Formatting for Gopher with GNU troff

Gopher is a text-first medium. Gopher clients are responsible for formatting the output that a user sees, and this usually consists of no more than wrapping lines at the terminal width. Even in a plain text environment, the design principles of contrast, repetition, alignment, and proximity play an important role. There are a number of ways to take plain text formatting beyond line-wrapping. There are processors like fold, par, and pandoc that will handle basic formatting. GNU troff gives us a rich set of commands for formatting text.

In the context of Gopher, formatting means wrapping lines at a predetermined length, justifying and filling output, inserting paragraph breaks, controlling indentation, centering text, and more. Troff (pronounced ‘tee-roff’) has a long history. [1] The first iterations of this tool were used to format text for output on teletypes. Later iterations were used to drive phototypesetters. Today, GNU troff comes with a variety of preprocessors and post processors and can output plain text, HTML, PDF, Postscript, and a variety of other formats.

Troff uses in-line formatting requests to format content. For the purpose of this discussion, there are two types of commands, primitives and macros. Primitives can be thought of as the native command set. Macros are composed of primitives and/or other macros. Adding troff formatting requests to a document is not unlike marking it up with HTML tags. Troff then processes the source and generates a separate output file.

GNU troff comes with mm and ms macro packages. These macro packages aim to make it easier for users to format documents. Increasing ease-of-use usually comes at the expense of flexibility and choice, and that's certainly true in this case. The authors of the mm, ms, and other macro packages wrote them with certain use-cases in mind. If you are formatting a memo for use at Bell Labs in 1976, the mm macros are very useful. Outside of that, it can be a struggle to format text just the way you want it using one of these macro packages. But we can use troff primitives and even write our own macros.

Because the macro packages didn't let me format my documents just so, I started formatting my documents using the troff primitives. Formatted this way, a short post might look like this:

```
.ll 67              \" Set the line length to 67 characters
.ce 2               \" Center the next two output lines
Centered Document Title
.br                 \" Look familiar? This inserts a line break.
Centered Document Subtitle
.sp 2               \" Insert two vertical spaces (aka blank lines).
Content goes here.
.sp 2
Another paragraph.
```

We can also control the indentation to format code snippets and block quotes.

```
.in +5              \" Increase indentation by five
.ll -5              \" Reduce line length by five
Block quoted content
.ll                 \" Restore previous line length value
.in                 \" Restore previous indent value
```

As you can see, this leads to a lot of magic numbers floating around our source files. If we want to left-justify headers or make the block quotes narrower, we have a lot of files to update. We need to be able to define these styles elsewhere and create aliases that refer to them.

In our example above, we start by setting the line length to 67 characters. Let's create an initialization macro that will hold these document-level settings. To create a macro, we use the ‘.de’ request.

```
.de IZ              \" Define a macro named IZ
.ll 67              \" Set the line length to 67
..                  \" End the macro definition
```

That's all there is to it. Now, we can include ‘.IZ’ at the top of each document and it will use our macro to set the line length. In the future, if we want to change the line length, we only need to change the definition of the IZ macro.

We can do something similar with our centered headings and sub-headings. This is a little different because we may want to have one or two sub-headings. To account for this variability, we are going to define a macro that accepts the heading and sub-heading strings as arguments, and uses conditional statements to produce the output.

```
.de H
.ce
\\$1
.if !"\\$2"" \{\
.br
.ce
\\$2\}
.if !"\\$3"" \{\
.br
.ce
\\$3\}
.sp 2
..
```

This macro just checks to see if the second and third arguments are there before creating requests to break the line and center the output. At the end, it inserts two blank lines, but we're repeating our magic number problem. Let's create a paragraph macro that will define what happens between paragraphs.

```
.de P               \" Define a macro called P
.sp 2               \" insert two blank lines
..
```

This is a simple macro, but it could be made to do a lot more, like indent the first line of the following text. Now we can update our H macro to use our P macro.

```
.de H
.ce
\\$1
.if !"\\$2"" \{\
.br
.ce
\\$2\}
.if !"\\$3"" \{\
.br
.ce
\\$3\}
.P
..
```

We're going to put all of our macro definitions in their own file called style.groff. Now that we have a “style sheet”, we can recreate the document from our first example like so:

```
.IZ
.H "Centered Document Title" "Centered Document Subtitle"
.                   \\" A line with just a dot has no effect,
.                   \\" but is useful for structuring documents
Content goes here.
.P
Another paragraph.
```

Once we have our source document with our content and formatting commands, we can use groff to ingest our source and produce formatted output. I'm skipping over some details, here, but the command will look

like this:

```
groff -Tascii style.groff <sourcefile> > <outputfile>
```

You could also leave off the redirection to a file and pipe the output to a pager. It's important to note that when given multiple file names, as we've done here, groff simply concatenates the files before processing them.

In my posts folder, I have a subfolder, groff, that contains all my source files. I have a small script I call publish that loops through all of the source files and creates formatted output in the posts folder.

```
#!
for f in groff/*.groff
do
  outfile="${f#groff/}"
  outfile="${outfile%.*}.txt"
  groff -Tascii style.groff "$f" > "$outfile"
done
```

There are many reference sources available for troff and friends. The man pages and info pages for groff and the macro packages are comprehensive. There are a number of excellent printed references as well:

B. Srinivasan, “UNIX Document Processing and Typesetting”, World Scientific, 1993

Narain Gehani, “Document Formatting & Typesetting on the UNIX System”, Silicon Press, 1987

Tim O'Reilly, Dale Dougherty, “UNIX Text Processing”, Hayden Books, 1987

References

1. http://troff.org
2. gopher://sdf.org/0/users/dbucklin/gopher_groff.groff
3. gopher://sdf.org/0/users/dbucklin/style.groff