THE LINUX MAN−PAGE−HOWTO

http://www.schweikhardt.net/

See further information on copying conditions below.

$Id: man_page_howto.html,v 1.20 2001/10/28 15:30:22 schweikh Exp schweikh $

Click here to browse the author's latest version of this document. Corrections and suggestions welcome!

This HOWTO explains what you should bear in mind when you are going to write on−line documentation −− a so−called man page −− that you want to make accessible via the `man`(1) command. Throughout this HOWTO, a manual entry is simply referred to as a man page, regardless of actual length and without sexist intention.

Table of contents

1) A few thoughts on documentation

Why do we write documentation? Silly question. Because we want others to be able to use our program, library function or whatever we have written and made available. But writing documentation is not all there is to it:

- Documentation must be accessible. If it's hidden in some non−standard place where the documentation−related tools won't find it −− how can it serve its purpose?
- Documentation must be reliable and accurate. There's nothing more annoying than having program behaviour and documentation disagree. Users will curse you, send you hate mail and throw your work into the bit bucket, with the firm intent to never install anything written by that jerk again.

The historical and well known way documentation is accessed on UNIX is via the man(1) command. This HOWTO describes what you have to do to write a man page that will be correctly processed by the documentation− related tools. The most important of these tools are `man`(1), `xman`(1x), `apropos`(1), makewhatis(8) and catman(8). Reliability and accuracy of the information are, of course, up to you. But even in this respect you will find some ideas below that help you avoid some common glitches.

2) How are man pages accessed?

You need to know the precise mechanism for acccessing man pages in order to give your man page the right name and install it in the right place. Each man page should be categorized in a specific section, denoted by a single character. The most common sections under Linux, and their human readable names, are:

```
Section The human readable name
   1    User commands that may be started by everyone.
   2    System calls, that is, functions provided by the kernel.
   3    Subroutines, that is, library functions.
   4    Devices, that is, special files in the /dev directory.
   5    File format descriptions, e.g. /etc/passwd.
   6    Games, self-explanatory.
   7    Miscellaneous, e.g. macro packages, conventions.
   8    System administration tools that only root can execute.
   9    Another (Linux specific) place for kernel routine documentation.
   n    (Deprecated) New documentation, that may be moved to a more appropriate section.
   o    (Deprecated) Old documentation, that may be kept for a grace period.
   l    (Deprecated) Local documentation referring to this particular system.
```

The name of the man page's source file (the input to the formatting system) is the name of the command, function or file name, followed by a dot, followed by the section character. If you write the documentation on the format of the `passwd' file you have to name the source file `passwd.5'. Here we also have an example of a file name that is the same as a command name. There might be even a library subroutine named passwd. Sectioning is the usual way to resolve these ambiguities: The command description is found in the file `passwd.1' and the hypothetical library subroutine in `passwd.3'.

> Sometimes additional characters are appended and the file name looks for example like `xterm.1x' or `wish.1tk'. The intent is to indicate that this is documentation for an X Window program or a Tk application, respectively. Some manual browsers can make use of this additional information. For example xman will use `xterm(x)' and `wish(tk)' in the list of available documentation.

Please don't use the n, o and l sections; according to the File System Standard these sections are deprecated. Stick to the numeric sections. Beware of name clashes with existing programs, functions or file names. It is certainly a bad idea to write yet another editor and call it ed, sed (for smart ed) or red (for Rocky's ed). By making sure your program's name is unique, you avoid having someone execute your program but read someone else's man page, or vice versa. Checking out the Linux Software Map (LSM) database on a program name is a place to start ensuring name uniqueness.

Now we know the name to give our file. The next decision is the directory in which it will finally be installed (say, when the user runs `make install' for your package.) On Linux, all man pages are below directories listed in the environment variable MANPATH. The doc−related tools use MANPATH in the same way the shell uses PATH to locate executables. In fact, MANPATH has the same format as PATH. Each contains a colon−separated list of directories (with the exception that MANPATH does not allow empty fields and relative pathnames −− it uses absolute names only.) If MANPATH is not set or not exported, a default will be used that contains at least the /usr/man directory. To speed up the search and to keep directories small, the directories specified by MANPATH (the so−called base directories) contain a bunch of subdirectories named `man<s>' where <s> stands for the one−character section designator introduced in the table above. Not all of the sections may be represented by a subdirectory because there simply is no reason to keep an empty `mano' subdirectory. However, there may be directories named `cat<s>', `dvi<s>' and `ps<s>' which hold documentation that is ready to display or print. More on this later. The only other file in any base directory should be a file named `whatis'. The purpose and creation of this file will also be described under paragraph 12). The safest way to have a man page for section <s> installed in the right place is to put it in the directory /usr/man/man<s>. A good `Makefile`, however, will allow the user to chose a base directory, by means of a `make` variable, *MANDIR*, say. Most of the GNU packages can be configured with the `--prefix=/what/ever` option. The manuals will then be installed under the base directory /what/ever/man. I suggest you also provide a way to do something similar.

With the advent of the Linux File System Standard (FS−Stnd), things became more complicated. The FS−Stnd 1.2 states that

> "Provisions must be made in the structure of /usr/man to support manual pages which are written in different (or multiple) languages."

This is achieved by introducing another directory level that distinguishes between different languages. Quoting again from FS−Stnd 1.2:

> "This naming of language subdirectories of /usr/man is based on Appendix E of the POSIX 1003.1 standard which describes the locale identification string −− the most well accepted method to describe a cultural environment. The <locale> string is:
> <language>[_<territory>][.<character−set>][,<version>]"

(See the FS−Stnd for a few common <locale> strings.) According to these guidelines, we have our man pages in /usr/man/<locale>/man[1−9lno]. The formatted versions should then be in /usr/man/<locale>/cat[1−9lno] of course, otherwise we could only provide them for a single locale. HOWEVER, I can not recommend switching to that structure at this time. The FS−Stnd 1.2 also allows that

> "Systems which use a unique language and code set for all manual pages may omit the <locale> substring and store all manual pages in <mandir>. For example, systems which only have English manual pages coded with ASCII, may store manual pages (the `man[1-9]` directories) directly in /usr/man. (That is the traditional circumstance and arrangement in fact.)"

I would not switch until all tools (like xman, tkman, info and many others that read man pages) can cope with the new structure.

3) How should a formatted man page look?

Let me present you an example. Below I will explain it in detail. If you read this as plain text it won't show the different typefaces (***bold*** and *italics*). Please refer to the paragraph "What are the font conventions?" for further explanations. Here comes the man page for the (hypothetical) `foo` program.

```
FOO(1)                    User Manuals                    FOO(1)



NAME
     foo − frobnicate the bar library

SYNOPSIS
     foo [−bar] [−c config−file ] file ...

DESCRIPTION
     foo  frobnicates the bar library by tweaking internal symbol
     tables. By default it parses all baz segments and rearranges
```

```
     them  in  reverse  order  by time for the xyzzy(1) linker to
     find them. The symdef entry is then compressed using the WBG
     (Whiz-Bang-Gizmo) algorithm.  All files are processed in the
     order specified.

OPTIONS
     -b   Do not write `busy' to stdout while processing.

     -c config-file
          Use the alternate system wide  config-file  instead  of
          /etc/foo.conf.   This overrides any FOOCONF environment
          variable.

     -a   In addition to the baz segments, also parse the  blurfl
          headers.

     -r   Recursive  mode.  Operates  as fast as lightning at the
          expense of a megabyte of virtual memory.

FILES
     /etc/foo.conf
          The system wide configuration file. See foo(5) for fur-
          ther details.
     ~/.foorc
          Per  user  configuration  file.  See foo(5) for further
          details.

ENVIRONMENT
     FOOCONF
          If non-null the full pathname for an  alternate  system
          wide foo.conf.  Overridden by the -c option.

DIAGNOSTICS
     The following diagnostics may be issued on stderr:

     Bad magic number.
          The input file does not look like an archive file.
     Old style baz segments.
          foo  can  only  handle  new  style  baz segments. COBOL
          object libraries are not supported in this version.

BUGS
     The command name should have been chosen more  carefully  to
     reflect its purpose.

AUTHOR
     Jens Schweikhardt <howto at schweikhardt dot net>
SEE ALSO
     bar(1), foo(5), xyzzy(1)

Linux                Last change: MARCH 1995                    2
```

Here's the explanation as I promised.

The NAME section

...is the only required section. Man pages without a name section are as useful as refrigerators at the north pole. This section also has a standardized format consisting of a comma−separated list of program or function names, followed by a dash, followed by a short (usually one line) description of the functionality the program (or function, or file) is supposed to provide. By means of `makewhatis`(8), the name sections make it into the `whatis` database files. `Makewhatis` is the reason the name section must exist, and why it must adhere to the format I described. In the `groff` source it must look like

.SH NAME foo \– frobnicate the bar library

The \– is of importance here. The backslash is needed to make the dash distinct from a hyphenation dash that may appear in either the command name or the one line description.

The SYNOPSIS section

...is intended to give a short overview on available program options. For functions this sections lists corresponding include files and the prototype so the programmer knows the type and number of arguments as well as the return type.

The DESCRIPTION section

...eloquently explains why your sequence of 0s and 1s is worth anything at all. Here's where you write down all your knowledge. This is the Hall Of Fame. Win other programmers' and users' admiration by making this section the source of reliable and detailed information. Explain what the arguments are for, the file format, what algorithms do the dirty jobs.

The OPTIONS section

...gives a description of how each option affects program behaviour. You knew that, didn't you?

The FILES section

...lists files the program or function uses. For example, it lists configuration files, startup files, and files the program directly operates on. It is a good idea to give the full pathname of these files and to make the install process modify the directory part to match user preferences: the `groff` manuals have a default prefix of /usr/local, so they reference /usr/local/lib/groff/* by default. However, if you install using 'make prefix=/opt/gnu' the references in the man page change to /opt/gnu/lib/groff/*

The ENVIRONMENT section

...lists all environment variables that affect your program or function and tells how, of course. Most commonly the variables will hold pathnames, filenames or default options.

The DIAGNOSTICS section

...should give an overview of the most common error messages from your program and how to cope with them. There's no need to explain system error error messages (from `perror`(3)) or fatal signals (from `psignal`(3)) as they can appear during execution of any program.

The BUGS section

...should ideally be non–existent. If you're brave, you can describe here the limitations, known inconveniences and features that others may regard as misfeatures. If you're not so brave, rename it the TO DO section ;–)

The AUTHOR section

...is nice to have in case there are gross errors in the documentation or program behaviour (Bzzt!) and you want to mail a bug report.

The SEE ALSO section

...is a list of related man pages in alphabetical order. Conventionally, it is the last section. You are free to invent other sections if they really don't fit in one of those described so far. So how exactly did you generate that man page? I expected that question, here's the source, Luke:

```
.\" Process this file with
.\" groff -man -Tascii foo.1
.\"
.TH FOO 1 "MARCH 1995" Linux "User Manuals"
.SH NAME
foo \- frobnicate the bar library
.SH SYNOPSIS
.B foo [-bar] [-c
.I config-file
.B ]
.I file
.B ...
.SH DESCRIPTION
.B foo
frobnicates the bar library by tweaking internal
symbol tables. By default it parses all baz segments
and rearranges them in reverse order by time for the
.BR xyzzy (1)
linker to find them. The symdef entry is then compressed
using the WBG (Whiz-Bang-Gizmo) algorithm.
All files are processed in the order specified.
.SH OPTIONS
.IP -b
Do not write `busy' to stdout while processing.
.IP "-c config-file"
Use the alternate system wide
.I config-file
instead of
.IR /etc/foo.conf .
This overrides any
.B FOOCONF
environment variable.
.IP -a
In addition to the baz segments, also parse the
blurfl headers.
.IP -r
Recursive mode. Operates as fast as lightning
at the expense of a megabyte of virtual memory.
.SH FILES
.I /etc/foo.conf
.RS
The system wide configuration file. See
.BR foo (5)
for further details.
.RE
.I ~/.foorc
.RS
Per user configuration file. See
.BR foo (5)
```

```
for further details.
.SH ENVIRONMENT
.IP FOOCONF
If non-null the full pathname for an alternate system wide
.IR foo.conf .
Overridden by the
.B -c
option.
.SH DIAGNOSTICS
The following diagnostics may be issued on stderr:

Bad magic number.
.RS
The input file does not look like an archive file.
.RE
Old style baz segments.
.RS
.B foo
can only handle new style baz segments. COBOL
object libraries are not supported in this version.
.SH BUGS
The command name should have been chosen more carefully
to reflect its purpose.
.SH AUTHOR
Jens Schweikhardt <howto at schweikhardt dot net>
.SH "SEE ALSO"
.BR bar (1),
.BR foo (5),
.BR xyzzy (1)
```

4) How do I document several programs/functions in a single man page?

Many programs (`grep`, `egrep`) and functions (`printf`, `fprintf`, ...) are documented in a single man page. However, these man pages would be quite useless if they were only accessible under one name. We cannot expect a user to remember that the `egrep` man page is actually the `grep` man page. It is therefore necessary to have the man page available under different names. You have several possibilities to achieve this:

1. have identical copies for each name.
2. connect all man pages using hard links.
3. symbolic links pointing to the actual man page.
4. use `groff`'s `source' mechanism provided by the `.so` macro.

The first way is obviously a waste of disk space. The second is not recommended because intelligent versions of the `catman` program can save a lot of work by looking at the the file type or contents. Hard links will prevent `catman` from being clever. (Note that `catman`'s purpose is to format all man pages so they can be displayed quickly.) The third alternative has a slight drawback: if flexibility is a concern, you have to be aware that there are file systems that do not support symbolic links. The upshot of this is that the Best Thing (TM) is using `groff`'s source mechanism. Here's how to do it: If you want to have your man page available under the names `foo' and `bar' in section 1, then put the man page in foo.1 and have bar.1 look like this:

```
.so man1/foo.1
```

It is important to specify the `man1/` directory part as well as the file name `foo.1' because when `groff` is run by the browser it will have the manual base directory as its current working directory (cwd) and groff interprets .so arguments relative to the cwd.

5) Which macro package should I use?

There are a number of macro packages especially designed for use in writing man pages. Usually they are in the groff macro directory /usr/lib/groff/tmac. The file names are tmac.<something>, where <something> is the argument to groff's –m option. Groff will use tmac.<something> when it is given the `–m <something>' option. Often the blank between `–m' and `<something>' is omitted so we may say `groff –man' when we are formatting man pages using the tmac.an macro package. That's the reason for the strange name `tmac.an'. Besides tmac.an there is another popular macro package, tmac.doc, which originated at the University of California at Berkeley. Many BSD man pages use it and it seems that UCB has made it its standard for documentation. The tmac.doc macros are much more flexible but alas, there are manual browsers that will not use them but always call groff –man. For example, all xman programs I have seen will screw up on man pages requiring tmac.doc. So do yourself a favor: use tmac.an –– use of any other macro package is considered harmful. tmac.andoc is a pseudo macro package that takes a look at the source and then loads either tmac.an or tmac.doc. Actually, any man page browser should use it but to this point, not all of them do, so it is best we cling to ye olde tmac.an. Anything I tell you from now on and concerning macros only holds true for tmac.an. If you want to use the tmac.doc macros anyway, here is a pointer to detailed information on how to use them: http://www.bsdi.com/bsdi–man There is a searchable index form on the page. Enter mdoc.samples and it will find you mdoc.samples(7), a tutorial sampler for writing BSD man pages. Some distros (I'm told) also come with mdoc(7), mdoc.samples(7) and groff_man(7).

The definitive dope for troff, with all macros explained, is the *Troff User's Manual*, available as html, PostScript (ps, 760K) or Portable Document Format (pdf, 240K). by Jospeh F. Ossanna and Brian W. Kernighan, revised November 1992. AT&T Bell Labs have made it publicly available. Don't forget to check out the late great W. Richard Steven's homepage (famous for *Unix Network Programming* as well as the *TCP/IP Illustrated* trilogy), who also has a list of Troff Resources including tbl, eqn, pic and other filters.

6) What preprocessors may I use?

Groff comes with at least three preprocessors, tbl, eqn, and pic (on some systems they are named gtbl, geqn and gpic.) Their purpose is to translate preprocessor macros and their data to regular troff input. Tbl is a table preprocessor, eqn is an equations/maths preprocessor and pic is a picture preprocessor. Please refer to the man pages for more information on what functionality they provide. To put it in a nutshell: don't write man pages requiring *any* preprocessor. Eqn will generally produce terrible output for typewriter–like devices, unfortunately the type of device 99% of all man pages are viewed on (well, at least I do). For example, XAllocColor.3x uses a few formulas with exponentiation. Due to the nature of typewriter–like devices, the exponent will be on the same line as the base. N to the power of two appears as `N2'. Tbl should be avoided because all xman programs I have seen fail on them. Xman 3.1.6 uses the following command to format man pages, e.g. signal(7):

```
gtbl /usr/man/man7/signal.7 | geqn | gtbl | groff –Tascii –man
/tmp/xmana01760 2> /dev/null
```

which screws up for sources using gtbl, because gtbl output is fed again into gtbl. The effect is a man page without your table. I don't know if it's a bug or a feature that gtbl chokes on its own output or if xman could be a little smarter and not use gtbl twice. Furthermore, some systems use grog to determine what options to pass to groff. Unfortunately grog sometimes guesses wrong and recommends groff –t when in fact tbl must not be used. We are basically left with two workarounds for tables:

1. Format the table yourself manually and put it between .nf and .fi lines so that it will be left unformatted. You won't have bold and italics this way but this beats having your table swallowed any day.
2. Use any `tbl` macros you like but distribute the `tbl` output instead of the input. There is however this quirk with `grog` who thinks that any file containing a line starting with `.TS` requires `tbl`. `Tbl` output for some reason unbeknownst to me still contains `.TS` and `.TE`. It seems you can simply remove them and have the result still look okay. YMMV, so please test this with your particular man page.

I have yet to see a man page requiring `pic` preprocessing. But I would not like it. As you can see above, `xman` will not use it and `groff` will certainly do the funky wadakiki on the input.

7) Should I distribute source and/or already formatted documentation?

Let me give the pros (+) and cons (−) of a few selected possibilities:

1. Source only:
   + smaller distribution package.
   − inaccessible on systems without `groff`.
2. Uncompressed formatted only:
   + accessible even on systems without `groff`.
   − the user can't generate a dvi or postscript file.
   − waste of disk space on systems that also handle compressed pages.
3. Compressed formatted only:
   + accessible even on systems without `groff`.
   − the user can't generate a dvi or postscript file.
   − which compression format would you use? .Z? .z? .gz? All of them?
4. Source and uncompressed formatted:
   + accessible even on systems without `groff`.
   − larger distribution package
   − some systems may expect compressed formatted man pages.
   − redundant information on systems equipped with `groff`.

IMHO it is best to distribute source only. The argument that it's inaccessible on systems without `groff` does not matter. The 500+ man pages of the Linux Documentation Project are source only. The man pages of XFree86 are source only. The man pages from the FSF are source only. In fact, I have rarely seen software distributed with formatted man pages. If any sysadmin is really concerned about having man pages accessible then he also has `groff` installed.

8) What are the font conventions?

First of all: don't use direct font operators like `\fB`, `\fP` etc. Use macros which take arguments. This way you avoid a common glitch: forgetting the font change at the end of the word and having the bold or italic extend up to the next font change. Believe me, it happens more often than you think. The `tmac.an` macros provide the following type faces:

.B Bold

.BI Bold alternating with italics

.BR Bold alternating with Roman

.I Italics

.IB Italics alternating with bold

.IR Italics alternating with Roman

.RB Roman alternating with bold

.RI Roman alternating with italics

.SM Small (scaled 9/10 of the regular size)

.SB Small bold (*not* small alternating with bold)

X alternating with Y means that the odd arguments are typeset in X while the even arguments are typeset in Y. For example

```
.BI "Arg 1 is Bold, " "Arg 2 is Italics, " "and Bold, " "and Italics."
```

The double quotes are needed to include white space into an argument; without them, no white space appears between the alternating typefaces. In fact, you'll only need the macros for alternating typefaces in cases where you *want* to avoid white space between typeface changes. So much for what's available. Here's how you should make use of the different typefaces: (portions shamelessly stolen from man(7))

Although there are many arbitrary conventions for man pages in the UNIX world, the existence of several hundred Linux−specific man pages defines our standards: For functions, the arguments are always specified using italics, even in the SYNOPSIS section, where the rest of the function is specified in bold:

```
.BI "myfunction(int " argc ", char **" argv );
```

Filenames are always in italics, except in the SYNOPSIS section, where included files are in bold. So you should use

```
.I /usr/include/stdio.h
```

and

```
.B #include <stdio.h>
```

Special macros, which are usually in upper case, are in bold:

```
.B MAXINT
```

When enumerating a list of error codes, the codes are in bold. This list usually uses the .TP (paragraph with hanging tag) macro as follows:

```
.TP
.B EBADF
.I fd is not a valid file descriptor.
```

```
.TP
.B EINVAL
.I fd is unsuitable for reading
```

Any reference to another man page (or to the subject of the current man page) is in bold. If the manual section number is given, it is given in roman, without any spaces:

```
.BR man (7)
```

Acronyms look best when typeset in small type face. So I recommend

```
.SM UNIX

.SM ASCII

.SM TAB

.SM NFS

.SM LALR(1)
```

9) Polishing your man page

Following are some guidelines that increase reliability, readability and 'formatability' of your documentation.

- Test examples to make sure they work (use cut and paste to give your shell the exact wording from the man page). Copy the output of your command into your man page, don't just type what you *think* your program will print.
- Proof read, ispell, and have someone else read it, especially if you are not a native English speaker. The HOWTO you are reading has passed the latter test (special thanks to Michael Miller for a particular heroic contribution! All the remaining rough edges are entirely my fault). Additional volunteers are always welcome.
- Test your man page: Does `groff` complain when you format your man page? It's nice to have the `groff` command line in a comment. Does the `man`(1) command complain when you call `man yourprog`? Does it produce the expected result? Will `xman`(1x) and `tkman`(1tk) cope with your manual? XFree86 3.1 has xman 3.1.6 – X11R6, it will try to uncompress using
  `gzip -c -d < %s > %s`
  `zcat < %s > %s`
- Will `makewhatis`(8) be able to extract the one–line description from the NAME section?

10) How do I get a plain text man page without all that ^H^_ stuff?

Have a look at `col`(1), because `col` can filter out backspace sequences. Just in case you can't wait that long:

```
funnyprompt$ groff -t -e -mandoc -Tascii manpage.1 | col -bx >
manpage.txt
```

The `-t` and `-e` switches tell `groff` to preprocess using `tbl` and `eqn`. This is overkill for man pages that don't require preprocessing but it does no harm apart from a few CPU cycles wasted. On the other hand, not using `-t` when it is actually required does harm: the table is terribly formatted. You can even find out (well, "guess" is a better word) what command is needed to format a certain `groff` document (not just man pages) by issuing

```
funnyprompt$ grog /usr/man/man7/signal.7
groff -t -man /usr/man/man7/signal.7
```

"Grog" stands for "GROff Guess", and it does what it says--guess. If it were perfect we wouldn't need options any more. I've seen it guess incorrectly on macro packages and on preprocessors. Here is a little perl script I wrote that can delete the page headers and footers, thereby saving you a few pages (and mother nature a tree) when printing long and elaborate man pages. Save it in a file named `strip-headers` & chmod 755.

```
#!/usr/bin/perl -wn
#  make it slurp the whole file at once:
undef $/;
#  delete first header:
s/^\n*.*\n+//;
#  delete last footer:
s/\n+.*\n+$/\n/g;
#  delete page breaks:
s/\n\n+[^ \t].*\n\n+(\S+).*\1\n\n+/\n/g;
#  collapse two or more blank lines into a single one:
s/\n{3,}/\n\n/g;
#  see what's left...
print;
```

You have to use it as the first filter after the `man` command as it relies on the number of newlines being output by `groff`. For example:

```
funnyprompt$ man bash | strip-headers | col -bx > bash.txt
```

11) How do I get a high quality PostScript man page?

```
funnyprompt$ groff -t -e -mandoc -Tps manpage.1 > manpage.ps
```

Print or view that using your favorite PostScript printer/viewer. See question 10) for an explanation of the options.

12) How do I get `apropos' and `whatis' to work?

Suppose you wonder what compilers are installed on your system and how these can be invoked. To answer this (frequently asked) question you say

```
funnyprompt$ apropos compiler
f77 (1) - Fortran 77 compiler
gcc (1) - GNU C and C++ compiler
pc (1) - Pascal compiler
```

`Apropos` and `whatis` are used to quickly report which man page has information on a certain topic. Both programs search a number of files named `whatis' that may be found in each of the manual base directories. As previously stated, the whatis data base files contain a one line entry for any man page in the respective directory tree. In fact, that line is exactly the NAME section (to be precise: joined on one line and with hyphenation removed; note that the section is mentioned within parentheses). The whatis database files are created with the `makewhatis`(8) program. There are several versions around, so please refer to the man page to determine what options are available. In order for `makewhatis` to be able to extract the NAME sections correctly it is important that you, the manual writer, adhere to the NAME section format described under question 3). The differences between `apropos` and `whatis` are simply where in the line they look, and what they are looking for. `Apropos` (which is equivalent to `man -k`) searches the argument string anywhere on the line, whereas `whatis` (equivalent to `man -f`) tries to match a complete command name only on the part before the dash. Consequently, `` `whatis cc' `` will report if there is a `cc` manual and remain quiet for `gcc`.

Corrections and suggestions welcome!

A) Copying conditions

B) Acknowledgements

- Michael Miller for proofreading the whole HOWTO (in February 2001); Gordon Torrie for many helpful grammar remarks (in August 2001). Any remaining grammar or style bogons are entirely my fault.
- S.u.S.E. (.de) (or .com) who are the only distributor to keep sending me a free copy of their latest product, acknowledging my work as a howto author.

If your name is missing here, drop me a note.

C) Changelog

- March 6 2001: HTML source now passes `weblint -pedantic.` Paragraph 6: Added workarounds for `tbl` screw−ups. Added appendices B) and C). Added RCS Id.
- August 9 2001: Howto put under a two clause BSD license.
- August 20 2001: Improved grammar. Use a numbered list for the TOC.

- October 28 2001: Added refs to mdoc(7), mdoc.samples(7) and groff_man(7).