

Speaking UNIX: Stayin' alive with Screen

Create and manage multiple shells on your console

Skill Level: Intermediate

[Martin Streicher](mailto:martin.streicher@gmail.com) (martin.streicher@gmail.com)

Software Developer
Pixel, Byte, and Comma

10 Feb 2009

The command line is a powerful tool, but it has a fatal weakness: If the shell perishes, so does your work. To keep your shell and your work alive—even across multiple sessions and dropped connections—use GNU Screen, a windowing system for your console.

Harry Potter may have his wand, Thor may have Mjölnir, and Buckethead may have his axe, but all those gizmos pale in power to the QWERTY. With a few taps at the command line, you can launch a Web site, recruit legions to your cause, or vanquish a marauding thunder lizard.

Alas, even the mighty QWERTY has a fatal weakness: It succumbs easily to a cut connection. A noisy phone line, a lost cellular or wireless connection, or a network time-out spells certain death for a remote shell. If you've spent hours on a task, such as debugging an application, it can be frustrating and maddening to lose your work in an instant.

But don't smash Mjölnir into your head. Instead, adopt GNU Screen. Screen creates and manages multiple shell windows within a console (say, a dumb terminal physically connected to the host), an xterm window, or a Secure Shell (SSH) login. You can switch from one shell window to another in a flash, and you can leave shells running perpetually and reconnect at any time. Effectively, Screen provides many virtualized consoles.

Figures 1 through 5 picture the features and operation of Screen. Looking at [Figure 1](#), assume that you have used SSH to log in to a remote host. Initially, you have your

original shell on your local host (say, your laptop or desktop) and a remote shell. As usual, you can use the remote shell to run commands on the remote host; output is encrypted and sent over the SSH connection to your local shell. (In the figures, blue highlights the shell output that is currently visible.) As lamented above, if your local shell or the remote shell or the connection between the two is terminated, the remote shell is terminated, taking your work to the big bit bucket in the sky.

Figure 1. A typical SSH connection

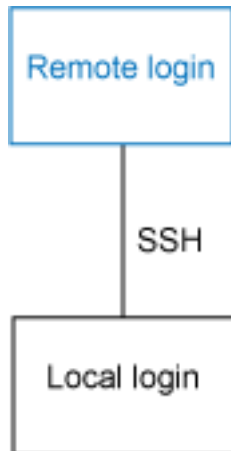
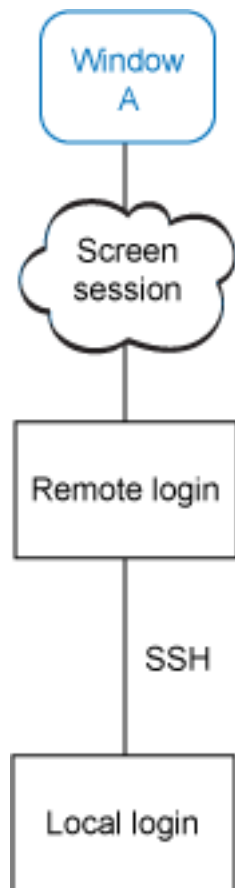


Figure 2 depicts state after you launch Screen on the remote host. The Screen utility launches and in turn spawns a new shell window, *A*, in which you can run commands. The output of *A* is visible, as connoted by blue; the output comes from the shell, traverses through Screen, goes through the remote login, and over the SSH connection, arriving at your local login.

Figure 2. Screen manages shell windows



Screen does not have a presence per se; it's a proxy to select among the available and running windows it manages. Screen can only show the output of one window at a time. Think of Screen as a virtual keyboard-video-mouse (KVM) switch.

But you can also detach from Screen, as shown in [Figure 3](#). The Screen proxy persists, as do all windows it controls, but the connection to Screen is temporarily severed, returning you to the prompt of your remote login shell.

Figure 3. You can detach from Screen and its windows persist

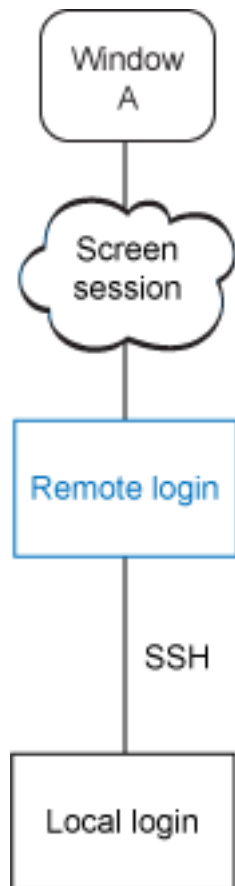
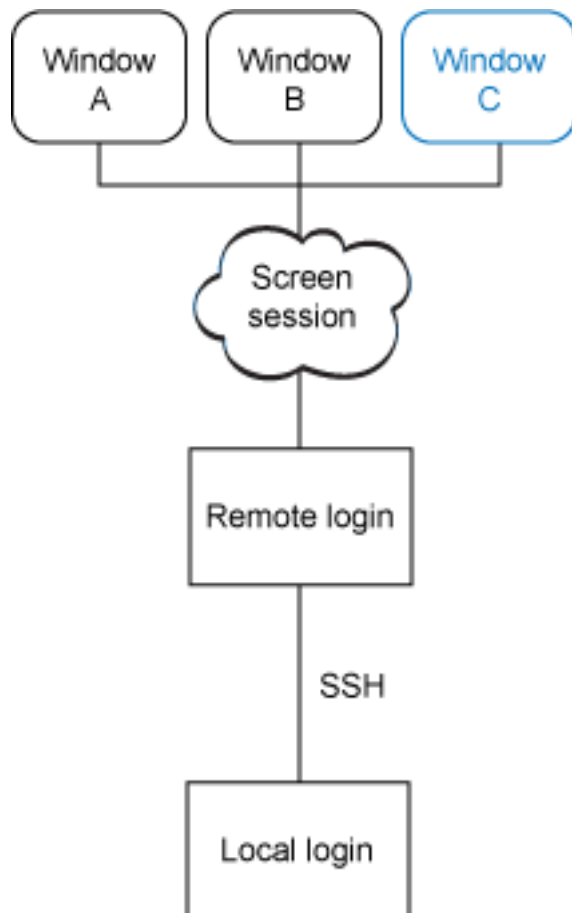


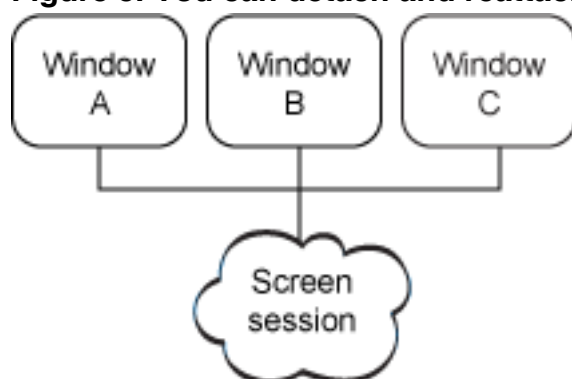
Figure 4 shows a possible, eventual scenario. The connection to Screen has been reestablished, and two additional windows—*B* and *C*—have been created. Windows *A* and *B* continue to run a shell and any subordinate jobs, yet only the output of *C* is visible. Of course, you can switch among the windows to monitor the state of your varied jobs.

Figure 4. Screen can manage multiple windows



Finally, [Figure 5](#) captures conditions if you detach from Screen and terminate your remote login. Screen and its windows persist. You can log in anew, reconnect to Screen (and by proxy to its windows), and carry on unperturbed.

Figure 5. You can detach and reattach to Screen at will



If you regularly access a remote server to perform maintenance or develop software, GNU Screen is indispensable.

Building and installing Screen

The original version of Screen was released more than 10 years ago, so chances are your system has the utility—typically named `/usr/bin/screen`. However, if your system lacks Screen, it is easily installed from your distribution's package manager. For example, if you use a variant of Debian Linux®, such as Ubuntu, you can install Screen in one step with `apt-get`:

```
$ sudo apt-get install screen
```

Alternatively, if you prefer to build from source, visit the GNU Screen project page (see [Resources](#) for a link), and download the latest code bundle. As of this writing, the most recent release of Screen is version 4.0.3, posted just months ago, in August 2008. To build and install from scratch, download and extract the code, change to the resulting source directory, and run the typical `./configure`, `make`, and `sudo make install` sequence:

```
$ wget http://ftp.gnu.org/gnu/screen/screen-4.0.3.tar.gz
$ tar xzf tar xzf screen-4.0.3.tar.gz
$ cd screen-4.0.3
$ ./configure
this is screen version 4.0.3
...
$ make
CPP="gcc -E " srcdir=. sh ./osdef.sh
...
$ sudo make install
...
You may also want to install ./etc/etcscreenrc in
/usr/etc/screenrc.
$ sudo cp ./etc/etcscreenrc /usr/etc/screenrc
```

Screen is now installed and ready to use. Type `man screen` to view the utility's man page.

Getting started with Screen

To use Screen, just launch it. When the license message appears, click **Return**; you should now see a new login shell prompt. (In the transcripts below, a nickname has been artificially added before each shell prompt to differentiate between the numerous shell instances and to draw parallels with Figures 1 through 5.)

```
Local $ ssh remote.example.com
Last login: Sun Dec 21 17:23:16 2008 from
local.example.com
Remote $ hostname
remote.example.com
```

```
Remote $ screen
A $ top
```

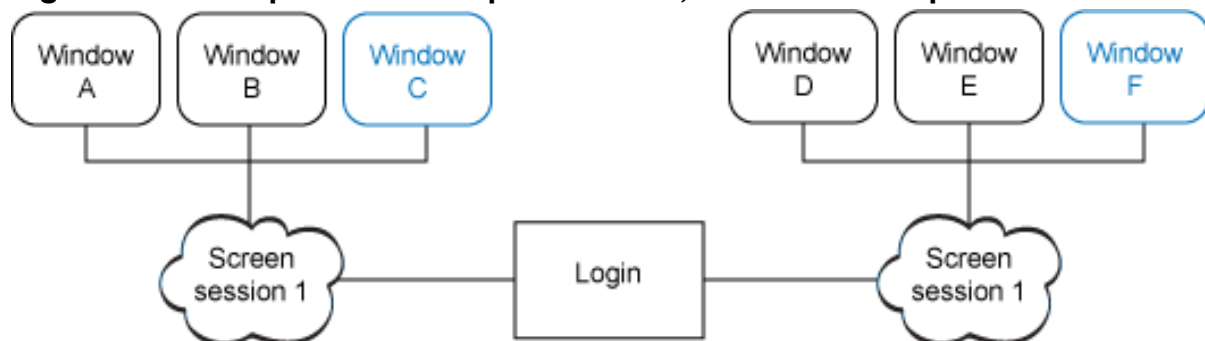
Your state now mirrors that of [Figure 2](#). A Screen session is running (albeit unseen), and window A is active, running the `top` system monitor, which refreshes system performance information every few seconds.

To temporarily detach from the Screen session and all its windows, click **Ctrl-a**, then **d** (lowercase D for "detach"). `Ctrl-a` is the Screen command prefix, and `d` is the specific command to detach. You are now in the scenario shown in [Figure 3](#). The Screen session and window A remain alive. To verify, run `screen -list`:

```
$ screen -list
There is a screen on:
21020.pts-2.remote (Detached)
1 Socket in /tmp/screens/S-strike.
```

The `screen -list` command shows all available screen sessions. (As an aside, you can have multiple screen sessions running simultaneously, each with its own set of concurrent windows. This is shown in [Figure 6](#). Each screen instance tracks its own current window. Some tips on the advanced use of Screen such as this are presented later.) The session numbered `21020` is detached and marked accordingly. Because only one screen session exists, you can reattach to it directly with `screen -r`. Oblivious to Screen's machinations, `top` continues uninterrupted.

Figure 6. Screen provides multiple sessions, each with multiple windows



To recreate [Figure 4](#), reattach to the current session, then click **Ctrl-a**, then **c** (lowercase C for "create") to create a new window. Next, click **Ctrl-a**, then **c** again. You now have one Screen session managing three windows.

To view the available windows in a session in a menu-like interface, click **Ctrl-a**, then the double quotation mark (") key in the current window:

Num	Name	Flags
0	bash	\$
1	bash	\$
2	bash	\$

By default, the name of a window is taken from the first command it launches—commonly, a shell. Hence, there are three *bash* windows in the menu above. To navigate the window menu, use the arrow keys to move up and down the list; to choose a window, simply press Return.

To dispose of the current window, type `exit` at the window's shell prompt, or click the keyboard shortcut **Ctrl-a**, then **k** (lowercase K for "kill"). If you use the latter method, an alert appears at the bottom of the window to confirm that you want to kill the window. Click **y** (lowercase y for "yes") to acknowledge or **n** (lowercase n for "no") to reject the command. If you kill all the windows running under a Screen session, the originating screen command prints a message and is terminated:

```
Remote $ screen
... Create and manipulate windows ...
... Exit from all windows...
[screen is terminating]
Remote $ screen -list
No Sockets found in /tmp/uscreens/S-supergiantrobot.
```

If you're following along and have killed all the open windows, the `screen -list` command yields `No sockets...`, indicating that no screen sessions are available.

Smarter screens

So far, you've seen how to create multiple windows within a Screen session. Alone, this is enough to persist your command-line work and never lose work again.

Admittedly, however, managing seemingly identical, concurrent windows can become confusing; it would be superior if you could easily differentiate between one window and another without opening each one, especially if any amount of time passes between a detach and a reattach.

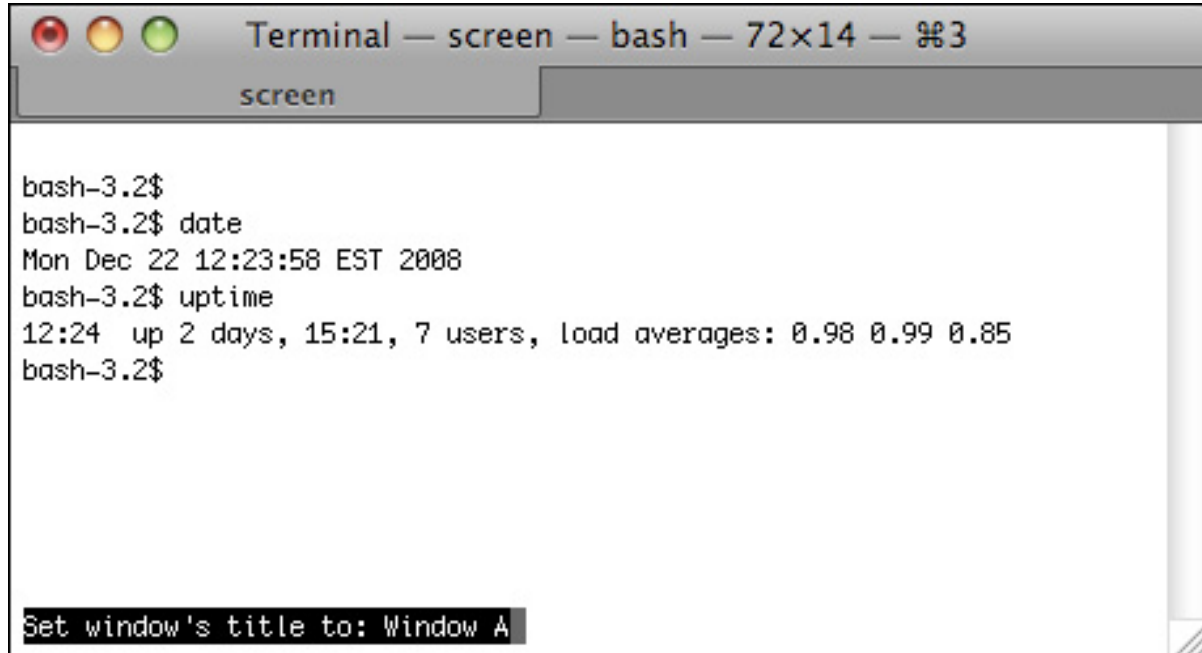
Indeed, Screen provides a number of options and tools to customize your work environment. You can name each window, and you can place a status bar at the bottom of each window to differentiate one from another.

To name a window, activate the window. Click **Ctrl-a A** (uppercase A, for "Annotate"), click **Backspace** as needed to elide any existing moniker, then type a meaningful name at the prompt:

Num	Name	Flags
0	Window A	\$
1	Window B	\$
2	Window C	\$

This is shown in [Figure 7](#), where a window is tagged *Window A*. A window nickname need not be unique.

Figure 7. You can assign a descriptive name to each window

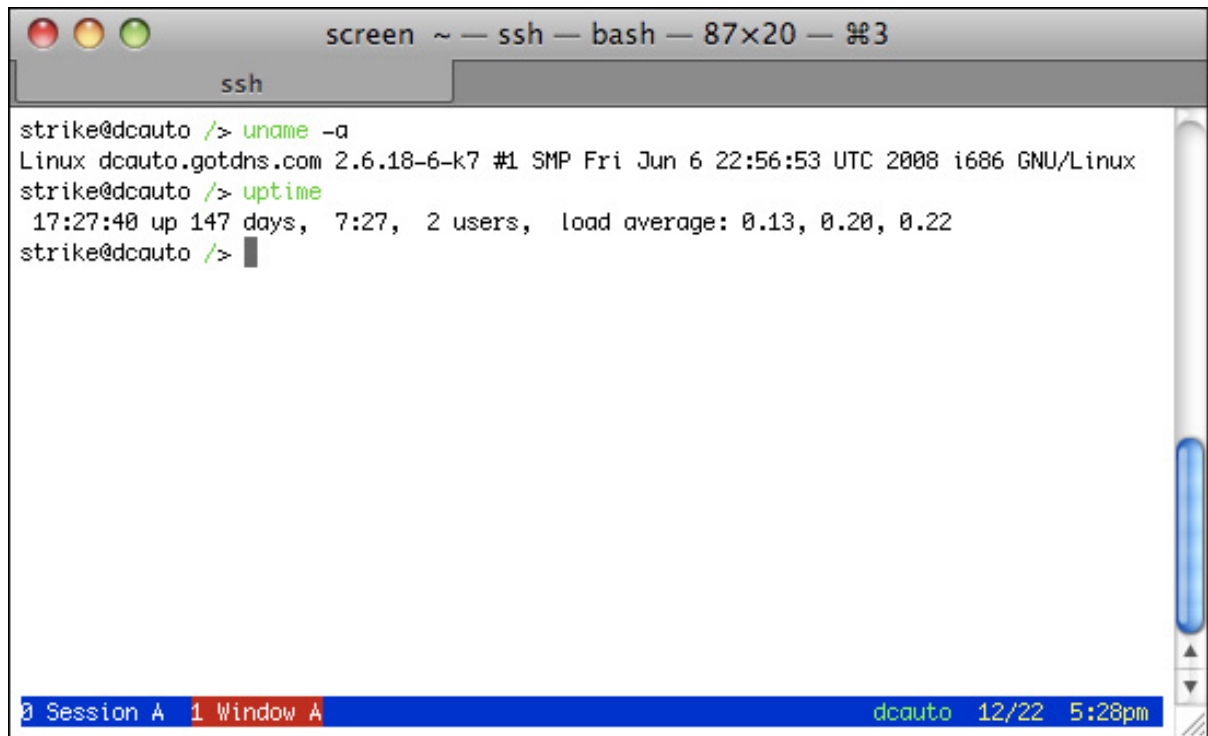


You can provide yourself additional visual cues to distinguish among windows with a status bar. Launch your favorite text editor, and create the file `.screenrc` in your home directory with the following lines:

```
hardstatus on
hardstatus alwayslastline
hardstatus string "%{.bW}%-w%{.rW}%n %t%{-}%+w %=%{..G} %H % {.Y} %m/%d %C%a "
```

With such a `.screenrc` configuration file in place, each new window displays a status line, including the window's name. [Figure 8](#) shows a window with a status line.

Figure 8. Use the status line to help identify each window at a glance



```
screen ~ — ssh — bash — 87x20 — 3
ssh
strike@dcauto /> uname -a
Linux dcauto.gotdns.com 2.6.18-6-k7 #1 SMP Fri Jun 6 22:56:53 UTC 2008 i686 GNU/Linux
strike@dcauto /> uptime
17:27:40 up 147 days, 7:27, 2 users, load average: 0.13, 0.20, 0.22
strike@dcauto />
0 Session A 1 Window A dcauto 12/22 5:28pm
```

Helpful Screen tips

Screen has too many features to list in a brief introduction, but fluent use requires just a modicum of know-how and a few helpful options. Here are some additional pointers:

- **Type `screen` without any arguments in any window to open a new window.** Clicking **Ctrl-a**, then **c** and typing **screen** are synonymous, except that the latter provides command-line options to immediately configure the new session.
- **You can name a window when you create it with `screen -t name`.** For instance, to create a new window and tag it *debugger*, go to a Screen-managed window, then type **screen -t debugger**. If you open the window menu, one of your options should be marked *debugger*.
- **If you've detached from a Screen session, you can reattach to a specific window with `screen -p ID`, where *ID* is a number or a name.** Let's try it:

```
Local $ ssh remote.example.com
Remote $ screen -t ghost
Ghost $ screen -t new
New $
... Press Control-a d to detach...
Remote $ screen -r -p ghost
Ghost $
```

- **You can log the output of every window with `screen -L`.** Each window has its own log file, typically named `~/screenlog.n`, where *n* is the window number shown in the window menu. This is another fantastic feature to record complex steps—say, when a system is being reconfigured.
- **See the Screen documentation for a complete list of accelerator keys.** Some of the most useful combinations are **Ctrl-a**, then **0** (the digit zero) through **Ctrl-a**, then **9** to switch immediately to a specific window; **Ctrl-a**, then **C** (uppercase C for "Clear") clears a window; **Ctrl-a**, then **H** toggles logging on and off; **Ctrl-a**, then **Ctrl-a** flip-flops between the current window and the previous window; and **Ctrl-a**, then **Ctrl-** (the backslash) kills all windows and terminates the current Screen session.

Advanced use of Screen

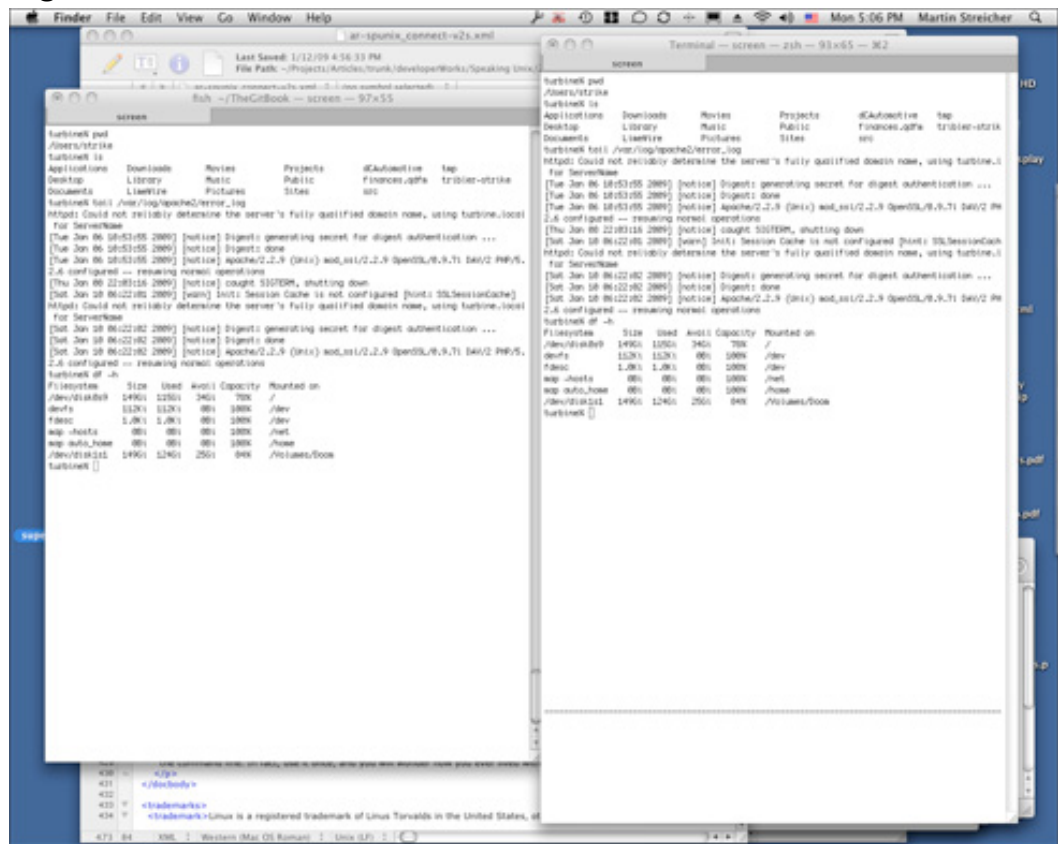
As mentioned earlier, you can create multiple, concurrent Screen sessions, where each session can manage a suite of windows. Each session has a unique identifier; use **screen -list** to catalog all available sessions. As with a window, you can name a session to refer to by its handle. Use **screen -S label** to assign a label to the new session.

Sharing is one of the best uses of a named Screen session. If permitted, you can connect to an existing session and collaborate with other users within any of the session's windows. You can even switch to another window within the session independently! If you're in the same window as another, anything typed or displayed is reflected to all partners. Let's try it:

1. Choose a machine to work on, and log in.
2. Type **screen -S sharing -t one**> to create a new Screen session named, aptly, *sharing*, and a new window named *one*.
3. Create a second window with **screen -t two**.
4. If you like, check your work with **Ctrl-a**, then **d**, **screen -list**, **screen -r sharing**, and **Ctrl-a**, then **"**.
5. Emit some output in the window named *one* by typing **echo** in that window.
6. On the same machine, open a second login window.

7. Within that window, type **screen -x -r sharing -p one**. The **-x** option specifies multi-user mode; **-p one** attaches directly to the window named *one*. You should immediately see the same output as the other login session, as shown in Figure 9.

Figure 9. A session can be shared



Using each login window, run UNIX® commands to produce output, click Screen keyboard shortcuts to switch among the windows in the shared session, and watch the results.

Magic Screen

To delve further into Screen, read about split-screen mode and learn how you can prevent access to individual windows with an old-fashioned lock.

Screen is a remarkable tool that you will quickly find invaluable in any work you perform on the command line. In fact, use it once, and you will wonder how you ever lived without it.

Resources

Learn

- [Speaking UNIX](#): Check out other parts in this series.
- Learn more about [UNIX shells](#).
- The [AIX and UNIX developerWorks zone](#) provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- [New to AIX and UNIX?](#) Visit the New to AIX and UNIX page to learn more.
- Browse the [technology bookstore](#) for books on this and other technical topics.

Get products and technologies

- Download [the source code for Screen](#).

Discuss

- Join the [Screen mailing list](#) to learn tips and tricks.
- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).
- Participate in the AIX and UNIX forums:
 - [AIX Forum](#)
 - [AIX Forum for developers](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant Forum](#)
 - [Performance Tools Forum](#)
 - [Virtualization Forum](#)
 - More [AIX and UNIX Forums](#)

About the author

Martin Streicher

Martin Streicher is a freelance Ruby on Rails developer and the former Editor-in-Chief of [Linux Magazine](#). Martin holds a Masters of Science degree in computer science from Purdue University and has programmed UNIX-like systems since 1986. He collects art and toys. You can reach Martin at

martin.streicher@gmail.com.

Trademarks

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.