

CS306: Introduction to Perl

Ch. 10: Interfacing with the OS

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

The Environment

- Both Windows and Unix maintain a set of environment variables for the user's shell.
- ```
[fran@franlt ~]$ env
HOSTNAME=franlt
SHELL=/bin/bash
USER=fran
PWD=/home/fran
HOME=/home/fran
[fran@franlt ~]$
```

Slide 2

## Reading the Environment in Perl

- ```
[fran@franlt ~]$ perl -e '@arr=keys %ENV; print
"%ENV Keys: @arr\n";'
%ENV Keys: SSH_ASKPASS XAUTHORITY
GDM_XSERVER_LOCATION LESSOPEN SSH_AUTH_SOCK PWD
LANG USER G_BROKEN_FILENAMES LOGNAME
GNOME_DESKTOP_SESSION_ID SHLVL INPUTRC PATH
WINDOWID COLORTERM HISTSIZE TERM HOME
DBUS_SESSION_BUS_ADDRESS SSH_AGENT_PID DISPLAY
GTK_RC_FILES GDMSESSION MAIL_QTDIR
GNOME_KEYRING_SOCKET HOSTNAME _DESKTOP_SESSION
DESKTOP_STARTUP_ID LS_COLORS SHELL KDEDIR
SESSION_MANAGER
```

Slide 3

File Globbing

- Similar to Unix or Windows wildcard file listing ('ls *.pl' or 'dir *.pl')
- In perl, we use the <> for globbing, usually in list context:

```
my @perlfiles = <*.pl>;
my @officefiles = <*.doc *.xls *.ppt>;
```

Slide 4

File Globbing

- Useful in a loop to process files:

```
while (<*.pl>) {  
    print "The next perl file is $_\n";  
}
```
- Remember to weed out special files:

```
while (<*>) {  
    next if $_ eq "." or $_ eq "..";  
    # process regular files now  
}
```

Slide 5

File Tests

- `while (<*>) {`

```
    print "$_ is a directory\n" if -d $_;  
    print "$_ is readable\n" if -r _;  
    print "$_ is writable\n" if -w _;  
    print "$_ is executable\n" if -e _;
```
- `_` is another shortcut – it means “the last file that was tested” - saves the OS from having to actually stat the file each time

Slide 6

Directories with `opendir()`

- `opendir` *DH*, “/home/fran” or die “Error: \$!”;

```
while ($file = readdir(DH)) {  
    next if $file eq "." or $file eq "..";  
    open FH, "<$file";  
    while ($line = <FH>) {  
        print $line;  
    }  
}  
closedir(DH);
```

Slide 7

Functions for Files and Directories

- `chdir(directory)` – Change the current working directory to *directory*. Without *directory* defaults to `$ENV{HOME}`.
- `unlink(file)` – This is Perl's 'rm' or 'del'.
- `rename(old, new)` – Rename a file. ALSO for moving files – `rename(olddir/file, newdir/file)`
- `link(), symlink(), readlink()` - `symlink` functions

Slide 8

Functions for Files and Directories

- `mkdir(directory, mode)` – `mode` expects unix style mode definitions as an octal number e.g. 0755 for owner-readwriteexecute, group-readexecute, world-readexecute.
- `chmod(file, mode)` – change permissions on a filesystem object (*file* can be a file or a directory)

Slide 9

Executing External Programs

- `system(command)` – Runs the command as if the user typed in *command* at the shell. If *command* writes to STDOUT, output will appear. `system()` returns the error status of the command – note that in Unix and Windows an error status of 0 usually means success and non-zero means failure.

Slide 10

Warning about system()

- Two warnings about `system()`...
 - It is a dangerous function that can be susceptible to arbitrary command injection! Do not blindly pass user input as the parameter for `system()`.
 - It is an inefficient function – it launches an entire shell. Use perl built-ins when available (e.g. `unlink()` for deleting files instead of a `system()` call)

Slide 11

The backticks

- If you want to capture the output of the external command for further processing in Perl, use the backticks
- `my $output = `command`;`
or
`my @linesofoutput = `command`;`

Slide 12