A selection of File and process management ideas that my be useful for your projects

Time stamps

- **`localtime`** returns the local time on a machine (seconds since midnight)

```
$now = localtime();
```

- Use **`localtime`** in list context to get human readable values

```
($sec,$mn,$hr,$dy,$mn,$yr,$wday,$yday,$isdst) = localtime();
```

- Use **`time`** to get a *timestamp* (on Unix systems it's the number of non-leap seconds since January 1, 1970 UTC).
  It doesn't role over at midnight!

- Look at the functions in the **`Time::HiRes`** package for finer time resolution (microseconds?)

Moving around the directory tree

- change the current working directory with the **`chdir`** function
  (functionally the same as **`cd`** at the shell level)

```
chdir $dir or die "cannot chdir error: $!\n";
```

- **`$!`** is set to an error message if **`chdir`** fails
- **`chdir`** goes to 'home' directory if no directory is specified.

file name lists

- Use **`glob`** to quickly get a *list* of filenames in the current working directory

```
@files = glob “*.txt”;
```

- The string can contain anything that you would type in an **`ls`** command, e.g.

```
ls *.txt
```

File Handles

- Old-style file handles

```
my $name = "foo.txt";
open FH, $name or die "Cannot open $name\n";
while (my $line = < FH >) {
       print "$line";
}
close FH;
```

- Starting with perl v. 5.6 you can use file handle *references* instead of *bare word* file handles. This is useful because now you can save or process file handles like variables

```
my $name = "foo.txt";
open my $fh, $name or die "Cannot open $name\n";
while (my $line = < $fh >) {
       print "$line";
}
close $fh;
```

Testing files

- **`-e`** test for existence of a file

```
if( -e $filename) {
      print "$filename exists!\n";
}
```

- Example: polling
- See "Learning Perl" p. 159 for other tests.

Directory Handles

- Directories can be opened for reading and accessed via *Directory handles*

```
my $dirName = "/users/fernando"; # my home directory
opendir HD, $dirName or die "Cannot open $dirName\n";
foreach my $file (readdir HD) {
       print "$file\n";
}
closedir HD;
```

removing files

- Remove files with the perl **`unlink`** function (functionally the same as **`rm`** at the shell level)

```
unlink $filename;
```

renaming files

- Rename files with perl **`rename`** function (functionally the same as **`mv`** at the shell level)

```
rename $oldname, $newname;
```

Process management
(how to control other programs)

How to launch processes

- **`system`**
- `` ` ` `` (*backtics*)
- **`exec`**
- **`fork`**

system

- **system** function executes a string as if were typed as a linux command. **STDIN**, **STDOUT** and **STDERR** are all inherited from perl.

```
$home = "~";
system "ls -l $home";
```

- **system** creates another process which executes the command.

` ` *backtics*

- **`system`** sends its output to **`STDOUT`** and **`STDERR`** of the perl script.
- To capture the output of a command for further processing use *backtics* instead of **`system`**.

```
my @results = `ls -l $home`;
foreach my $line (@results) {
        if($line =~ /\d\sfernando/) {
                print "$line";
        }
}
```

exec

- **exec** is not used much in practice.
- It is similar to **system** except that no new process is started.
- The command is executed in the same process as perl.
- This has the effect of losing the perl process.
- You can't continue executing perl statements after the **exec** statement.

`sleep`

- Sometimes you want to put your perl script to sleep for a fixed period of time before doing something, e.g. submitting a process, or poling for the occurrence of something.

Example: measuring the latency of the ls command

```
#!/usr/local/bin/perl
# measure the latency of 'ls' command
use Time::HiRes qw(gettimeofday tv_interval);
while(1){
        $start = [&gettimeofday]; # an array reference
         system('ls');             # execute the ls command
        $stop =[&gettimeofday];   # an array reference

         # tv_interval expects array references
        $diff = tv_interval($start,$stop);


        ($sec,$min,$hr)=localtime(time);
        $frac = (60*$min+$sec)/3600;
        $frac = int(100000*$frac)/100000;
        $hrs = $hr+$frac;
        print "$hrs\t$diff\n";

        sleep 5;
}
```

Example: polling for, e.g. existance of file

```
#!/usr/bin/perl
# test for the existance of a file and do something

sub doSomething {
        print "file  exists ... bye-bye\n";
        exit;
}


while(!(-e "test")){
        sleep 5;
}


doSomething();
```

Formatted output

- Use printf when you want to finely control your output format.
- The `printf` function prints a list of values according to the format template specified in a *format string*.
- `Example:`

```
printf("The values are: %d and %10.2f\n",  $x, $y)
```

  - The format string contains two *conversions*
  - %d tells perl to print `$x`as in integer
  - %10.2f tells perl to print `$y` as a float in 10 spaces and 2 decimal places
- See "Learning Perl" for more...

Conversions

- Conversions
  - %d integer (*truncates* floats)
  - %f floating point
  - %g general (integer, floating point or exponential)
  - %s string

- Modifiers
  - Fixed length field width
    - Insert width after %, e.g. %10f
  - Justification
    - Right justified in field by default
    - Left justified by using negative field width, e.g. %-10.f
  - Rounding
    - e.g. to round to 2 decimal places use%10.2f

- To print a percent sign use %%

email

- use **email** to notify yourself of job completion.

```
$to = "fernando.pineda\@jhu.edu";
$subject = “the script completed;
$msg = `ps -aux`;
`echo $msg | mail -s '$subject' $to`;
```

- e.g. VERIZON wireless customers can send text messages to their cell phones. If your phone number is **xxxxxxxxxx**, your email address is:

  xxxxxxxxxx@vtext.com

- Sign up here:

  http://www.vtext.com