The *shdoc* package – Manual for version v2.1, 2015/05/24

Simon Michael Laube
simon.laube@gmx.at

```
simon@linuxmint ~ $  tex
This is TeX, Version 3.14159265 (TeX Live 2014)
**\relax
*Hello World!
*\bye
```

Abstract

The *shdoc* packages helps documenting commandline actions in a fancy way. It tries to imitate the look and feel of the original shell prompt while offering a wide range of personalization options.

Contents

1 Introduction

Many people who use LaTeX and do a bit of coding in other languages have used LaTeX's *listings* package at least once. However, when me and my colleague wrote the documentation of our diploma project in 2015, we stumbled across a "problem"[1] with *listings*. A lot of command line scripting had to be done for our project to configure a Raspberry Pi B+ and those actions needed to be documented. *listings* seemed to be the wrong package to look for, so we wrote our own. At first there were few small macros that we referred to as *bashdoc* – this was version 1.0. Later the macros got a bit more complex and we simply threw away the *ba* part of *bashdoc*, that's how the *shdoc* package was born. The basic idea somehow stayed the same since the very first draft, but the package is much more applicable now.

I hope that *shdoc* fits your application as it fitted ours. If there are any serious problems, feel free to send me an email[2]. Moreover, I do not guarantee full compatibility with other packages and different operating systems. The package is distributed under the LaTeX Project Public License version 1.3.

2 Acknowledgement

At this point I would like to thank *Timm Severin* for his active feedback concerning problems and ideas about the package. He pointed out a major bug of version v2.0 and submitted the draft source for some of the new features in version v2.1. Thank you very much!

[1] obviously it's not a real problem, but we wanted something more shell-specific
[2] Please do not send spam mails of any kind. Plain text email format would be appreciated

3 User commands

shdoc implements user commands that allow one to typeset the content/commands of a command line session appropriately. Further there are macros to create and modify styles and appearance of the final output, as well as automated functions that let the user execute commands via *shdoc* and typeset their output immediately.

3.1 Package inclusion and options

Just like every LaTeX package, *shdoc* can be included using the standard `\usepackage` command:

```
\usepackage[⟨options⟩]{shdoc}
```

Since version 2.0 *shdoc* supports several package options to make the environment more adjustable for users. The available options are described in detail in the following sections.

3.2 Color and style management

The shbox environment uses plenty of *different* colors and symbols, which can be adjusted through optional package arguments or macros within the document. Thus, you can either specify the color at the package inclusion in the preamble or just before your session in the document.

`\shchange` To modify the appearance of shell sessions within your document one can use three different commands. The essential macro is `\shchange{⟨<name>⟩} {⟨<new value>⟩}`, which lets you specify a new value for every existing package option. Basically this command is all you need, but it is not recommended to use it for style changes. As mentioned there are two more macros, namely `\shchangecolor` and `\shchangesymbol`. The reason you should use these two for style changes is simply because of enhanced readability of your code. Anyone who reads your source will be able to tell what change you made at the first glimpse – that is mainly important when defining presets, see section 3.2.3. However, the commands all work the same as the latter two are exact copies of `\shchange`. Here are some examples of style changes:

```
\usepackage[backgroundcolor=white,indicatorsymbol=$+$]{shdoc}
or
\shchangecolor{backgroundcolor}{white}
\shchangesymbol{indicatorsymbol}{$+$}


\usepackage[usernamecolor=blue!70]{shdoc}
or
\shchange{usernamecolor}{blue!70} % works, but isn't recommended
```

Be aware that specifying style settings as a package option always sets the style for the whole document, unless the settings are changed again via `\shchange`.

3.2.1 Colors

Table 1 shows the package-specific names of all the colors, their default value and where they are used. Every color is specified using a key-value element as an optional package argument. If no argument is given for a specific color the default value is used.

Name	Description	Default color
backgroundcolor	color of the environment background	gray!70
usernamecolor	color of the username	green!80!yellow
machinenamecolor	color of the machine-name	green!80!yellow
indicatorcolor	color of the indicator symbol	RoyalBlue
separatorcolor	color of the user-machine separator symbol	green!80!yellow
pathcolor	color of the current path	RoyalBlue
optioncolor	color of the little option box	white!60!gray
textcolor	color of the rest of the text	

Table 1: Adjustable colors of the shbox environment

3.2.2 Symbols

In total, only two symbols can be changed in the *shdoc* package. Table 2 shows their name, default value and description. Just as the colors, every symbol is changed using a key-value element as an optional package argument. If no arguments are given, the default values are used.

Name	Description	Default symbol
indicatorsymbol	separates user input and displayed path	$
separatorsymbol	separates user- and machine-name	@

Table 2: Adjustable symbols of the shbox environment

Please pay attention, that the @'s *catcode* is changed to letter internally. Thus, you can use @ directly:

```
\shchangesymbol{separatorsymbol}{@}
```

3.2.3 Appearance and presets

As described above, *shdoc* has some settings that can be changed by the user. Some setting values have been tested and stored as a preset that a user can easily load. Further, users are able to create their own presets via built-in commands.

`\shpreset` One can easily load a preset with `\shpreset{`⟨*name*⟩`}` by using the preset's name as an argument to that macro. When a preset has been loaded it is valid for every new *shbox* environment afterwards, until the `\shpreset` macro is invoked again.

`\shpresetdef` Most of the time, users might want to create their own styles, which can be achieved by defining a new preset via `\shpresetdef{`⟨*name*⟩`}{`⟨*def*⟩`}`. The command has two mandatory arguments: The first one is the desired preset-name and the second one is the preset definition. For example one can use:

```
\shpresetdef{useless}{\relax}
```

and is then able to apply the preset by calling:

```
\shpreset{useless}
```

A preset definition basically exists of multiple style changes, as described at the beginning of section 3.2. For further information about preset definitions, please have a look at all the examples in section 4.1.2 and section 5.2. Please also note that every new preset definition is based on the default preset, as `\shpresetdef` invokes `\shpreset{default}` prior to the definition.

Several presets have been defined by the author and are included in the *shdoc* package. One can use them by calling the `\shpreset{`⟨*name*⟩`}` command with the name of the desired preset. The following list shows the appearance and name of the predefined presets – for visuals see section 4:

default/mint The default preset. It implements a gray background theme with light gray option boxes, light green user- and machine-name and a blue path value.

arrows Not recommended for printed documents. Implements a black background theme with gray option boxes, red username, white separator, light green machine-name, blue path and two arrows as an indicator symbol. The rest of the text is white, too.

darkblue Implements a dark blue background theme with white text, light blue option boxes, orange user- and machine-name and a green path value.

airy A light colored style. Implements a light blue background theme with cyan option boxes and user- & machine-name, as well as light orange path value and black text.

blackwhite Best for non-color prints. Implements a light gray background theme with white option boxes, black text, darker gray user- and machine-name and a dark gray path value.

3.3 The environment

Basically a complete terminal session consists out of two environments that contain the necessary information that is used to typeset the session. The outer environment is named `\sh` and basically acts as the float environment that could contain one or more so called `\shbox`es. Every `\shbox` contains in- and output lines of the terminal, with information provided by the user. The following code shows a brief overview of the possible and necessary commands:

`\sh`

`\shbox`

```
\shpreset{mint}
\begin{sh}
  \shuser{simon}
  \shmachine{linuxmint}
  \begin{shbox}
     \shline{}{cd Desktop/}
     \shline{Desktop/}{whoami}
     \shoutput{}{simon}
  \end{shbox}
\end{sh}
```

3.3.1 Metadata

\shuser
\shmachine

At least once before an occurrence of a `\shbox`, the user- and machine-name should be specified. They are set globally and thus are valid for every `\shbox` that follows the macro call. If no user and machine values are specified, the defaults user:`root` and machine:`linux` are used.

3.3.2 Shell input & path handling

\shline

Within the `\shbox` there are only two valid commands[3], namely `\shline`, which is basically the users' input, the executed command or an arbitrary line; and `\shoutput`, which is ment to be the machine output.`\shline` has two mandatory arguments, where the first one is the path value and the second one is the command that's about to be run.

There are several possibilities to handle the path value. First of all, you can leave the first argument empty, which will force `\shline` to print

defaultpath

the *default path*, namely ~. As one may want to have a different default path, it can be changed by either passing a key-value element of the form `defaultpath=<value>` as an optional argument to the package; or using the al-

\shpath

ready mentioned `\shchange{`⟨*defaultpath*⟩`}{`⟨*<value>*⟩`}` command. Second, one could use `\shpath` to automatically get the absolute path of the current `*.tex` file. Please note that this command does not affect the default path. If you would like to set the default path to the current path of your file, use `\shchange{defaultpath}{\shpath}`. At last, if you just want to use another path for one line, just use the first argument of `\shline` to do so. For example:

```
\shline{/home/user/Desktop}{exit}
```

3.3.3 Shell output

\shoutput

`\shoutput` implements two mandatory macros, too. Here, the first argument is ment for the options of a shell command. For example if you typeset the `--help` output of a command, there may be one or more options like `-c`, `-a` and `-T`. These can be typeset within a fancy optionbox by using the first argument of `\shoutput`. The second argument is ment for any text and is printed without any special formatting. However, it is not mandatory and sometimes not even useful to use the optionbox. In this case you can leave the first argument empty, which forces the macro to suppress the complete optionbox including the additional indent it would otherwise cause.

3.3.4 Name variables

For international compatibility the *shdoc* packages implements two commands to make the list and caption name of the float easily accessible.

\shlistname

One can use `\shlistname` to change the title of the sessions list. The default value is "`\shlistname`{List of Terminal Sessions}".

\shfloatname

In the same manner as described above, the float caption that's displayed below each float could be changed. Its default value is "`\shfloatname`{Terminal Session}".

[3] Actually that's not true, but as you will probably know from the TEXbook, authors need to lie from time to time, to make things a bit easier.

3.4 Automatic command execution and typesetting

Starting with version v2.1, *shdoc* includes macros that let the user execute shell commands from within the LaTeX file and typeset their output immediately. As the whole package is intended to be used on a Linux/Unix shell it uses dedicated commands that will not work on other shells[4].

Note that all the following macros in this subsection are valid within the `\sh` float environment, but not within `\shbox`, as each of them produces its own, enclosed shbox. In other words, each automatic inclusion macro produces a standalone shbox. The necessary float environment, caption and label stay the same.

3.4.1 Warning

Caution! In order to make the following features work, you need to allow TeX to use the `\write18` instruction. For TeXLive this can be done by using the `--shell-escape` option on the shell. It is not recommended to use this option permanently, as you could unconciously execute harmful LaTeX code. Further, only execute LaTeX and shell scripts that you trust, as nobody will take responsibility for their effects.

3.4.2 Read saved command outputs from a file

`\shread` The easiest way to include shell output into your LaTeX file automatically is by redirecting the output stream of the shell command to a file, which is then read by *shdoc*. For example:

```
root@linux ~ $ xsensors -help > xsensors-out.save
```

The resulting file will look like this:

```
Usage: xsensors [options]

Options:
--------

-f              Display all temperatures in Fahrenheit.
-h              Display this help text and exit.
-c filename     Specify the libsensors configuration file.
-i filename     Specify the image file to use as a theme.
-t time         Specify the update time in number of seconds.
                Set this to a negative number for no update.
                Don't set this to 0.
-v              Display version number.
```

xsensors-out.save

After saving the file, `\shread` is used to include it into your document. The macro has two mandatory arguments: The first one is the initial command, as *shdoc* has no way to find out the matching input command to your output file, whereas the second one is the path and name of the file you want to read. Thus, to include the example file from above one would write:

```
\shread{xsensors -help}{xsensors-out.save}
```

[4] The manual mode, however, will always work the same.

While reading command output in that manner, the user still has to take care of at least two steps, but it can be very useful if the output of a command is used elsewhere, too. Furthermore, this is still the fastest way to include command output as all the other methods slow down the LaTeX build of your document.

3.4.3 Run commands and typeset their output

`\shrun` Nevertheless, two steps are still one too much. Thus, `\shrun` lets the user run a shell command from within the document and typesets its output automatically. The macro has one mandatory argument, namely the command that should be passed to the shell. Once again make sure to have `\write18` enabled:

```
\shrun{xsensors -help}
```

The `\shrun` macro takes the shell command, passes it to the system shell and captures its output in a file called `shrun.tmp`. This same file is used for every occuring `\shrun`, so its content is highly temporary. However, the file will not be deleted from your working directory, unless you tell *shdoc* to do so – see section 3.4.7. Note that the macro will also automatically redirect the `stderr` stream to the output stream, so errors should be visible within your document – see session 12 for an example.

3.4.4 Autoread and typeset a file (experimental)

The results of `\shread` are normally nice, but the macro does not support the optionbox of `\shoutput`. That's because files are read line by line, where each line can be seen as a string that is directly passed to `\shoutput` as the second argument. Further, it is very difficult to isolate the shell options (`-c`, `-U`,...) from the line of text. That's mainly because every single program formats its output in a slightly different way. Supporting every programs formatting would nearly be impossible, so another approach had to be taken by the author; and that was string manipulation.

`\shautoread` The result of fooling around with various manipulation methods is a macro called `\shautoread`, which is basically `\shread` except for a few additions that are provided by the *stringstrings* package.

Just as `\shread`, the `\shautoread` macro accepts two mandatory arguments that are treated in the same way as before. The real magic happens when TeX is reading the given file line by line. Each line is stored in an internal macro, which is subsequently accessed by *stringstrings*, extracting the first word of the line. To find out if the first word is an option or not, *stringstrings* then searches this word for the '-' character. If the word contains the '-' exactly once[5], the word is treated as *the option* an is thus passed to `\shoutput` as the first argument. The rest of the line gets printed as text i.e. the second argument.

To this day, the autoread feature is experimental. It may mess up with the output of some programs and you maybe not liking it at all. That's fine, because the macro is simply ment to be worth a try, if you are looking for an opportunity to enhance the appearance of your output once more.

Warning: Using the `\shautoread` function has a huge impact on your LaTeX build time, as *stringstrings* somehow has a lot of things to do when manipulating strings.

[5] I know that there are hundrets of options like `-show-all` or options with two leading dashes, like `--shell-escape`, but to differ between all these options would again be nearly impossible.

Expect at least one minute per autoread, depending on the length of the output and your system performance.

3.4.5 Run and autoread a command (experimental)

`\shautorun` Just like `\shrun`, `\shautorun` lets the user execute a shell command and typeset its output. The only difference is that `\shautorun` reads the temporary file via `\shautoread` – described in section 3.4.4

3.4.6 Autopath & autoformat (experimental)

`\shautopath` **Autopath** In section 3.3.2 we learned that `\shpath` can be used to get the current directory we are working in. To automate things even more both, `\shread` and `\shautoread` can use the current path by invoking `\shpath` on their own. As this functionality is turned off by default, you can enable it by typing:

```
\shautopath{true}
```

You can also turn it off again by passing `'false'` to this macro. Passing any other text may not result in a warning or error, but will be visible in the log file.

`\shautopath` **Autoformat** Another automation feature – and probably the most useful one – is the autoformat script that is distributed with the package. As is, the script uses the `'sed'` stream editor to modify the temporary file `shrun.tmp` and stores the result in `shrun-formatted.tmp`. Thus, the script can be used by both, `\shrun` and `\shautorun` to improve the visual quality of your document. This feature is turned off by default. You can enable it by typing:

```
\shautoformat{true}
```

However, before doing so, you should open the script `shreformat.sh` with your favourite text editor and double check that it won't *harm your computer*[6]! You can also modify it according to your needs in order to get even better results.

3.4.7 Clear temporary files

`\shclearfiles` Most of the time you may want to delete the three temporary files `shrun.tmp`, `shpath.tmp` and `shrun-formatted.tmp` at the end of your LaTeX build, as they are not really useful for anything but the *shdoc* package. One can do so by using `\shclearfiles` at an appropriate point within the document, e.g. before the end.

[6] You shouldn't use unknown scripts blindly

4 Examples

4.1 Various examples

4.1.1 Basic Sessions

```
\shpreset{blackwhite}
\begin{sh}
    \shuser{simon}
    \shmachine{linuxmint}
    \begin{shbox}
        \shline{}{cd Desktop/}
        \shline{Desktop/}{xsensors -help}
        \shoutput{}{}
        \shoutput{}{\underline{Options:}}
        \shoutput{}{}
        \shoutput{-f}{Display all temperatures in Fahrenheit.}
        \shoutput{-h}{Display this help text and exit.}
        \shoutput{-c}{+filename Specify the libsensors configuration file.}
        \shoutput{-t}{+time Specify the update time in number of seconds.}
        \shoutput{-v}{Display version number.}
    \end{shbox}
    \caption{The options of \textit{xsensors}}
    \label{sh:xsensor}
\end{sh}
```

```
01 simon@linuxmint ~ $  cd Desktop/
02 simon@linuxmint Desktop/ $ xsensors -help
03
04 Options:
05
06 [-f] Display all temperatures in Fahrenheit.
07 [-h] Display this help text and exit.
08 [-c] +filename Specify the libsensors configuration file.
09 [-i] +filename Specify the image file to use as a theme.
10 [-t] +time Specify the update time in number of seconds.
11 [-v] Display version number.
```

Terminal Session 1: The options of *xsensors*

```
\shpreset{default}
\begin{sh}
    \shuser{simon}
    \shmachine{linuxmint}
      \begin{shbox}
        \shline{}{tex}
            \shoutput{}{This is TeX, Version 3.14159265 (TeX Live 2014)}
            \shoutput{}{**\cmd{\relax}}
            \shoutput{}{*Hello World!}
            \shoutput{}{*\cmd{\bye}}
          \end{shbox}
    \caption{Hello World in \TeX{}}
    \label{sh:TeX}
\end{sh}
```

```
simon@linuxmint ~ $  tex
This is TeX, Version 3.14159265 (TeX Live 2014)
**\relax
*Hello World!
*\bye
```

Terminal Session 2: Hello World in TEX

4.1.2 Preset creation

```
\shpresetdef{odd}{
    \shpreset{blackwhite}
    \shchangecolor{usernamecolor}{orange}
    \shchangecolor{machinenamecolor}{orange}
}
\shpreset{odd}
\begin{sh}
    \shuser{doctorX}
    \shmachine{supercomputer}
    \begin{shbox}
        \shline{/home/doctorX/Documents}{exit}
    \end{shbox}
    \caption{Go away.}
    \label{sh:exit}
\end{sh}
```

```
doctorX@supercomputer /home/doctorX/Documents $ exit
```

Terminal Session 3: Go away.

```
\shpresetdef{mystyle}{
    \shchangecolor{backgroundcolor}{white}
    \shchangecolor{usernamecolor}{red}
    \shchangecolor{machinenamecolor}{red}
    \shchangecolor{separatorcolor}{RoyalBlue}
    \shchangecolor{optioncolor}{orange!50!yellow}
    \shchangesymbol{indicatorsymbol}{!}
    \shchangesymbol{separatorsymbol}{$\geq$}
}

\begin{sh}
    \shuser{joe}
    \shmachine{joesraspian}%
    \begin{shbox}
        \shline{}{history}
        \shoutput{480}{rubber}
        \shoutput{481}{xsensors}
        \shoutput{482}{tracepath www.google.com}
        \shoutput{483}{whoami}
        \shoutput{484}{help}
        \shoutput{485}{exit}
    \end{shbox}
    \caption{History of my session}
    \label{sh:history}
\end{sh}
```

```
01 joe≥joesraspian ~ !   history
02  [480]  rubber
03  [481]  xsensors
04  [482]  tracepath www.google.com
05  [483]  whoami
06  [484]  help
07  [485]  exit
```

Terminal Session 4: History of my session

4.2 All the predefined presets

```
simon@linuxmint ~ $  cd Desktop/
simon@linuxmint Desktop/ $ xsensors -help

Options:

 [-f] Display all temperatures in Fahrenheit.
 [-h] Display this help text and exit.
 [-c] +filename Specify the libsensors configuration file.
 [-i] +filename Specify the image file to use as a theme.
 [-t] +time Specify the update time in number of seconds.
 [-v] Display version number.
```

Terminal Session 5: The options of *xsensors* – *default* or *mint* preset

```
simon@linuxmint ~ ≫  cd Desktop/
simon@linuxmint Desktop/ ≫ xsensors -help

Options:

 [-f] Display all temperatures in Fahrenheit.
 [-h] Display this help text and exit.
 [-c] +filename Specify the libsensors configuration file.
 [-i] +filename Specify the image file to use as a theme.
 [-t] +time Specify the update time in number of seconds.
 [-v] Display version number.
```

Terminal Session 6: The options of *xsensors* – *arrows* preset

```
simon@linuxmint ~ $  cd Desktop/
simon@linuxmint Desktop/ $ xsensors -help

Options:

 [-f] Display all temperatures in Fahrenheit.
 [-h] Display this help text and exit.
 [-c] +filename Specify the libsensors configuration file.
 [-i] +filename Specify the image file to use as a theme.
 [-t] +time Specify the update time in number of seconds.
 [-v] Display version number.
```

Terminal Session 7: The options of *xsensors* – *darkblue* preset

```
simon@linuxmint ~ $  cd Desktop/
simon@linuxmint Desktop/ $ xsensors -help

Options:

 [-f] Display all temperatures in Fahrenheit.
 [-h] Display this help text and exit.
 [-c] +filename Specify the libsensors configuration file.
 [-i] +filename Specify the image file to use as a theme.
 [-t] +time Specify the update time in number of seconds.
 [-v] Display version number.
```

Terminal Session 8: The options of *xsensors* – *airy* preset

```
simon@linuxmint ~ $  cd Desktop/
simon@linuxmint Desktop/ $ xsensors -help

Options:

 [-f] Display all temperatures in Fahrenheit.
 [-h] Display this help text and exit.
 [-c] +filename Specify the libsensors configuration file.
 [-i] +filename Specify the image file to use as a theme.
 [-t] +time Specify the update time in number of seconds.
 [-v] Display version number.
```

Terminal Session 9: The options of *xsensors – blackwhite* preset

4.3 Automated command execution and formatting

4.3.1 Basic examples

```
\begin{sh}
    \scriptsize
    \shread{avrdude -c usbasp -p attiny24}{avrdude-log.save}
    \caption{\texttt{avrdude} session, read from a file via \cmd{\shread}}
    \label{sh:avrdude_session}
\end{sh}
```

```
simon@linuxmint ~ $  avrdude -c usbasp -p attiny24

avrdude:  AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude:  Device signature = 0x1e910b

avrdude:  safemode:  Fuses OK (H:FF, E:DF, L:E2)

avrdude done.    Thank you.
```

Terminal Session 10: `avrdude` session, read from a file via `\shread`

As you can see the automatic execution of commands works fine[7], although most of the tab characters in session 11 get printed as dashes or other weird symbols. Thus, I recommend to at least use the autoformat script option permanently. The redirection of the error stream in session 12 also shows the expected result.

[7]Note: In case you want to build this documentation on your own, you will need to run `tex shdoc.ins` to get the package file, as well as the file for the `avrdude` session-read example. Further, you may want to make sure to use a system where the given shell commands actually exist.

```
\begin{sh}
    \shautoformat{false}
    \shrun{ifup --help}
    \caption{The help pages of \texttt{ifup}, standard \cmd{\shrun}}
    \label{sh:ifup}
\end{sh}
```

```
01 simon@linuxmint ~ $  ifup -help
02 Usage:  ifup <options> <ifaces...>
03
04 Options:
05 -h, -help--this help
06 -V, -version--copyright and version information
07 -a, -all--process all interfaces marked "auto"
08 -allow CLASS--ignore non-"allow-CLASS" interfaces
09 -i, -interfaces FILE-use FILE for interface definitions
10 -X, -exclude PATTERN-exclude interfaces from the list of
11 ----interfaces to operate on by a PATTERN
12 -n, -no-act--print out what would happen, but don't do it
13 ----(note that this option doesn't disable mappings)
14 -v, -verbose--print out what would happen before doing it
15 -o OPTION=VALUE--set OPTION to VALUE as though it were in
16 ----/etc/network/interfaces
17 -no-mappings--don't run any mappings
18 -no-scripts--don't run any hook scripts
19 -no-loopback--don't act specially on the loopback device
20 -force---force de/configuration
```

Terminal Session 11: The help pages of `ifup`, standard `\shrun`

```
\begin{sh}
    \shrun{xsensor -help}
    \caption{A typo resulting in an error}
    \label{sh:error}
\end{sh}
```

```
01 simon@linuxmint ~ $  xsensor -help
02 sh:  1:  xsensor:  not found
```

Terminal Session 12: A typo resulting in an error

4.3.2 Autoformat script before/after comparison

```
\begin{sh}
    \shautoformat{false}
    \shautorun{xsensors -help}
    \caption{The options of xsensors before using the autoformat script}
    \label{sh:xsensors_noautof}
\end{sh}
```

```
01 simon@linuxmint ~ $  xsensors -help
02
03 Usage:  xsensors [options]
04
05 Options:
06 ----
07
08  [-f..Display]   all temperatures in Fahrenheit.
09  [-h..Display]   this help text and exit.
10  [-c]   filename.Specify the libsensors configuration file.
11  [-i]   filename.Specify the image file to use as a theme.
12  [-t]   time..Specify the update time in number of seconds.
13 ~~Set this to a negative number for no update.
14 ~~Don't set this to 0.
15  [-v..Display]   version number.
16
```

Terminal Session 13: The options of xsensors before using the autoformat script

As you can see in session 13 and 14 the autoformat script can really improve your output. However, xsensors is still an easy job for *shdoc* and sometimes the `\shautorun` macro is not really the best way to run a command. At first, it's still experimental and due to the *stringstrings* manipulation it can take a massive amount of time to build your document. The second reason you might not always use autorun is, when commands output a large number of lines and the occuring options vary a lot in terms of length and style. In this case autorun is not likely to produce a nice output – as seen in session 16 and 17.

```
\begin{sh}
    \shautoformat{true}
    \shautorun{xsensors -help}
    \caption{The options of xsensors after using the autoformat script}
    \label{sh:xsensors_autof}
\end{sh}
```

```
01 simon@linuxmint ~ $  xsensors -help
02
03 Usage:  xsensors [options]
04
05 Options:
06 ----
07
08  [-f]    Display all temperatures in Fahrenheit.
09  [-h]    Display this help text and exit.
10  [-c]   filename Specify the libsensors configuration file.
11  [-i]   filename Specify the image file to use as a theme.
12  [-t]   time  Specify the update time in number of seconds.
13 Set this to a negative number for no update.
14 Don't set this to 0.
15  [-v]    Display version number.
16
```

Terminal Session 14: The options of xsensors after using the autoformat script

```
\begin{sh}
    \tiny
    \shautoformat{false}
    \shrun{gcc --help}
    \caption{Help pages of gcc before autoformat, standard \cmd{\shrun}}
    \label{sh:gcc_noautof_noautorun}
\end{sh}
```

```
simon@linuxmint ~ $  gcc -help
Usage:  gcc [options] file...
Options:
-pass-exit-codes         Exit with highest error code from a phase
-help                    Display this information
-target-help             Display target specific command line options
-help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...]
Display specific types of command line options
(Use '-v -help' to display command line options of sub-processes)
-version                 Display compiler version information
-dumpspecs               Display all of the built in spec strings
-dumpversion             Display the version of the compiler
-dumpmachine             Display the compiler's target processor
-print-search-dirs       Display the directories in the compiler's search path
-print-libgcc-file-name  Display the name of the compiler's companion library
-print-file-name=<lib>   Display the full path to library <lib>
-print-prog-name=<prog>  Display the full path to compiler component <prog>
-print-multiarch         Display the target's normalized GNU triplet, used as
a component in the library path
-print-multi-directory   Display the root directory for versions of libgcc
-print-multi-lib         Display the mapping between command line options and
multiple library search directories
-print-multi-os-directory Display the relative path to OS libraries
-print-sysroot           Display the target libraries directory
-print-sysroot-headers-suffix Display the sysroot suffix used to find headers
-Wa,<options>            Pass comma-separated <options> on to the assembler
-Wp,<options>            Pass comma-separated <options> on to the preprocessor
-Wl,<options>            Pass comma-separated <options> on to the linker
-Xassembler <arg>        Pass <arg> on to the assembler
-Xpreprocessor <arg>     Pass <arg> on to the preprocessor
-Xlinker <arg>           Pass <arg> on to the linker
-save-temps              Do not delete intermediate files
-save-temps=<arg>        Do not delete intermediate files
-no-canonical-prefixes   Do not canonicalize paths when building relative
prefixes to other gcc components
-pipe                    Use pipes rather than intermediate files
-time                    Time the execution of each subprocess
-specs=<file>            Override built-in specs with the contents of <file>
-std=<standard>          Assume that the input sources are for <standard>
-sysroot=<directory>    Use <directory> as the root directory for headers
and libraries
-B <directory>           Add <directory> to the compiler's search paths
-v                       Display the programs invoked by the compiler
-###                     Like -v but options quoted and commands not executed
-E                       Preprocess only; do not compile, assemble or link
-S                       Compile only; do not assemble or link
-c                       Compile and assemble, but do not link
-o <file>                Place the output into <file>
-pie                     Create a position independent executable
-shared                  Create a shared library
-x <language>            Specify the language of the following input files
Permissible languages include:  c c++ assembler none
'none' means revert to the default behavior of
guessing the language based on the file's extension

Options starting with -g, -f, -m, -O, -W, or -param are automatically
passed on to the various sub-processes invoked by gcc.    In order to pass
other options on to these processes the -W<letter> options must be used.

For bug reporting instructions, please see:
<file:///usr/share/doc/gcc-4.8/README.Bugs>.
```

Terminal Session 15: Help pages of gcc before autoformat, standard `\shrun`

```
\begin{sh}
    \tiny
    \shautoformat{false}
    \shautorun{gcc --help}
    \caption{Help pages of gcc before using the autoformat script}
    \label{sh:gcc_noautof}
\end{sh}
```

```
01 simon@linuxmint ~ $  gcc -help
02 Usage:  gcc [options] file...
03 Options:
04 -pass-exit-codes         Exit with highest error code from a phase
05 -help                    Display this information
06 -target-help             Display target specific command line options
07 -help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...]
08 Display specific types of command line options
09 (Use '-v -help' to display command line options of sub-processes)
10 -version                 Display compiler version information
11 [-dumpspecs] cs               Display all of the built in spec strings
12 [-dumpversion] on             Display the version of the compiler
13 [-dumpmachine] ne             Display the compiler's target processor
14 -print-search-dirs       Display the directories in the compiler's search path
15 -print-libgcc-file-name  Display the name of the compiler's companion library
16 -print-file-name=<lib>   Display the full path to library <lib>
17 -print-prog-name=<prog>  Display the full path to compiler component <prog>
18 -print-multiarch         Display the target's normalized GNU triplet, used as
19 a component in the library path
20 -print-multi-directory   Display the root directory for versions of libgcc
21 -print-multi-lib         Display the mapping between command line options and
22 multiple library search directories
23 -print-multi-os-directory Display the relative path to OS libraries
24 -print-sysroot           Display the target libraries directory
25 -print-sysroot-headers-suffix Display the sysroot suffix used to find headers
26 [-Wa,<options>] s>             Pass comma-separated <options> on to the assembler
27 [-Wp,<options>] s>             Pass comma-separated <options> on to the preprocessor
28 [-Wl,<options>] s>             Pass comma-separated <options> on to the linker
29 [-Xassembler] er <arg>         Pass <arg> on to the assembler
30 [-Xpreprocessor] or <arg>     Pass <arg> on to the preprocessor
31 [-Xlinker] er <arg>           Pass <arg> on to the linker
32 -save-temps              Do not delete intermediate files
33 -save-temps=<arg>        Do not delete intermediate files
34 -no-canonical-prefixes   Do not canonicalize paths when building relative
35 prefixes to other gcc components
36 [-pipe] pe                     Use pipes rather than intermediate files
37 [-time] me                     Time the execution of each subprocess
38 [-specs=<file>] e>             Override built-in specs with the contents of <file>
39 [-std=<standard>] d>           Assume that the input sources are for <standard>
40 -sysroot=<directory>    Use <directory> as the root directory for headers
41 and libraries
42 [-B] -B <directory>           Add <directory> to the compiler's search paths
43 [-v] -v                       Display the programs invoked by the compiler
44 [-...] ..                     Like -v but options quoted and commands not executed
45 [-E] -E                       Preprocess only; do not compile, assemble or link
46 [-S] -S                       Compile only; do not assemble or link
47 [-c] -c                       Compile and assemble, but do not link
48 [-o] -o <file>                Place the output into <file>
49 [-pie] ie                     Create a position independent executable
50 [-shared] ed                  Create a shared library
51 [-x] -x <language>            Specify the language of the following input files
52 Permissible languages include:  c c++ assembler none
53 'none' means revert to the default behavior of
54 guessing the language based on the file's extension
55
56 Options starting with -g, -f, -m, -O, -W, or -param are automatically
57 passed on to the various sub-processes invoked by gcc.    In order to pass
58 other options on to these processes the -W<letter> options must be used.
59
60 For bug reporting instructions, please see:
61 [<file:///usr/share/doc/gcc-4.8/README.Bugs>.]
```

Terminal Session 16: Help pages of gcc before using the autoformat script

```
\begin{sh}
    \tiny
    \shautoformat{true}
    \shautorun{gcc --help}
    \caption{Help pages of gcc after using the autoformat script}
    \label{sh:gcc_autof}
\end{sh}
```

```
01 simon@linuxmint ~ $  gcc -help
02 Usage:  gcc [options] file...
03 Options:
04 -pass-exit-codes         Exit with highest error code from a phase
05 -help                    Display this information
06 -target-help             Display target specific command line options
07 -help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...]
08 Display specific types of command line options
09 (Use '-v -help' to display command line options of sub-processes)
10 -version                 Display compiler version information
11 [-dumpspecs]                Display all of the built in spec strings
12 [-dumpversion]              Display the version of the compiler
13 [-dumpmachine]              Display the compiler's target processor
14 -print-search-dirs       Display the directories in the compiler's search path
15 -print-libgcc-file-name  Display the name of the compiler's companion library
16 -print-file-name=<lib>   Display the full path to library <lib>
17 -print-prog-name=<prog>  Display the full path to compiler component <prog>
18 -print-multiarch         Display the target's normalized GNU triplet, used as
19 a component in the library path
20 -print-multi-directory   Display the root directory for versions of libgcc
21 -print-multi-lib         Display the mapping between command line options and
22 multiple library search directories
23 -print-multi-os-directory Display the relative path to OS libraries
24 -print-sysroot           Display the target libraries directory
25 -print-sysroot-headers-suffix Display the sysroot suffix used to find headers
26 [-Wa,<options>]              Pass comma-separated <options> on to the assembler
27 [-Wp,<options>]              Pass comma-separated <options> on to the preprocessor
28 [-Wl,<options>]              Pass comma-separated <options> on to the linker
29 [-Xassembler] <arg>         Pass <arg> on to the assembler
30 [-Xpreprocessor] <arg>      Pass <arg> on to the preprocessor
31 [-Xlinker] <arg>            Pass <arg> on to the linker
32 -save-temps              Do not delete intermediate files
33 -save-temps=<arg>        Do not delete intermediate files
34 -no-canonical-prefixes   Do not canonicalize paths when building relative
35 prefixes to other gcc components
36 [-pipe]                      Use pipes rather than intermediate files
37 [-time]                      Time the execution of each subprocess
38 [-specs=<file>]              Override built-in specs with the contents of <file>
39 [-std=<standard>]            Assume that the input sources are for <standard>
40 -sysroot=<directory>    Use <directory> as the root directory for headers
41 and libraries
42 [-B] <directory>            Add <directory> to the compiler's search paths
43 [-v]                        Display the programs invoked by the compiler
44 [-...]                      Like -v but options quoted and commands not executed
45 [-E]                        Preprocess only; do not compile, assemble or link
46 [-S]                        Compile only; do not assemble or link
47 [-c]                        Compile and assemble, but do not link
48 [-o] <file>                 Place the output into <file>
49 [-pie]                      Create a position independent executable
50 [-shared]                   Create a shared library
51 [-x] <language>             Specify the language of the following input files
52 Permissible languages include:  c c++ assembler none
53 'none' means revert to the default behavior of
54 guessing the language based on the file's extension
55
56 Options starting with -g, -f, -m, -O, -W, or -param are automatically
57 passed on to the various sub-processes invoked by gcc.    In order to pass
58 other options on to these processes the -W<letter> options must be used.
59
60 For bug reporting instructions, please see:
61 [<file:///usr/share/doc/gcc-4.8/README.Bugs>.]
```

Terminal Session 17: Help pages of gcc after using the autoformat script

List of Terminal Sessions

`\listofsh` As the float environment supports automatic lists, you can use them with *shdoc*, too. Just use `\listofsh` for that.

5 Implementation

This section describes the implementation of the *shdoc* package and its features. The package was written as a `*.dtx` source file and therefore the package code begins with the following instruction:

```
1 ⟨*package⟩
```

5.1 Options

First of all, the package options for colouring and symbol options and their default values are defined. The options are then processed to make them available for the user.

color, symbol

```
2 \DeclareStringOption[gray!70]{backgroundcolor}
3 \DeclareStringOption[green!80!yellow]{usernamecolor}
4 \DeclareStringOption[green!80!yellow]{machinenamecolor}
5 \DeclareStringOption[RoyalBlue]{pathcolor}
6 \DeclareStringOption[RoyalBlue]{indicatorcolor}
7 \DeclareStringOption[green!80!yellow]{separatorcolor}
8 \DeclareStringOption[white!60!gray]{optioncolor}
9 \DeclareStringOption[black]{textcolor}
10 \DeclareStringOption[\$]{indicatorsymbol}
11 \DeclareStringOption[\~]{defaultpath}
12 \DeclareStringOption[@]{separatorsymbol}
13 \ProcessKeyvalOptions*
```

5.2 Presets

Right after the option definitions the preset commands are defined. Every preset is stored in a macro with the generic name `\sh@preset@NAME`, where `NAME` stands for the preset name. To make the presets better loadable for users `\shpreset` is defined to simply execute the package internal preset macro:

\shpreset

```
14 \def\shpreset#1{\csname sh@preset@#1\endcsname}%
```

To simplify the creation of new presets a few macros are needed. The main command is `\shpresetdef`, which creates a new preset macro with the desired settings. The macro calls `\shpreset{default}` at the beginning, so the inital settings of the new preset are the default settings of *shdoc*.

\shpresetdef

```
15 \def\shpresetdef#1#2{%
16 \expandafter\gdef\csname sh@preset@#1\endcsname{\shpreset{default} #2}%
17 }%
```

Basically, every command can be used as the second argument of `\shpresetdef`, but most of them won't change any settings in *shdoc*. The only useful macros in this context are those, who explicitly change a color, symbol or value within the package, namely `\shchange`, `\shchangecolor` and `\shchangesymbol`.

`\shchange` has two arguments, where the first is the name of the parameter that should be changed and the second is the new value. Although there won't be

an error, other names than those implemented by the package are not valid. The macro simply redefines the according parameter.

\shchange

```
\def\shchange#1#2{\expandafter\gdef\csname shdoc@#1\endcsname{#2}}
```

\shchangesymbol
\shchangecolor

\shchangesymbol and \shchangecolor are two other commands, that implement the exact same definition as \shchange. Thus, these commands only exists for logical reasons. They simply redefine the desired symbol or color.

```
\let\shchangesymbol\shchange%
\let\shchangecolor\shchange%
```

Since \shpresetdef calls \shpreset{default}, the default preset had to be defined manually via \def.

\sh@preset@default

```
\def\sh@preset@default{%
\shchangecolor{backgroundcolor}{gray!70}%
\shchangecolor{usernamecolor}{green!80!yellow}%
\shchangecolor{machinenamecolor}{green!80!yellow}%
\shchangecolor{pathcolor}{RoyalBlue}%
\shchangecolor{indicatorcolor}{RoyalBlue}%
\shchangecolor{separatorcolor}{green!80!yellow}%
\shchangecolor{optioncolor}{white!60!gray}%
\shchangecolor{textcolor}{black}%
\shchangesymbol{indicatorsymbol}{\$}%
\shchangesymbol{defaultpath}{\~}%
\shchangesymbol{separatorsymbol}{@}%
}%
```

As the *shdoc* package was initally ment to imitate the Linux Mint bash, the default preset is equal to the *mint* style, which is defined right after the default preset:

\sh@preset@mint

```
\shpresetdef{mint}{\shpreset{default}}%
```

The second preset style is the "arrows" style, which is a dark color scheme with two arrows as the indicator symbol.

\sh@preset@arrows

```
\shpresetdef{arrows}{%
\shchangecolor{usernamecolor}{red}%
\shchangecolor{machinenamecolor}{green!80!yellow}%
\shchangecolor{separatorcolor}{white}%
\shchangecolor{indicatorsymbol}{$\gg$}%
\shchangecolor{indicatorcolor}{green!80!yellow}%
\shchangecolor{backgroundcolor}{black}%
\shchangecolor{textcolor}{white}%
\shchangecolor{optioncolor}{gray}%
}%
```

Another dark scheme is the "darkblue" preset, which defines a dark blue background color and more or less appropriate other colors. The definition of this preset is furthermore a nice example, how macros of other packages could also make sense within the \shpresetdef command.

\sh@preset@darkblue

```
\shpresetdef{darkblue}{%
\definecolor{shdarkblue}{RGB}{7,75,138}% xcolor syntax
\shchangecolor{backgroundcolor}{shdarkblue}%
\shchangecolor{textcolor}{white}%
\shchangecolor{separatorcolor}{white}%
\shchangecolor{usernamecolor}{orange}%
\shchangecolor{machinenamecolor}{orange}%
\shchangecolor{pathcolor}{green!60!black}%
\shchangecolor{optioncolor}{SkyBlue!80!black}%
}
```

For those, who print their documents with the black ink cartridge only there is the "blackwhite" preset. The definition sets the background and all the other colors to different versions of gray, black and white – see the examples in section 4.

\sh@preset@blackwhite

```
\shpresetdef{blackwhite}{%
\shchangecolor{backgroundcolor}{gray!30}%
\shchangecolor{textcolor}{black}%
\shchangecolor{separatorcolor}{black}%
\shchangecolor{usernamecolor}{gray}%
\shchangecolor{machinenamecolor}{gray}%
\shchangecolor{pathcolor}{gray!50!black}%
\shchangecolor{optioncolor}{white}%
\shchangecolor{indicatorcolor}{white}%
}
```

The last and probably lightest preset is the "airy" preset with light blue and green elements.

\sh@preset@airy

```
\shpresetdef{airy}{%
\shchangecolor{backgroundcolor}{SkyBlue!15}%
\shchangecolor{usernamecolor}{Emerald}%
\shchangecolor{machinenamecolor}{Emerald}%
\shchangecolor{pathcolor}{orange!70}%
\shchangecolor{indicatorcolor}{orange!70}%
\shchangecolor{separatorcolor}{Emerald}%
\shchangecolor{optioncolor}{Emerald!30}%
}
```

5.3 Environment

After all the setting definitions above the actual environment for the shell commands can be defined. The whole environment consists of a float environment with caption and label and one or more inner frameboxes, which are generated with the *mdframed* package. Further there is a line number on the left side of every box. The number is defined as a standard LaTeX counter and is initially set to 0.

\shlinenumber

```
\newcounter{shlinenumber}% def new counter
\setcounter{shlinenumber}{0}% set counter to 0
```

Since the outer environment is a float, a list name variable is needed to make the name user-adjustable. There are two macros that implement this functionality: `\@shlistname` is the name variable that holds the current value of the list name. `\shlistname` is a user command, that redefines the name according to the users' input. After the definition the standard value is set.

`\shlistname`

```
\let\@shlistname\relax%
\gdef\shlistname#1{\gdef\@shlistname{#1}}%
\shlistname{List of Terminal Sessions}% set default value
```

5.3.1 Float environment

Now, the whole float environment is defined. The float is named `\sh` and uses the plain float style. The float name is stored in a variable, which is implemented in the same way as the list name. Afterwards the name is set. The caption is set to be at the bottom of the float and a macro for the generation of the "List of Terminal Sessions" is defined.

`\sh`

```
\newfloat{sh}{tbph}{lsh}% define new float
\restylefloat*{sh}%
\floatstyle{plain}% set style

\let\@shfloatname\relax%
\gdef\shfloatname#1{\gdef\@shfloatname{#1}}%
\shfloatname{Terminal~Session}% default float name
\floatname{sh}{\@shfloatname}%

\captionsetup[sh]{position=bottom}%
\def\listofsh{\listof{sh}{\@shlistname}}%
```

5.3.2 Framebox

For each session the user and machine values can and must be set, but they could also be set at the beginning of a document to be valid for every terminal session. In the same manner as before, two name variables and their setup commands are defined. The default value for the username is `root`, whereas the default for the machine is `linux`.

`\shuser, \shmachine`

```
\let\@shuser\relax%
\let\@shmachine\relax%
\def\shuser#1{\gdef\@shuser{#1}}%
\def\shmachine#1{\gdef\@shmachine{#1}}%
\def\@default@shuser{root}%
\def\@default@shmachine{linux}%
\shuser{\@default@shuser}%
\shmachine{\@default@shmachine}%
```

`\sh@lnscale`
`\sh@fontcheck`

Within the framebox `\shbox` – which is defined at the very end of the source code – there are two commands: `\shline` for user inputs and `\shoutput` for shell outputs. Each of these macros prints the line number onto the left side of the

box first. The number is relatively scaled down by a factor of 0.7, which is stored in `\sh@lnscale`. However, if the overall font size gets too small (i.e. when using `\scriptsize`, `\tiny`, ...) this factor is increased to prevent warnings from the *relscale* package, which does the line number scaling. The macro that acomplishes this automatic scaling is `\sh@fontcheck`.

```
\def\sh@lnscale{.7}% default number scaling
\def\sh@fontcheck{% if fontsize < 6pt
\ifthenelse{\f@size<6}{%
\gdef\sh@lnscale{1}% factor = 1
}{% elif fontsize < 8pt
\ifthenelse{\f@size<8}{%
\gdef\sh@lnscale{.84}% factor = 0.84
}{% else: nothing
}%
}%
}%
```

`\shline` In order to keep the text from mixing up with the line numbers a hangindent is required. The indent is set according to the width of the linenumbers, so at the beginning of the `\shline` definition `\sh@linenumberwidth` is defined.

Within a `\shline` the font is set to typewriter style, the font size gets checked and the scaling factor for the line numbers is adjusted. Starting without any indent, `\shline` determines if a leading 0 for the line number is needed. If so, the 0 is added, the number gets typeset and the width of the line number is stored. After the number a small horizontal space creates the distance, that's needed between the number and the following username and machine values. They are typeset in their dedicated colors right after the appropriate hangindent is set. A small if-else construct checks if the first argument of the macro is empty and acts accordingly – see section 3.3.2. At the end, the line number is incremented to be ready for the next line.

```
\newlength{\sh@linenumberwidth}%
\long\def\shline#1#2{
\ttfamily\sh@fontcheck\noindent%
\ifnum\value{shlinenumber}<10%
\textcolor{\shdoc@textcolor}{\relscale{\sh@lnscale}0\theshlinenumber}%
\settowidth{\sh@linenumberwidth}{%
\relscale{\sh@lnscale}0\theshlinenumber%
}%
\else%
\textcolor{\shdoc@textcolor}{\relscale{\sh@lnscale}\theshlinenumber}%
\settowidth{\sh@linenumberwidth}{%
\relscale{\sh@lnscale}\theshlinenumber%
}%
\fi\hspace{\sh@linenumberwidth}%
\hangindent=2\sh@linenumberwidth%
\textcolor{\shdoc@usernamecolor}{\@shuser}%
\textcolor{\shdoc@separatorcolor}{\shdoc@separatorsymbol}%
\textcolor{\shdoc@machinenamecolor}{\@shmachine}%
\ \ifx&#1&%
\textcolor{\shdoc@pathcolor}{\shdoc@defaultpath\ }%
\textcolor{\shdoc@indicatorcolor}{\shdoc@indicatorsymbol}
\else%
```

```
\textcolor{\shdoc@pathcolor}{#1\ }%
\textcolor{\shdoc@indicatorcolor}{\shdoc@indicatorsymbol}%
\fi\ \textcolor{\shdoc@textcolor}{#2}%
\stepcounter{shlinenumber}\par
}
```

`\shoutput` The `\shoutput` command is very similar to its pendant, but has other requirements, too. As there may be optionboxes within an output line, the hangindent handling needs to be adjusted. Thus, a new length `\sh@optionboxwidth` is added.

Just as before, the font is set to typewriter style and the size is checked. The line number gets typeset, the horizontal space is added and the hangindent is set. Again a small if-else construct checks the first argument. If the argument is empty, the width of the optionbox is set to zero and the box is suppressed. Otherwise, a framed colorbox is used to create the optionbox and the width of the whole box is stored. The stored value is then used to increase the hangindent appropriately.

```
\newlength{\sh@optionboxwidth}
\long\def\shoutput#1#2{
\ttfamily\sh@fontcheck\noindent%
\ifnum\value{shlinenumber}<10%
\textcolor{\shdoc@textcolor}{\relscale{\sh@lnscale}0\theshlinenumber}%
\settowidth{\sh@linenumberwidth}{%
\relscale{\sh@lnscale}0\theshlinenumber%
}%
\else%
\textcolor{\shdoc@textcolor}{\relscale{\sh@lnscale}\theshlinenumber}%
\settowidth{\sh@linenumberwidth}{%
\relscale{\sh@lnscale}\theshlinenumber%
}%
\fi\hspace{\sh@linenumberwidth}%
\hangindent=2\sh@linenumberwidth%
\ifx&#1&%
\textcolor{\shdoc@textcolor}{#2}\stepcounter{shlinenumber}%
\settowidth{\sh@optionboxwidth}{0cm}%
\else%
\fcolorbox{\shdoc@optioncolor}%
{\shdoc@optioncolor}{\textcolor{\shdoc@textcolor}{[#1]}}\ %
\settowidth{\sh@optionboxwidth}{%
\fcolorbox{\shdoc@optioncolor}%
{\shdoc@optioncolor}{\textcolor{\shdoc@textcolor}{[#1]}}\ }%
\addtolength{\hangindent}{\sh@optionboxwidth}%
\textcolor{\shdoc@textcolor}{#2}\stepcounter{shlinenumber}%
\fi\par%
}
```

`\shautoformat, \shautopath` The autoformat and autopath macros that enable or disable the corresponding functionality are defined as a macro that holds whatever text the user passes as an argument. If the given text is valid is decided later through an if-else statement.

```
\let\sh@autoformat\relax%
\def\shautoformat#1{\gdef\sh@autoformat{#1}}%
\shautoformat{false}%

```

```
\let\sh@autopath\relax%
\def\shautopath#1{\gdef\sh@autopath{#1}}%
\shautopath{false}%
```

\shpath Sometimes it's desired to get the path of the actual working directory. \shpath prints the current path by capturing the output of the Linux-specific *print working directory* 'pwd' command. The output is further stored in the temporary file shpath.tmp.

```
\def\shpath{%
\immediate\write18{pwd >shpath.tmp 2>&1}%
\newread\sh@pathfile%
\openin\sh@pathfile=shpath.tmp%
\endlinechar=-1%
\readline\sh@pathfile to \sh@path%
\sh@path%
\closein\sh@pathfile
}
```

\shread *shdoc* is able to read text files, that contain the output of a shell command, via the \shread macro. At first a new readfile needs to be opened. The endlinechar is set to −1, so it won't appear in the document. After that a new \shbox is started and a loop reads the lines from the text file. A few ifs are used to check if the first argument of \shread is empty and to check if autopath is enabled.

```
\def\shread#1#2{%
\newread\sh@inputfile%
\openin\sh@inputfile=#2%

\endlinechar=-1% hide the endlinechar (-1 is invalid, thus not printed)
\begin{shbox}%
\ifthenelse{\equal{#1}{}}{}% empty: do not print
{%
\ifthenelse{\equal{\sh@autopath}{true}}{%
\shline{\shpath}{#1}%
}{%
\ifthenelse{\equal{\sh@autopath}{false}}{}%
{\typeout{-----------------------------------------------}%
\typeout{ Package shdoc info: I don't know what you}%
\typeout{ mean when passing '\sh@autopath' to autopath}%
\typeout{ and I'm going to ignore it and restore the}%
\typeout{default 'false'} %
\typeout{-----------------------------------------------}%
\shautopath{false}}%
\shline{}{#1}%
}%
}
\loop\unless\ifeof\sh@inputfile%
\readline\sh@inputfile to \sh@fileline%
\ifeof\sh@inputfile% again check of eof prevents last blank line
\else%
\shoutput{}{\sh@fileline}%
\fi%
\repeat%
\end{shbox}%
```

```
\closein\sh@inputfile%
}
```

\shrun `\shrun` lets the user specify a command that is passed to the system shell. Its output is stored in a file – and possibly preformatted by the reformat script – and is then read by `\shread`. An if statement checks if autoformat is enabled.

```
\def\shrun#1{%
\ifthenelse{\equal{\sh@autoformat}{true}}{%
\immediate\write18{#1 >shrun.tmp 2>&1}%
\immediate\write18{./shreformat.sh}%
\shread{#1}{shrun-formatted.tmp}%
}{%
\ifthenelse{\equal{\sh@autoformat}{false}}{}%
{\typeout{-----------------------------------------------}%
\typeout{ Package shdoc info: I don't know what you}%
\typeout{ mean when passing '\sh@autoformat' to autoformat}%
\typeout{ and I'm going to ignore it and restore the}%
\typeout{default 'false'}%
\typeout{-----------------------------------------------}%
\shautoformat{false}}%
\immediate\write18{#1 >shrun.tmp 2>&1}%
\shread{#1}{shrun.tmp}%
}%
}%
```

\shautoread Autoread is an experimental feature of *shdoc* that lets the package detect possible shell options and typeset them within an optionbox. The read functionality is the same as in `\shread`, however *stringstrings* is used to modify the read file lines. At first the first word is extracted from the line via `\getnextword`. After that, the word is searched for the '-' character. If exactly one is found, the word is typeset within an optionbox. The amount of chars that the word consists of is then removed from the complete line and the rest of the line gets typeset as text.

```
\def\shautoread#1#2{%
\newread\sh@inputfile%
\openin\sh@inputfile=#2%
\endlinechar=-1%
\begin{shbox}
\ifthenelse{\equal{#1}{}}{%
}{%
\ifthenelse{\equal{\sh@autopath}{true}}{%
\shline{\shpath}{#1}%
}{%
\ifthenelse{\equal{\sh@autopath}{false}}{}%
{\typeout{-----------------------------------------------}%
\typeout{ Package shdoc info: I don't know what you}%
\typeout{ mean when passing '\sh@autopath' to autopath}%
\typeout{ and I'm going to ignore it and restore the}%
\typeout{default 'false'}%
\typeout{-----------------------------------------------}%
\shautopath{false}}%
\shline{}{#1}%
}%
}%
```

```
\loop\unless\ifeof\sh@inputfile%
\readline\sh@inputfile to \sh@fileline%
\ifeof\sh@inputfile%
\else%
\getnextword[e]{\sh@fileline}%
\findchars[q]{\thestring}{-}%
\ifthenelse{\equal{\theresult}{1}}{%
\shoutput{\thestring}%
{\stringlength[e]{\thestring}\gobblechars[v]{\sh@fileline}{\theresult}}%
}{%
\shoutput{}{\sh@fileline}%
}%
\fi%
\repeat%
\end{shbox}%
\closein\sh@inputfile%
}%
```

`\shautorun` `\shautorun` implements the same functionality as `\shrun`, but invokes autoread to read the file.

```
\def\shautorun#1{
\ifthenelse{\equal{\sh@autoformat}{true}}{%
\immediate\write18{#1 >shrun.tmp 2>&1}%
\immediate\write18{./shreformat.sh}%
\shautoread{#1}{shrun-formatted.tmp}%
}{%
\ifthenelse{\equal{\sh@autoformat}{false}}{}%
{\typeout{-----------------------------------------------}%
\typeout{ Package shdoc info: I don't know what you}%
\typeout{ mean when passing '\sh@autoformat' to autoformat}%
\typeout{ and I'm going to ignore it and restore the}%
\typeout{default 'false'}%
\typeout{-----------------------------------------------}%
\shautoformat{false}}%
\immediate\write18{#1 >shrun.tmp 2>&1}%
\shautoread{#1}{shrun.tmp}%
}%
}
```

`\shclearfiles` `\shclearfiles` removes all the temporary files by calling the shell command '`rm`'.

```
\def\shclearfiles{
\immediate\write18{rm shrun.tmp; rm shrun-formatted.tmp; rm shpath.tmp}
}
```

Finally, the mdframed box is defined through the environment definition command of the *mdframed* package. The frameline color is white and the default backgroundcolor is set. Further, the linecounter is set to 1, when the `\shbox` environment is started.

`\shbox`

```
\newmdenv[linecolor=white,backgroundcolor=\shdoc@backgroundcolor,
settings=\setcounter{shlinenumber}{1}]{shbox}
```

```
\makeatother
```

⟨/package⟩