Documenting Code

Plain Old Documentation (POD) markup language

User documentation

- Suppose that you have reached the point where you are creating useful programs and modules. Suppose too that you have been very diligent about documenting the source code so that someone reading your program will fully understand how it works.
- This is fine for the people who want to know ***how the program works***. What about the people who will just use your program? They do not need to know how the program works; they just need to know ***how to use the program***.
- They need a ***user manual***, an instruction manual that tells them how to use the program. This is called ***user documentation***.

User documentation for programs

- Programs can be used by non-programmers, ordinary ***end-users***. The documentation for a program should include how to run the program, which includes

  (1) what arguments are required,

  (2) what arguments are optional,

  (3) what options there are (switches, for example),

  (4) what output is produced and where (on standard output, any log files, where error messages go, etc.), and

  (5) the detailed answers to all of these questions.

User documentation for modules

- ***Perl programmers***, not end-users, use modules. Hence, the documentation for a module must include the module's ***programming interface,*** meaning:

  (1) the list of exported symbols (functions, constants, objects such as hashes or lists),

  (2) for each function, a description of all arguments and return values, and any error values and messages that might be produced by the function, and

  (3) any modules on which this module depends (so that the programmer knows to install those modules.)

Plain Old Documentation: POD

- The standard way to document a Perl program or module is to embed the documentation within the program itself in such a way that a special Perl program called ***perldoc*** can extract the documentation and construct a manual page from it.
- In order for perldoc to do this, the documentation must be written in a special markup language called ***POD***, which is the acronym for ***Plain Old Documentation***.
- A ***markup language*** is a language for annotating text in a way which is syntactically distinguishable from the text itself.

Markup languages

- Examples of markup languages include:
  - Editors' revision instructions on manuscripts: a system of indicating changes to text,
  - HTML (hypertext markup language): a system of formatting tags for text and hypertext,
  - photo and graphical artists markup language: a system for marking graphic art to indicate formatting and changes,
  - dozens of electronic markup languages created in the last ten years.

About POD

- Pod is a simple-to-use markup language used for writing documentation for Perl programs and modules.
- Translators are available for converting Pod to various formats like plain text, HTML, man pages, and more.
- Pod markup consists of three basic kinds of paragraphs: ***ordinary***, ***verbatim***, and ***command***.
- Pod is easy to learn and easy to use.

Basic POD rules

- Separate all paragraphs above and below by blank lines.
- Start all command and ordinary paragraphs at the left margin – no leading space of any kind!
- Although POD can be interspersed between Perl statements, it requires care to do so. Therefore, in the beginning, put all POD after the end of the program, but before any **`__DATA__`** cut the program might have.
- The very last line of any POD markup must be the **`=cut`** command, no leading space!

`=cut`

POD command paragraphs

- The basic commands in POD are:

=head1	**heading text**	***level 1 heading***
=head2	**heading text**	***level 2 heading***
=head3	**heading text**	***level 3 heading***
=head4	**heading text**	***level 4 heading***
=over		***starts a list***
=item		***a list item***
=back		***ends a list***
=cut		***ends pod***

Ordinary POD paragraphs

- Most paragraphs in your documentation will be ordinary blocks of text. You can type the text without any markup at all and with just ***one blank line before and after***. When it gets formatted, it will undergo minimal formatting, like being re-wrapped, maybe put into a proportionally spaced font, and possibly justified.
- Ordinary paragraphs can have special formatting tags, such as ***I<text>*** to italicize, ***B<text>*** to bolden, ***C<text>*** for code, and ***F<filename>*** for special highlighting of filenames, to name a few.

Verbatim paragraphs

- A verbatim paragraph is a ***what-you-see-is-what-you-get*** block.
- Verbatim paragraphs are usually used for presenting a code block or other text which does not require any special parsing or formatting, and which shouldn't be wrapped.
- To create a verbatim paragraph, make the first character a space or a tab. It will be reproduced exactly, with tabs assumed to be on 8-column boundaries. Formatting codes are not allowed inside them -- you can't italicize or anything like that.

Lists

- A list is started with the **`=over`** command. The **`=over`** command has an optional integer argument that specifies the indentation for the list items in ***ems***. The default is 4 if omitted.
- Each item begins with an **`=item`** command. The **`=item`** command should be followed by one of
  - an asterisk to create a bullet,
  - a number, for a numbered list, or
  - text, to create text labels instead.

Bulleted lists

- A bulleted list is illustrated below, but ***without the blank lines between paragraphs***, to make it fit in a slide.

```
=over
=item *
This is the first of a bulleted list.
=item *
This is the second bulleted item.
=item *
This is the third item.
=cut
```

Numbered lists

- In a numbered list, the numbers replace the asterisks. You must provide the correct numbers yourself – Perl will not compute them

```
=over
=item 1
This is the first of a bulleted list.
=item 2
This is the second bulleted item.
=cut
```

Labeled lists

- You can create labeled lists by placing the label after the **=item** command:

```
=over
=item Inputs
The inputs come in several forms...
=item Parameters
The program depends on the …
=item Outputs
The program writes to …
=cut
```

Components of user documentation

- A properly documented Perl program or module must contain the following four sections:

  `NAME`

  `SYNOPSIS`

  `DESCRIPTION`

  `AUTHOR`

- Other sections may be warranted in specific cases. You might need an **`OPTIONS`** section to describe in detail command-line options to a program, a **`SEE ALSO`** section in case there are related documents, and perhaps **`EXAMPLES`** to be very helpful.

The NAME section

- The purpose of the **NAME** section is to provide a one line description of the software, such as

```
random – assorted random object generation routines
```

- It must have the program or module name followed by the summary, and it is best to keep it short enough to fit on an unwrapped text line.

The SYNOPSIS section

- The **SYNOPSIS** section is where proper usage of the program/ module is specified. There is flexibility in how much to put here. At the very least, it should have the different ways to use the program or module, e.g.,

```
use random;
use random qw( randint randlist randsting );
```

or

```
gendata [-avgt] [-s <int>] file file …
```

- Sometimes it is a good idea to even show examples. Run **`perldoc Integer`** or **`perldoc bignum`** to see examples of this.

The DESCRIPTION Section

- The **`DESCRIPTION`** must provide enough detail so that a user can learn exactly how to use every feature and option of the software. It cannot be vague or ambiguous, but it need not be highly technical. It should use complete and correct English sentences, and provide ample examples.
- You should take a look at the man pages for some of the most detailed shell commands or Perl modules to get an idea of the level of detail required.

Escape sequences

- There are various character escape sequences, in case you want to put a symbol into the documentation that has special meaning in POD, such as < or >. These symbol escape sequences are

  ```
  E<lt>
  E<gt>
  E<verbar>
  E<sol>
  ```

- If the system supports it, you can use **`E<0`*`nn`*`>`** where ***nn*** is an octal code. It may not display properly.

Other translators

- Perl POD can be translated to **`html`** with the **`pod2html`** program:

```
pod2html -infile=myprog.pl > myprog.html
```

- will produce an html version of your user documentation, suitable for posting on a website, for example.
- The **`pod2man`** translator will produce a man page that can be installed in one of the directories that the man page viewer (man) searches:

```
pod2man myprog.pl > myprog.1
```

Help with POD

- Perl is bundled with a program called **`podchecker`** that can check the syntax of your POD. Running

```
    podchecker myprog.pl
```

- on a program with embedded POD will tell you if it has syntax errors and where they are.
- You can also consult the **`perlpod`** man page.