**

  Create a perl script that does the following:

  1. Change the current directory to /tmp
  2. Reads the contents of the directory /tmp
  3. Print all directory names, the permissions, user ID group ID, and last access date.