Useful Modules

Laziness is a virtue!

Standard Modules

- Perl comes with a plethora of standard modules that you can use in any script.
- Nothing needs to be downloaded or installed separately to use any of them.
- **`perldoc perlmodlib`** for the full list of standard modules included in your version of Perl
- For more info on any of these:
  `perldoc Modulename`

integer

- A "pragma" or "pragmatic module" (similar to **`strict`** or **`warnings`**)
- for the duration of its lexical scope, do all numeric operations in integer context:
- ```
  print 7/3, "\n";
  {
     use integer;
     print 7/3, "\n";
  }
  ```
- 2.33333333333
  2
  ```

List::Util

- Provides several commonly used subroutines for dealing with lists of data.
- Does not export any subroutines by default. Specifically import the one(s) you want:
- `use List::Util qw/min max sum/;`
- `my @nums = (5, 3, 10, -4, 2);`
-
```
my $min = min @nums;
my $max = max @nums;
my $sum = sum @nums;
```
  - -4, 10, 16, respectively
- Also: **`minstr`** and **`maxstr`** for string-comparisons

more List::Util

- `use List::Util qw/first shuffle/;`
- `my @nums = (-4, -2, 1, 4, 5);`
- `my $pos = first { $_ > 0 } @nums;`
  - similar syntax to map/grep,
  - stops after first one found
    - `my ($pos) = grep { $_ > 0 } @nums`
    - would have same effect, but would keep searching through all elements even after first element found.
- `my @unsorted = shuffle(@nums);`
  - randomizes elements in @nums

File::Find

- Recursively search a filesystem
- exports one subroutine: **`find()`**.
- Takes a reference to a subroutine to call, and a list of starting directories.
  - the referenced subroutine will be called for each file/directory found within those directories
- Within your subroutine, File::Find sets several variables:
  - **`$_`** → name of the current entry (either file or directory)
  - **`$File::Find::name`** → full path of the current entry
  - **`$File::Find::dir`** → directory in which current entry is located
- Current working directory is also automatically changed to **`$File::Find::dir`**

find() example

- `use File::Find;`
- `my @unreadable;`
- `find (\&wanted, '/home', '/web');`
- `sub wanted {`
  `if (-f $_ and !-r _){`
  `push @unreadable, $File::Find::name;`
  `}`
  `}`
- `print "Unreadable files:\n",`
  `join("\n", @unreadable), "\n";`
- Searches /home and /web, including all subdirectories, and adds the full path of any files that are not readable to the **`@unreadable`** array.

File::Basename

- split a full path into the directory and file
  - and optionally, the extension
- `use File::Basename;`
- `my $path = '/foo/bar/baz.pl';`
- `my ($file, $dir) = fileparse($path);`
  or
  `my $file = basename $path;`
  `my $dir = dirname $path;`
  - **`$dir`** → '/foo/bar', **`$file`** → 'baz.pl'
- `my($f,$d,$s)=fileparse($path, qr/\..*/);`
  - **`$d`** → '/foo/bar/', **`$f`** → 'baz', **`$s`** → '.pl'

File::Copy

- copy or move files/directories
- exports two functions: copy and move
- `copy $file1, $file2 or die …;`
  `copy $file1, $dir or die …;`
- `move $file1, $file2 or die …;`
  `move $file1, $dir or die …;`

Getopt::Long

- Allow users of your program to easily set command line options
- Exports one function: **`GetOptions`**
- takes a hash where key determines name and type of option, and value is a ref to a variable that will hold the option
- `my ($length, $file, $verbose);`
- ```
  GetOptions(
     'length=i' => \$length,
     'file=s' => \$file,
     'verbose' => \$verbose
  )   or die "Invalid params";
  ```
- `./prog1.pl --length 5 --file foo.txt -v`

GetOptions keys

- The option names listed have three parts.
  - name of the option
  - **`=`** for required to take a value, **`:`** for optional value
  - type of value (**`s`** => string, **`i`** =>int, **`f`** =>float)
- If an option does not take a value, the variable is set to 1 if the option is provided.
- If an option's optional value is not given, strings get **`''`**, and ints/floats get **`0`**.
- If an option is not given, the variable's value is not changed.
  - usually means it will remain **`undef`**, unless you've given it a different value.

Getopt::Long errors

- **`GetOptions`** will fail if any of:
  - An unknown option is given on the cmd line
  - An option that requires a value is not given one
  - An option that requires a numeric value is given a string value
- **`GetOptions`** will NOT fail if:
  - Any of the options listed are not given on the command line.
    - They're OPTIONS, not "requireds"
  - If you want to test for this possibility, you need to verify for yourself that the variables were all defined.
  ```

Net::FTP

- Simple interface to ftp
- ```
  use Net::FTP;
  my $ftp = Net::FTP->new(
                'ftp.example.com');
  $ftp->login('user', 'pass');
  ```
- ```
  $ftp->cwd('public/downloads');
  $ftp->get('file.txt');
  ```
- ```
  $ftp->cwd('uploads');
  $ftp->put('upload.txt');
  ```

POSIX

- defines several useful subroutines
  - exports none by default – request the ones you want
- **`ceil($x)`** - returns smallest int greater than $x
- **`floor($x)`** - returns largest int less than $x
- **`asin($x)`** - arcsine of $x
- **`acos($x)`** - arccosine of $x
- **`atan($x)`** - arctangent of $x

POSIX, continued

- **`strftime`** – "string from time"
- Takes a format string, and a list of values representing the time
  - values are same as those returned by **`localtime`**()
- ```
  use POSIX 'strftime';
  my $now = strftime(
  '%Y-%m-%d %H:%M:%S', localtime());
  ```
  - "2009-03-25 16:20:14"
- ```
  my ($m, $d, $y) = (3, 25, 2009)
  ```
- ```
  my $later = strftime('%m/%d/%Y',
  0, 0, 0, $d, ($m-1) + 10, $y-1900);
  ```
- in addition to **`perldoc POSIX`**, also look at **`man strftime`**

Text::Wrap

- wrap long lines of text
- `my $first_tab = "\t";`
  `my $rest_tabs = "";`
- `my @wrapped = wrap(`
  `$first_tab, $rest_tabs, @lines`
  `);`
- Returns a list of lines formatted to no more than **`$Text::Wrap::columns`** long

CPAN

- Comprehensive Perl Archive Network
- Repository of thousands of non-standard modules.
  - Download the tarball, gunzip, tar xvf, perl Makefile.PL PREFIX=~/, make, make test, make install
- CPAN.pm – standard module to install modules
- type **`cpan`** at command line
  - first time, tell it you're NOT ready for manual config
  - **`o conf makepl_arg PREFIX=~/`**
  - **`o conf mbuildpl_arg --install_base=~/`**
  - **`o conf prerequisites_policy follow`**
  - **`o conf commit`**
    - if commit gives an error, try this instead:
      `o conf commit ~/.cpan/CPAN/MyConfig.pm`
  - To install a module: **`cpan ModuleName`**

List::MoreUtils

- Provides some additional, but lesser used list utilities
- **`any`**, **`all`**, **`none`**
  - **`print "All defined\n"`**
    `if all { defined $_ } @items;`
- **`true`**, **`false`**
  - **`my $positives = true {$_ > 0} @nums;`**
- **`uniq`**
  - **`my @items = (5, 4, 1, 4, 1, 6, 5, 2);`**
  - **`my @unique = uniq(@items);`**
    - (5, 4, 1, 6, 2)
- Beware the naming: **`List::Util`** vs **`List::MoreUtils`**

File::Stream

- Recall that **$/** is the "input record separator". It defines Perl's concept of a "line" for the **<>** operator.
- **$/** can be any string at all... but only a string
- until you download and use **File::Stream**
- ```
  use File::Stream;
  open my $fh, '<', 'file.txt';
  my $stream = File::Stream->new($fh);
  ```
- ```
  local $/ = qr/\d+/;
  while (<$stream>) { ... }
  ```
- records are now delimited by a series of digits

Regexp::Common

- many many common regular expressions. Implemented as a giant hash of regexps:
- **if (/$RE{num}{int}/) {...}**
  - integer
- **if (/$RE{comment}{C}/) {...}**
  - C-style comment
- **if (/$RE{profanity}/){...}**
  - any pre-defined profanity words
- **if (/$RE{URI}{HTTP}/) {...}**
  - a web address
- **if (/$RE{time}{mdy}/) {...}**
  - A month-day-year pattern
  - **Regexp::Common::time** must be installed separately.
- Use additional **{-keep}** key to set **$1**, **$2**, **$3**,...
  - See relevant perldocs for descriptions

Date::Parse

- **Regexp::Common::time** is good for *finding* dates, but not really suited for getting usable values out of the date.
  - You still have to translate month name or abbreviation to numbers manually
- **Date::Parse** exports two subroutines
  - **str2time** takes a string, and returns a unix timestamp
  - **strptime** takes a string, and returns a 7-element list (seconds, minutes, hours, days, months, year, timezone)
    - same as returned by **localtime**()

LWP::Simple

- LWP == libwww-perl
- Simplistic interface to retrieving websites
- **`use LWP::Simple;`**
  `my $g = get 'http://www.google.com';`
- **`getprint 'http://www.rpi.edu';`**
- **`getstore ('http://www.perl.com',`**
  `'perl.txt');`
- For more complicated techniques, see the base **`LWP`** module.

Mail::Send

- One of about a hundred different emailing modules.
- One of the simplest interfaces I've found
  - Labstaff graciously installed it system-wide for us.
- **`use Mail::Send;`**
  `my $mail = Mail::Send->new;`
- **`$mail->to('joe@example.com');`**
  `$mail->subject('Hey Joe');`
- **`my $fh = $mail->open();`**
  `print $fh "Hey Joe, what's up?";`
  `$fh->close;`

Documentation

- All of these modules (and many many more) can be found at http://search.cpan.org
- You can read the documentation there, or, after you've installed the module, simply run perldoc:
  `perldoc Mail::Send`
- If you get a message about no documentation found, tell Perl where it's located:
  `export`
  `PERL5LIB=$HOME/lib/perl5/site_perl/5.10.0`