Toc, a multipass framework for troff

John Ankarström

11 July 2021

TABLE OF CONTENTS

Introduction

Toc is a very simple solution – in 50 or so lines – to the problem of generating tables of contents and other forms of forward references in troff documents. It consists of a shell script called *toc* and a macro package called *toc.tmac*.

The *toc* script is a wrapper around troff, passing the document through the typesetter three times. In every pass, it sets the values of a register and a string named *te* and *tf*, respectively.

The *toc.tmac* package defines two macros, named `te` and `to`. If the *te* register is non-zero, the `te` macro hands its argument to the `tm` request, which prints it on standard error, prefixed by "(toc)". If the *tf* string is non-empty, the `to` macro invokes the `so` request, reading and inserting lines from the file specified by *tf* into the troff document.

Operation

In the first pass, the *toc* script sets *te* = 1 and *tf* = empty. This enables the `te` macro, printing its arguments on standard error. The lines prefixed with "(toc)", printed on standard error, are written to the file *$g* with the "(toc)" prefix removed.

In the second pass, *toc* sets *te* = 1 and *tf* = *$g*. This enables the `to` macro as well, inserting the contents of *$g* into the troff source. The `te` macro is still activated and accordingly prints its arguments on standard error yet another time. The output is processed like earlier and written to the file *$h*.

In the third and final pass, *toc* sets *te* = 0 and *tf* = *$h*. This disables `te`, which means that nothing is printed on standard error, but `to` is still enabled, inserting the contents of *$h* into the document.

Theoretically, three is the number of passes that are necessary – and sufficient – for generating forward references, such as tables of contents. Two passes are not enough, as the generated references may push a referent to the next page, rendering the generated references incorrect. To account for the addition of the generated references, a third pass is needed.

In practice, however, *toc* has the ability to detect how many passes are needed and will never do more work than what is necessary.

Usage

Macro package

The `te` and `to` macros do not apply any formatting to or perform any processing of their input. The `te` macro prints its arguments on standard error verbatim. For example, the request

```
.te .nr &ref \n%
```

will print

```
(toc).nr &ref \n%
```

on standard error. The "(toc)" prefix is removed before the line is written to the temporary file.

When `to` is invoked, it will literally insert

```
.nr &ref 3
```

into the troff source, assuming `te` was invoked on the third page.

As such, *toc* provides the tools needed to create forward references, including tables of contents, but the exact formatting must be programmed by the user himself.

Script

The *toc* script is a wrapper around troff and any potential troff pre- or post-processors. On standard input, it expects the troff source to be processed. As arguments, it takes a shell command line to be evaluated on every pass. For example, the invocation

```
$ <example.t toc refer -p refs \| groff -C
```

passes the contents of *example.t* through the *refer* preprocessor and the *groff* implementation of troff. Note the escaped pipe character; because *toc* passes its arguments directly to */bin/sh*'s *eval*, arbitrary shell syntax is supported, as long as it is escaped.[1]

Note that input must be given on standard in; it cannot be provided as a filename to *refer* or *groff*.

Examples

Table of contents

```
.so toc.tmac
.eo
.de he
. ft B
. sp 1v
\$*
. br
. ft
. te .the \$*\t\n%
..
.de the
. ta 0 \n(.luR
. tc .
\$*
. tc
. br
..
.ec
.sp |1i
.to
.\" ...
.he First heading
.\" ...
.he Second heading
```

[1] This also means that whitespace in arguments is not properly preserved. If you need to include whitespace in the arguments to troff or a troff preprocessor, create a separate shell script and invoke *toc* on that.