env, export & Co.

Controlling Variability

Controlling variables is a fundamental skill no one should be without. Here we will tell you about shell variables. These hidden stores, by which we pass information such as where to find programs to the shell and scripts, can easily be under your complete control.

BY HEIKE JURZIK AND HANS-GEORG EßER

When you type a command at the console, the shell knows where to look for the program. If you launch an X program from a remote server, the shell knows on which desktop it should be displayed. Even your IRC client knows what IRC server you want to visit and with which nick – all of these things depend on shell variables being set correctly.

Bash: The Final Frontier… The shell and many other programs derive their functionality on Linux from so-called shell variables. These are variables just like those used by many programming languages with the exception that they can only be used to store strings. Variables are assigned values using the assignment operator (the = sign). If the variable needs to store spaces or other non-standard characters then the string must be enclosed in ticks or quotes, such as *a = "1 2 3"*.

Once defined, variables can be accessed arbitrarily using shell scripts or directly within the shell by prefixing a dollar sign; thus the *echo $PATH* command with output the value of the PATH shell variable. The PATH variable stores a colon-separated list of directories that the shell will search through if a command is entered without an absolute pathname being specified (such as *vi* instead of */bin/vi*):

```
huhn@transpluto:~$ echo $PATH
/usr/local/bin:/usr/bin:➘
/usr/X11R6/bin:/bin:/usr/games:➘
/opt/gnome2/bin:/opt/gnome/bin:➘
/opt/kde3/bin:/opt/kde2/bin:.:➘
/usr/sbin:/sbin
```

If you want the variable to be valid in any of the programs you launch, and not only in the shell, you will need to prefix the *export* keyword to the variable assignment, so *a = "1 2 3"* becomes *export a = "1 2 3"*.

As there are lots of things you can configure Linux makes liberal use of shell variables – you can type the *env* (for environment) command to display a list:

```
huhn@transpluto:~$ env
PWD=/home/huhn
PS1=\u@\h:\w\$
ENV=/home/huhn/.bashrc
LS_COLORS=no=00:fi=00:...
LANG=en_GB
SHELL=/bin/bash
HOME=/home/huhn
PATH=/usr/local/bin:/usr/bin:➘
/usr/X11R6/bin:/bin:/usr/games:➘
/opt/gnome2/bin:/opt/gnome/bin:➘
/opt/kde3/bin:/opt/kde2/bin:.:➘
/usr/sbin:/sbin
LESSCHARSET=latin1
TERM=xterm
HOST=transpluto
...
```

This output shows you a few of the environment variables; for example *PS1* defines the appearance of the shell prompt. In this case the variable is set to *\u@\h:\w\$*: *\u* refers to the username, followed by an *@* sign. *\h* outputs the hostname, then after the colon, the *\w* outputs the current (working) directory, and finally the prompt is terminated by a dollar sign to indicate a "normal" user, or a number sign (#) in the case of the administrative user *root*.

The *HOME* variable defines your home directory; you can change to it by simply typing *cd*. If you change the value for the *HOME* variable, this also changes the way *cd* and other programs react when they parse the variable. The *LANG* variable ("en_GB" in this case) defines the language that the shell and other applications will use when they talk to you. So, if you notice that your Linux machine has developed a strange accent, you might like to check the *LANG* variable setting.

For ever?

When you assign a value to a variable, the assignment only applies to the current shell, opening another terminal gets you a new shell where the previously set variable no longer applies. To make the variable available to all shells they need to be *export*ed. These *export*ed variables are so useful you will want them to be set automatically, done by putting the *export* command in one of the autostart files, like *.profile* or *.bashrc* found in your */home* directory.

Before you start defining new variables, you should always ensure that they have not been defined already, as other commands may not react as expected if you change these assignments.

If you overwrite the *DISPLAY* variable, for example, you will not be able to launch any X window programs in the current shell, as X window depends on a correct *DISPLAY* variable setting. Normally this variable will contain a value of the "hostname:0" type.

In some cases shell variables can also be used to define convenient features for programs. Thus you can tell the BitchX IRC client to log on to a specific IRC server using a nickname of your choice, by setting three variables:

```
IRCNICK="mynickname"
IRCNAME="identname"
IRCSERVER="irc.freenode.net"
export IRCNICK IRCNAME IRCSERVER
```

Whereas *IRCNICK* and *IRCSERVER* will apply the settings we just discussed, *IRCNAME* will define what is known as an "Ident" for IRC; in many cases, this will default to your username on the current machine.

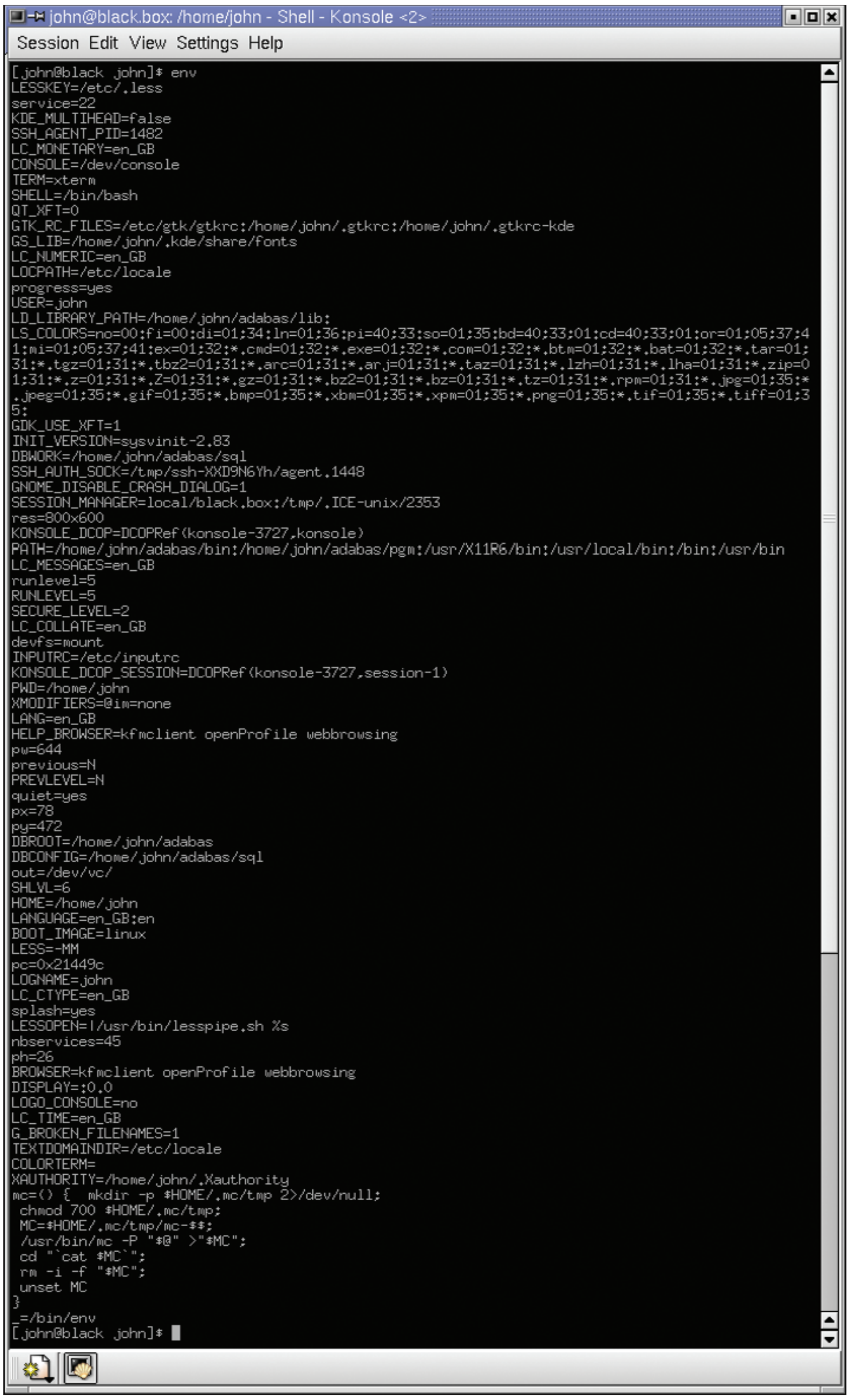
Dropping Assignments

Incidentally, you can call the *env* command with the *-u* flag to drop a variable when launching a specific command. Imagine you want to launch YaST2 in character based mode on SuSE Linux; you could use the following syntax to do so

```
env -u DISPLAY yast2
```

This removes the *DISPLAY* variable from the environment before calling *yast2*. Of course, you could remove the variable using the *unset* command – but this would make the change persistent (for the current shell).

The *env* command we just discussed only applies to the current command and so once you have finished with this shell the DISPLAY variable is reset and your X windows will launch correctly.


john@black.box: /home/john - Shell - Konsole <2>
Session Edit View Settings Help

```
[john@black john]$ env
LESSKEY=/etc/.less
service=22
KDE_MULTIHEAD=false
SSH_AGENT_PID=1482
LC_MONETARY=en_GB
CONSOLE=/dev/console
TERM=xterm
SHELL=/bin/bash
QT_XFT=0
GTK_RC_FILES=/etc/gtk/gtkrc:/home/john/.gtkrc:/home/john/.gtkrc-kde
GS_LIB=/home/john/.kde/share/fonts
LC_NUMERIC=en_GB
LOCPATH=/etc/locale
progress=yes
USER=john
LD_LIBRARY_PATH=/home/john/adabas/lib:
LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;01:or=01;05;37;4
1:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;32:*.com=01;32:*.btm=01;32:*.bat=01;32:*.tar=01;
31:*.tgz=01;31:*.tbz2=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lha=01;31:*.zip=0
1;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.bz=01;31:*.tz=01;31:*.rpm=01;31:*.jpg=01;35:*
.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.xbm=01;35:*.xpm=01;35:*.png=01;35:*.tif=01;35:*.tiff=01;3
5:
GDK_USE_XFT=1
INIT_VERSION=sysvinit-2.83
DBWORK=/home/john/adabas/sql
SSH_AUTH_SOCK=/tmp/ssh-XXD9N6Yh/agent.1448
GNOME_DISABLE_CRASH_DIALOG=1
SESSION_MANAGER=local/black.box:/tmp/.ICE-unix/2353
res=800x600
KONSOLE_DCOP=DCOPRef(konsole-3727,konsole)
PATH=/home/john/adabas/bin:/home/john/adabas/pgm:/usr/X11R6/bin:/usr/local/bin:/bin:/usr/bin
LC_MESSAGES=en_GB
runlevel=5
RUNLEVEL=5
SECURE_LEVEL=2
LC_COLLATE=en_GB
devfs=mount
INPUTRC=/etc/inputrc
KONSOLE_DCOP_SESSION=DCOPRef(konsole-3727,session-1)
PWD=/home/john
XMODIFIERS=@im=none
LANG=en_GB
HELP_BROWSER=kfmclient openProfile webbrowsing
pw=644
previous=N
PREVLEVEL=N
quiet=yes
px=78
py=472
DBROOT=/home/john/adabas
DBCONFIG=/home/john/adabas/sql
out=/dev/vc/
SHLVL=6
HOME=/home/john
LANGUAGE=en_GB:en
BOOT_IMAGE=linux
LESS=-MM
pc=0x21449c
LOGNAME=john
LC_CTYPE=en_GB
splash=yes
LESSOPEN=|/usr/bin/lesspipe.sh %s
nbservices=45
ph=26
BROWSER=kfmclient openProfile webbrowsing
DISPLAY=:0.0
LOGO_CONSOLE=no
LC_TIME=en_GB
G_BROKEN_FILENAMES=1
TEXTDOMAINDIR=/etc/locale
COLORTERM=
XAUTHORITY=/home/john/.Xauthority
mc=() {  mkdir -p $HOME/.mc/tmp 2>/dev/null;
 chmod 700 $HOME/.mc/tmp;
 MC=$HOME/.mc/tmp/mc-$$;
 /usr/bin/mc -P "$@" >"$MC";
 cd "`cat $MC`";
 rm -i -f "$MC";
 unset MC
}
_=/bin/env
[john@black john]$
```

Figure1: Examples of the many different shell variables