Screen Idol

# Multiple Screen Power

Some tools are born great, some tools achieve greatness, and some tools have greatness thrust upon them. This month we shine our spotlight on a great tool named *screen*. BY STEVEN GOODWIN

The purpose of *screen* is to create several virtual consoles from within a single terminal session. Each console is able to run programs, independently of the others, in its own window – even after the user has logged out from the session. Using *screen* you can control several consoles with only one connection, and re-attach to them at a later date. Something that's not possible with *nohup*.

## Premiers Symptomes

*Screen* is started by simply typing its name, at which point the welcome message appears. Hit [return] to remove this and you're apparently back to square one, at a bash prompt. Type *ls*, and it will give you a directory listing as it would on any other day of the week. The only clue that something has happened is that the cursor appears on the first line of the display.

What has actually happened is that *screen* has started running, and spawned a separate process. This will be your default shell, usually bash. However, it is *screen* that is in control. It has not only taken over the session, but the control + a key. This is the combination used to issue commands to *screen*. So to begin, we'll open the help page by pressing control + a together (known henceforth as [C-a], to match the man page), followed by [?] (remembering to release [C-a] first). Remove the help page with [return].

To begin, we'll create a new shell. This is done with [C-a] followed by [c] (without shift), which stands for 'create'. Again, nothing appears to have happened: we still have a bash prompt. However, we have actually spawned a *new* bash prompt. There are now two of them inside *screen*! To prove this, press [C-a "] and you should see the list of virtual windows that *screen* controls. You can pick one using the cursor keys, and select it with [return].

```
Num Name         Flags
  0 bash         $
  1 bash         $
```

This is just one of many ways to change the window (see BOXOUT: Polygon Window). Although you can open as many of these virtual consoles as memory allows, only the first ten have elegant key shortcuts, and only one is visible at any particular time. Despite first appearances, however, all the programs within the consoles continue to run. So we could use one of these windows to edit a program, another to compile it, and yet another to run and test it – working in much the same way as we might with virtual consoles and the [alt + F1] – [alt + F6] keys.

By default, the name shown will reflect the running program, usually bash. Instead, we can give the current window a more descriptive name with the 'title' command, which is invoked using [C-a A] (capital A).

## Double Barrel

In order to copy and paste between windows you need to change into copy mode. From here you can scroll around the visible text (and whatever lines are in *screen*'s history buffer) to select the beginning, and end, of the text to be copied. Press [C-a], followed by the [escape] key to enter copy mode, followed by [H], [J], [K] and [L] to move the cursor left, down, up and right, respectively. The cursor keys will also work, depending on your terminal type. Once you're positioned at the start point, press [space].

Now move the cursor to the end point, and press [space] again to mark the selection (which should be highlighted

### Bad Dreams

Not everything is rosy in the land of *screen* because the default key-press to invoke commands ([C-a]) is the same one used by the shell to jump to the beginning of the current line. And the startup message is a little verbose. Both of these can be solved with a simple amendment to the *$HOME/.screenrc* file.

```
escape "^Ss"
startup_message off
```

with inverse video as you do so). Marking the end point will automatically place the text into the copy buffer. This can then be pasted into another window with [C-a ] ] (close square bracket). When using an editor, remember that it must be in input mode before you can paste text. *screen* sends the text to the window as if it were coming from the keyboard, irrespective of the program running.

The paste buffer exists solely in memory, but can be written to disc with specific instructions available from *screen*s command line mode. This can be accessed with [C-a : ]. The prompt appears on the bottom line of the window, reminiscent of *vi* waiting for instruction.

```
:writebuf ~/my_file_of_buffer➥
_text
```

Be warned! Any filenames given without a path will be placed in the directory in which *screen* was first started, and not the current directory. The same is true of the equivalent *readbuf* command.

In addition, *screen* can exchange large blocks of text between sessions. This works by writing the buffer to a file on disc, and reading it back later. Saving is done with [C-a >], and loading with [C-a <], requiring fewer keystrokes than *writebuf* and *readbuf*. By default, this temporary file is called */tmp/screen-xchg*, but can be changed with the *bufferfile* command.

## Exit Music

One of *screen*'s biggest selling points is that of detachment. This makes it possible to detach all the virtual windows from the user's terminal, but allow the programs to continue running in the background. You can even log out and those processes will continue to run. This allows you to keep an IRC session alive, let a slow download finish, or permit the trilogy of *./configure && make && make install* to complete, while you do something else.

The techniques behind these forking processes are very mysterious, but can be summoned with the magic keys. Namely [C-a C-d] (control + d). This detaches *screen* from the terminal, returning you to a "real" bash prompt. If you now type *screen -list* you will see that there is one detached *screen* session sitting in the background. You can now do anything you wish (including logout), and re-attach to it again later, from anywhere. This includes a different machine.

```
$ screen -r
```

If the current user has several *screen* sessions running, you will see a list consisting of the process id, terminal, and machine name of each one running. In this case you must re-attach to a specific terminal, so its process id must be given:

```
$ screen -list
There are several suitable ➥
screens on:
17169.pts-1.tori     (Detached)
17200.pts-1.tori    (Detached)
Type "screen [-d] -r [pid.]tty➥
.host" to resume one of them.
$ screen -r 17169
```

For users in a hurry, the writers of *screen* have thoughtfully provided the power detach option, which will detach *screen* and logout the current user with the command, [C-a D D].

It is also possible to attach to a process that hasn't been detached! This requires the special -x option, and allows two virtual windows to control the same piece of software from two different places, perhaps an email client, at the same time. Screen updates appear on all connected machines (although some visual problems can occur if the terminals have different dimensions). This can be used for cooperative working or as a monitoring tool, since you do not need to detach the original window. The same rules for detached *screen*s apply to this 'multi-display mode' if there is more than one *screen* available.

```
$ screen -x
```

### Polygon Window

| | |
|---|---|
| C-a C-a | Toggle between two most recent windows |
| C-a " | Produce selectable list |
| C-a 0 | Switch to window 0 |
| C-a 1 | Switch to window 1 (and so on, up to 9) |
| C-a n | Next window |
| C-a p | Previous window |

## Drop the Boy

There is another instance where the detach and re-attach facilitates of *screen* are enjoyed. Those are where you don't mean to log out. And something (or someone) disconnects you from the machine you're trying to use. If all your work is hidden inside a *screen* session, it will still be running safely. Even if your network has become a notwork! Re-attaching to such a session uses exactly the same methods shown above.

Additionally, we can load *screen* with our email client directly, bypassing the need for bash. So we can type:

```
screen mutt
```

Any program started this way can also take a set of arguments. Any options intended for *screen* must precede the program name to avoid confusion. In addition, this will hide the license screen automatically. *Screen* will exit when every program it is taking care of has closed down, at which point the words,

```
[screen is terminating]
```

will appear.

## Light Rain

Screen has many, many, options and we've only played with a few of them. The man page is full of other commands, examples and key-presses we do not have the time to explorer here, like logging, Braille and Kanji characters. Some, like the ability to lock a console screen (with [C-a x]) are very simple, while those concerned with configuring terminal settings, are not. So now it's your turn to spotlight features… ■

### INFO

[1] Screen: *http://ftp.uni-erlangen.de/pub/utilities/screen/*

THE AUTHOR

*When builders go down the pub they talk about football. Presumably therefore, when footballers go down the pub they talk about builders! When Steven Goodwin goes down the pub he doesn't talk about football. Or builders. He talks about computers. Constantly...*