Dr. Linux MAKE IT SNAPPY WITH THAT SHELL

MARIANNE WACHHOLZ

Complicated organisms like Linux systems suffer little complaints all of their own. Dr. Linux observes the patients in Linux newsgroups, issues prescriptions and proposes alternative healing methods.

(Virtual) console: *One of the text screens between which you can toggle under Linux using the key combination Alt plus function keys (usually [F1] to [F6]). You can get from a graphical user interface with Ctrl+Alt, plus function keys [F1] to [F6], to a text screen.*

Shift key: *The key which makes any letter typed in into a capital.*

Deja vu

During longer sessions at a (text) **console** I would like to read earlier inputs again – especially in cases when a great deal of work has been done. How can I do this?

Dr. Linux: If you cannot manage to scroll through using [**Shift**+scroll up] and [Shift+scroll down] you can make a session log using

```
user$ script Name_of_logfile
```

If you use the *script* command without specifying a file, a log will automatically be produced with the name *typescript* in the current working directory.

Script runs in the background and can be stopped again with Ctrl+D.

Many text consoles provide colour representation which is controlled by *Escape Sequences* (see the box of the same name). This also holds *script* in the log file, so that, in the reproduction, coloured listings are also shown correctly.

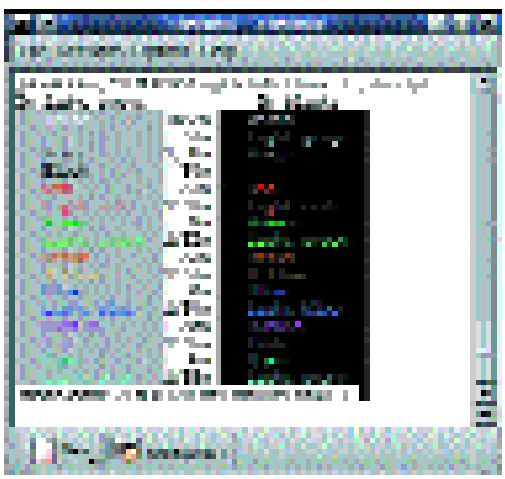

Figure 1: Escape sequences for colours

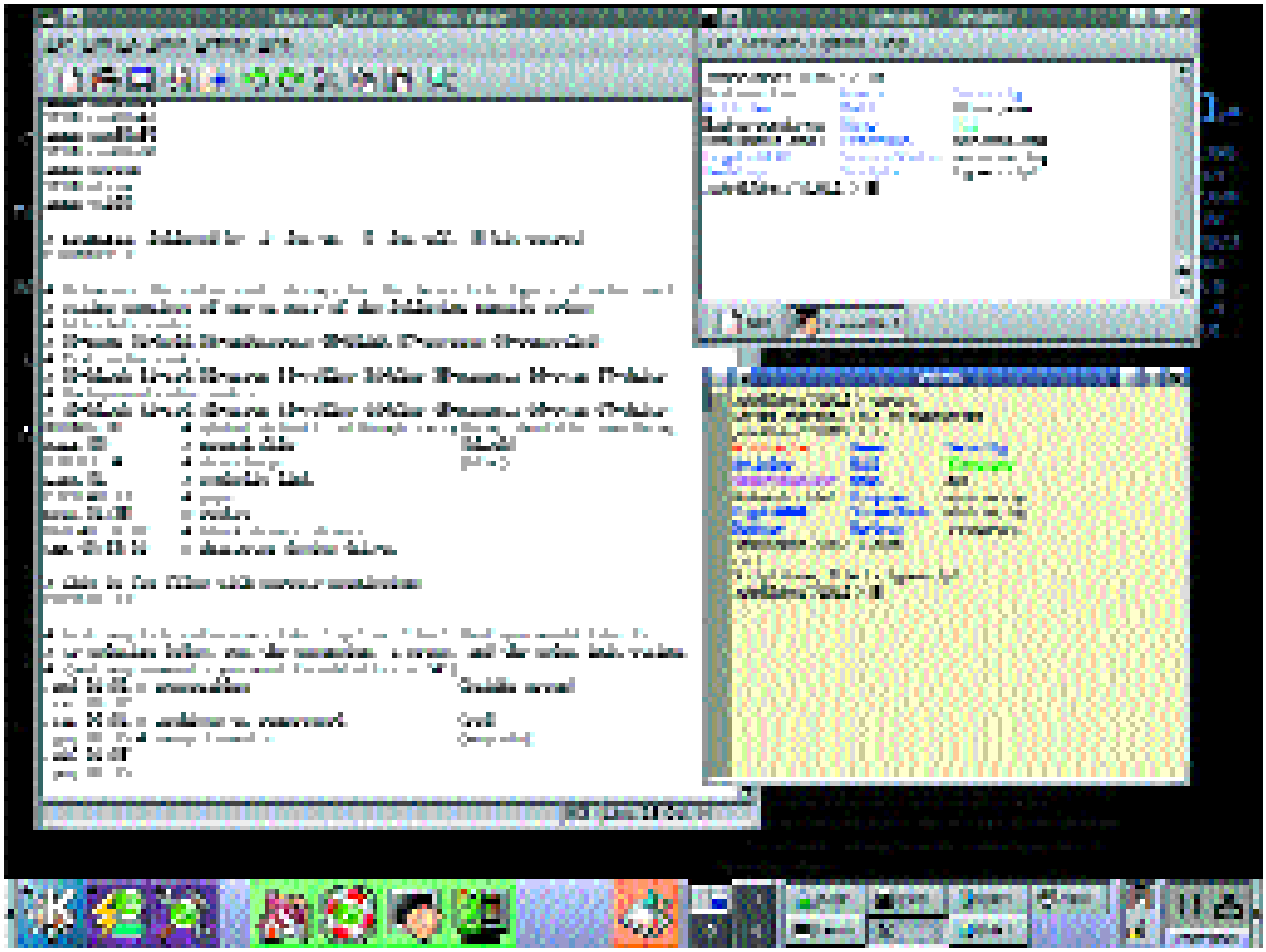

Figure 2: Reproduce a session in text mode with *script*

Begin by logging on to your home directory. To create the log file *script* requires you to have write permission for the current directory. With

```
user$ cat filename_of_log
```

the log can be reproduced from a command line. In the case of longer logs there is the option of a sideways display with *more* instead of *cat*, or you can call up the log in an X-terminal program, which provides a scrollbar.

A personal welcome

Can the log-in message, for example

```
Red Hat Linux release 7.0 (Guinness)
Kernel 2.2.16-22 on an i586
```

be individually configured?

Dr. Linux: The distributions issue messages before the log-in whose content is stored in */etc/issue*. The Superuser can edit this file and thereby change the output before the log-in.

If you open the file with an editor, you may notice that this opening text consists not only of text, but also of control characters. So in SuSE for instance, you will find the following entry:

```
Welcome to SuSE Linux 7.1 (i386) - Kernel \r
(\l).
```

Debian comes across as more Spartan with

```
Debian GNU/\s 2.1 \n \l
```

and Mandrake actually draws a complete penguin with a whole heap of escape sequences:

```
^[[40m^[[40m
^[[2J^[[0;0H
^[[0;1;30;44m   ^[[40m   ^[[44m   ^[[40m
[...]
^[[0m^[[255D

Linux Mandrake release 7.0 (Air)
Kernel 2.2.18 on an i686 / \l
```

With the control characters the case is as follows:

- \V creates in the output the number of the text console (*tty1*, *tty2*, ...)
- \t gives the time in the format HH:MM:SS
- \m gives the processor type (such as i586)
- \r gives the kernel number
- \d inserts the date in the format 2001-06-13
- \n is the control character replaced by the computer name

When you save the entry

```
Welcome - this is kernel \r.
It is \t.
\l is ready to log in.
```

in your */etc/issue*, the log in message looks like this:

```
Welcome - this is kernel 2.2.18.
It is 20:40:22.
tty2 is ready to log in.
```

If you would like to experiment with colours in the Mandrake welcome message, you can fall back on the control characters described in *Escape Sequences*. However, the Escape character wil no longer be written as *\033*, but with Ctrl+V, followed by pressing the [Esc] key.

You will then see a *^[* in the file.

```
^[[47m^[[31m Welcome - this is ^[[40m Kernel
\r.^[[0
It is \t.
\l is ready to log in.
```

ensures that *Welcome - this is* appears as red text on a grey background. The kernel details are output in red on black, and with *Ctrl+V*, *[0* the original colouration is reactivated.

If you would also like to alter the log-in message which is shown to users who log onto your computer via Telnet, then edit the file */etc/issue.net* in the same way.

Infotext after the log in

Escape Sequences

The American National Standards Institute (ANSI) set itself the task of standardising the terminal control characters. This is put into practice with escape sequences, which are mentioned several times in this article and which you will find described in detail in the Bash Prompt HOWTO (http://www.linux.com/howto/Bash-Prompt-HOWTO-6.html).

Some of these control characters will already be familiar to you. The well-known beep, which your system utters from time to time, is also an escape sequence. When placed on a command line, the key combination Ctrl+G makes your computer beep (assuming your system is not configured as mute).

You can use this kind of character for colour control in a way which is much more multimedia – for coloured outputs in text mode or to make the ***Prompt*** *appear in colour for instance. The HOWTO offers a few examples that invite you to try them out as ready-made scripts. Copied into a file and made executable, you can view the colour scheme on your command line. The following script from the HOWTO demonstrates the potential colours (Figure 1):*

```
#!/bin/bash
#
# This file outputs a range of colour codes
# on the terminal.
# Each echo command prints out a foreground colour on
# a grey and black background and writes
# the code for the foreground colour in the middle of the line.
# This script has been tested with white, black and green
# terminal backgrounds (2 December 98).
#
echo "In light grey:          In black:"
echo -e "\033[47m\033[1;37m   white     \033[0m\
1;37m \
\033[40m\033[1;37m   white     \033[0m"
echo -e "\033[47m\033[37m   light grey   \033[0m\
  37m \
\033[40m\033[37m   light grey   \033[0m"
echo -e "\033[47m\033[1;30m   Grey      \033[0m\
1;30m \
\033[40m\033[1;30m   Grey      \033[0m"
echo -e "\033[47m\033[30m   Black    \033[0m\
  30m \
\033[40m\033[30m   Black    \033[0m"
echo -e "\033[47m\033[31m   Red       \033[0m\
  31m \
\033[40m\033[31m   Red       \033[0m"
echo -e "\033[47m\033[1;31m   Light red    \033[0m\
1;31m \
\033[40m\033[1;31m   Light red    \033[0m"
echo -e "\033[47m\033[32m   Green      \033[0m\
  32m \
\033[40m\033[32m   Green      \033[0m"
echo -e "\033[47m\033[1;32m   Light green   \033[0m\
1;32m \
\033[40m\033[1;32m   Light green   \033[0m"
echo -e "\033[47m\033[33m   Brown     \033[0m\
  33m \
\033[40m\033[33m   Brown     \033[0m"
echo -e "\033[47m\033[1;33m   Yellow      \033[0m\
1;33m \
\033[40m\033[1;33m   Yellow      \033[0m"
echo -e "\033[47m\033[34m   Blue      \033[0m\
  34m \
\033[40m\033[34m   Blue      \033[0m"
echo -e "\033[47m\033[1;34m   Light blue    \033[0m\
1;34m \
\033[40m\033[1;34m   Light blue   \033[0m"
echo -e "\033[47m\033[35m   Purple     \033[0m\
  35m \
\033[40m\033[35m   Purple     \033[0m"
echo -e "\033[47m\033[1;35m   Pink      \033[0m\
1;35m \
\033[40m\033[1;35m   Pink      \033[0m"
echo -e "\033[47m\033[36m   Cyan      \033[0m\
  36m \
\033[40m\033[36m   Cyan      \033[0m"
echo -e "\033[47m\033[1;36m   Light cyan   \033[0m\
1;36m \
\033[40m\033[1;36m   Light cyan   \033[0m"

\033 in this case stands for the character created by
pressing the  ESCape key, hence the name escape sequences.
The 33 in the octal numbering system corresponds to the
ASCII code for the escape character (27 decimal).
```

After logging in, I get the following information text on the text console of my system:

```
Last login: Sat Apr 28 14:35:54 on tty6
```

Where do I go to stop this?

Dr. Linux: If you don't want to see this information, you must act as Superuser. The message of the last successful log-ins (all users) is stored in */var/log/lastlog* and output by the log in shell after you log onto the system again. The decision as to whether this data appears on the screen is made in the file */etc/login.defs*.

Open this file with the editor of your choice, and in the line

```
LASTLOG_ENAB     "yes"
```

change the entry "*yes*" to "*no*".

For Mandrake users there is bad news: In some versions of this distribution, setting *LASTLOG_ENAB* does not change anything in the output of the "*Last login*" message.

Getting graphical?

I have specified a graphical log-in in my installation. But I would prefer to log onto a text console and start the graphical user interface with *startx*. Which file do I enter this into?

Dr. Linux: UNIX systems can run in various operating statuses, which are referred to as *run levels*. In the file */etc/inittab* you will find a definition of the preconfigured run level of your system. The numbering of the run level is not the same in all the various distributions.

Common to all is the run level 0, which powers down the system, and run level 6, which triggers a reboot. Therefore it is not advisable to enter one of these two as *default run level*, thus as the run level which is automatically assumed on booting.

The other run levels in the SuSE-7.1 distribution look as follows:

- Run level 1 is the single-user mode, in which *root* has the opportunity to rescue something from a crashed system with only a little functionality
- Run level 2 offers multi-user, but not network functionality
- Run level 3 also allows several users to work on the machine at the same time (multi-user) and also provides network functionality (local mail, logging in from remote machines, etc.)
- Run level 5 is preconfigured as an additional multi-user operating mode with network and graphical log-in

In the file */etc/inittab*, apart from a definition of the run level you will also find the default run level. The watchword is *initdefault*:

```
id:5:initdefault:
```

The number in this line determines which run level your system starts at. In the case of a SuSE 7.1 the number 5 has to be changed to 3, so that the graphical user interface won't be automatically available when you next start the system. In the case of other distributions you can simply read the definitions in */etc/inittab* attentively to find an appropriate multi-user run level.

Program start by return

Why can some programs be started by entering

***Prompt**: The enter prompt (also called the standby character) signals the readiness of a **shell** to execute a command. Since the prompt can be configured, on some systems you will find one that displays the current directory, while other computers, often in networks, also give the respective computer name. When it comes to the purists among administrators, you might even find just individual characters such as %, > or $.*

***Shell**: A command interpreter. This program is started when logging onto the system (log in shell) and accepts the commands entered by the user (interactive shell). Shells usually have elements of programming language (variables, loops, conditional queries etc.) built in, so that you can assemble and have executed instructions for the shell programs or 'shell scripts'. In addition to the bash, which is used by the overwhelming majority of Linux users, under Linux and other UNIX operating systems you can choose from the following shells:*

- *The syntax of the csh or C-shell is similar to the C programming language*
- *The tcsh is an extended C-shell with the option of editing the command line.*
- *The sh or Bourne shell has no editable command line*
- *The ksh or Korn shell which is largely back-compatible with the Bourne shell provides an editable command line*
- *The zsh (Z-shell) is the most recent of the listed shells and can also be edited. Although it is in many respects similar to the ksh, it does have a few special nuances and more extensive features*

***Shell variable:** Many functions of the bash are controlled by variables, which in turn exert an influence on the entire function of a system. The initial settings in the files /etc/profile, /etc/profile.local, /etc/bashrc (depending on the distribution) are read in first. Users have the option of individualising these settings by editing the corresponding dot files in their home directory. The bash processes the following files after a user logs in, in the respective home directory:*

- *.bash_profile*
- *.bash_login*
- *.profile*
- *if applicable, .bashrc*

If one of the files is not available, the next will be searched for possible entries. If you want to permanently set or change the value of a variable, write it in one of these files. The command set will show you which variables are set in the system on the output screen.

*If the directory /sbin is not entered in the PATH variable, this directory will not be searched, such programs must be started with the **absolute or relative pathname**. If a program in the current directory, which is not in the search path, is called up, this can be reached by placing ./ before the program name.*

***Absolute and relative pathname:** An absolute pathname, often also called the full pathname, begins with the root directory, symbolised by /. It usually consists of /Directoryname/any_other_directories/filename. A relative pathname begins in the current directory. /Directoryname is replaced by a dot (.), which produces the command ./any_other_directories/filename. So the system can distinguish (even with files of the same name) as to which file the command applies.*

■

Figure 3: Resetting the graphics card from the alternative mode

their name on a command line and not others? There are various programs such as *fdisk* which I cannot simply start from the command line, even though the program is located under */sbin/fdisk*. Is my system wrongly configured?

Dr. Linux: Assuming you do not wish to start any program files whose execution is reserved for *root* or a specific user and user group (such as *fetchnews*), all programs can be started by entering the full path name as command. Enter the following:

```
user$ /sbin/fdisk
```

Your shell then searches, with the aid of the **shell variable** *PATH* for the program.

This procedure, which appears fiddly, actually makes complete sense. What lies behind it is the basic idea of giving users more security against access and/or damage.

A closer look clarifies this security mechanism. The variable contains a series of directory paths separated from each other by colons. This series is worked through from the top down to the directory path, in which the desired command and/or program is found, while the rest is ignored.

An infiltrated program, which bears only the name of an often-used command, but in the background triggers something completely different, is executed instead of the real program, if it is found by the shell before the actual command.

This is why it is safer only to search through the proposed directory paths. If the current working directory or even the temporary directory jointly used by many were listed upfront in *PATH*, evil-minded programs would only have to wait for a command from the daily flood of email onto standalone workstations or attacker programs in the */tmp* directory – especially in networks.

A careful administrator starts programs with the absolute pathname, so as not to give an infiltrated program the rights of the Superuser.

Using the command

```
user$ echo $PATH
```

you can take a look at which directories belong to your program search path:

```
usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/
/usr/lib/java/:.
```

To briefly include a directory in the *PATH* variable set the following command on the command line:

```
user$ export
PATH=$PATH:/directory/subdirectory
```

The effect of placing *$PATH* in front is that the existing content of the path variables is retained. This change of path has no effect on other terminal windows opened.

If you would still like to include another directory in the variable permanently, you can achieve this by making a corresponding entry in the *~/.bashrc* or *~/.profile*.

Clear sight, clear instructions

Sometimes, in text mode or in an XTerm I open a program with *cat*. This fills the screen with meaningless characters, and all characters entered are shown scrambled. How can I repair it?

Dr. Linux: A program file listed in the heat of the moment almost always results in illegible character output, since the output of program files usually includes control characters, which in turn causes the graphics card to switch into another mode.

You can usually stop the program with Ctrl+C and get back a legible prompt. If the entry prompt continues to be illegible, try using Ctrl+V and then Ctrl+O (Figure 3). With Ctrl+V you tell your shell that you are putting down a control character, for which you will receive no acknowledgement from the shell whatsoever. Ctrl+O is the control character and normally appears in legible form as *^O* on the command line. Then press Return (Enter), which restores the legible information *Command not found* and hopefully a restored prompt.

If you really want to go the whole hog, you should send the command

```
user$ reset
```

afterwards, which removes clutter from the screen output.

In an emergency, you won't be able to put your finger on the key combinations to reset the the graphics card mode. A *reset* typed in blind followed by *Return* (perhaps twice in succession) might be all you need to do to restore your screen. ■