Start programs simply by name

QUICK STARTER

The Spotlight utility for the Macintosh has even the most hardened Apple fans scurrying back from the mouse to the keyboard. A short Perl script implements the utility for the Linux desktop. **BY MICHAEL SCHILLI**

If you have ever searched a cluttered desktop for an icon belonging to a particular application, you might have asked yourself who invented the approach of using the mouse to select applications. If you know the application's name, there is really no need to waste valuable time picking an icon from dozens on your desktop.

Spotty, a piece of Perl code, gives you a customizable hotkey that immediately pops up a window at the top right side of your desktop (Figure 1). The keyboard focus automatically shifts to the input box. As soon as you start typing – *fi*, for example – Spotty knows that the program you want to launch must be Firefox, pushes the name to the top of the selection list, and gives you the option of pressing the Tab key to accept the suggestion and launch the application.

All of this takes less than two seconds and works like pressing Alt + F2 in Gnome or KDE; however, coding your own script means that you learn something new and can modify it to your heart's content.

Spotty learns from successful launches and "remembers" the program names it has found by keeping them in a persistent database. The first time you use Spotty to launch an application, you must type *firefox*, for example, and confirm the entry by pressing the Enter key.

Spotty then searches through the directories defined by your *$PATH* environment variable and then launches the program. The next time you use the program, it attempts to compare your input with the program names it has learned and shows matches on the right side of the input box. After Spotty has pushed the program you are looking for to the top of the list, just press the Tab key to tell Spotty to fire up the application.

No Way Back after Exec

To launch the application, Spotty uses *exec*. This overloads the current process (the Perl script) with the external application, thus removing Spotty from the process table and leaving the launched application in its place. Thus, the *exec* in the *launch* function in line 133 is the end of the script because the process does not return from it.

The database in which the program names are stored is a persistent hash that uses the CPAN *DB_File* module employing a Berkeley DB. The script updates the database whenever the hash associated with it by the *tie* command is changed. To close everything gracefully, line 125 issues an *untie* shortly before the *exec* command to untie the hash from the database and store the changes.

Drawing with Tk

As you can see from Listing 1, Spotty uses the CPAN *Tk* module to draw the application window with the input box. To prevent the window appearing just anywhere on the desktop and to keep it firmly in the top right-hand corner, Spotty then calls the *geometry()* method with the *-0 + 0* parameters. *-0* stands for the *x* coordinate on the far right, and *+0* stands for the topmost *y* coordinate.

The main window, *$top*, is of the *MainWindow* type and contains two widgets: an *Entry* type input field on the left and a *Label* type display to its right. Linked to the *$entry* input box widget is a text variable, *$input*, which *Tk* uses to store the text typed by the user and which is refreshed after each keystroke. Because the *-validate* option has a value of "*key*", *Tk* jumps to the *validate()*

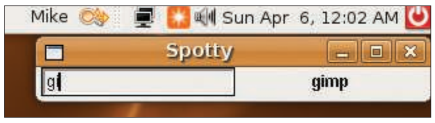
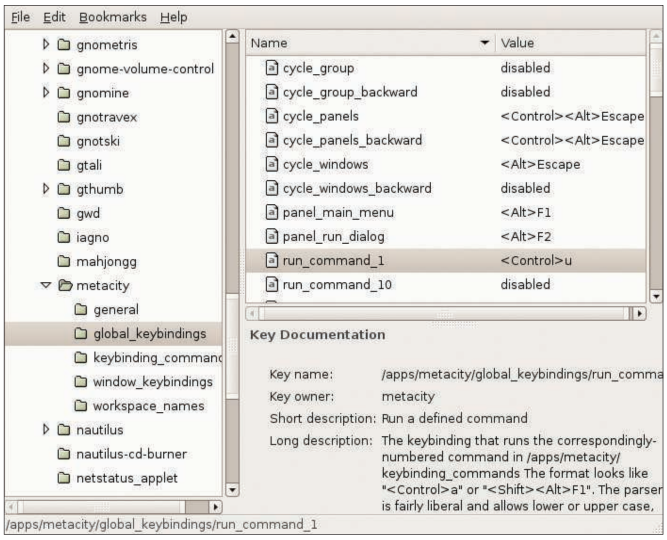


Figure 1: The user here has just launched Spotty by pressing the hotkey Ctrl+U and typing "gi" in the input box.



gnometris
gnome-volume-control
gnomine
gnotravex
gnotski
gtali
gthumb
gwd
iagno
mahjongg
metacity
general
global_keybindings
keybinding_commanc
window_keybindings
workspace_names
nautilus
nautilus-cd-burner
netstatus_applet

Name	Value
cycle_group	disabled
cycle_group_backward	disabled
cycle_panels	<Control><Alt>Escape
cycle_panels_backward	<Control><Alt>Escape
cycle_windows	<Alt>Escape
cycle_windows_backward	disabled
panel_main_menu	<Alt>F1
panel_run_dialog	<Alt>F2
run_command_1	<Control>u
run_command_10	disabled

Key Documentation

Key name: /apps/metacity/global_keybindings/run_comma
Key owner: metacity
Short description: Run a defined command
Long description: The keybinding that runs the correspondingly-numbered command in /apps/metacity/keybinding_commands The format looks like "<Control>a" or "<Shift><Alt>F1". The parser is fairly liberal and allows lower or upper case,

/apps/metacity/global_keybindings/run_command_1

Figure 2: The gconf-editor utility maps run_command_1 to the "<Control>u" hotkey in Apps/Metacity/global_keybindings.

function (lines 75ff.) for each keystroke; the *-validatecommand* option is used to pass a reference to the function in to the widget. Of course, the function doesn't actually validate anything here because it always returns a *1*; it simply serves to execute a callback that searches the database for matches for each character the user types. The Label widget to the right of the Entry widget monitors a text variable, *$label_text*, and the Tk Manager updates the display whenever its value changes.

If the *validate()* function notices that *matches()* (lines 85ff.) finds one or more matches for the word entered by the user, it separates them by line breaks and concatenates them to a single string that it stores in *$label_text*. This prompts Spotty to display the matches to the right of the input window without any additional programming effort.

The packer (lines 52 and 54) uses the *-side => "left"* option to pack both widgets into the container object, the main window *$top*. If *"left"* moves multiple objects into a container, the packer lines them up from left to right. The *"left"* option tells the packer to glue any new widgets to the left border of the available space. If a widget is already sitting in this slot, the next one is dropped on the far left of the free space to its right.

Spotty reacts to the Return and Tab keys. Return confirms the string entered thus far, and Tab accepts the first element in the list of suggestions. Perl-Tk uses *bind* calls in lines 59 and 61 to bind the keys to the *launch()* (lines 110ff.) and *complete()* (lines 102ff.) functions, respectively. The latter simply sets the variable for the Entry widget to the top match, *$first_match*, and then calls *launch()* so the user does not need to press Enter to launch the application. The *bind* entry in line 57 tells Spotty to cancel the action and exit the program if somebody presses Ctrl + Q.

The call to *focus()* in line 64 shifts the keyboard focus to the Entry widget, which is important because the user would otherwise need to drag the mouse and click the entry box to get the widget to accept keyboard input.

After defining all these settings, *MainLoop* in line 65 launches the GUI, which runs until an application launches or the user bails out of the program by pressing Ctrl + Q. In this case, or in case of an

Listing 1: Spotty (continued on p78)

```
001 #!/usr/local/bin/perl -w
002 use strict;
003 use Log::Log4perl qw(:easy);
004 use DB_File;
005 use Tk;
006
007 my %sudo_programs =
008   map { $_ => 1 }
009   qw(synaptic);
010
011 my @misc_paths =
012   qw(/usr/sbin);
013
014 my ($home) = glob "~";
015 my $spotty_dir =
016   "$home/.spotty";
017
018 #Log::Log4perl->easy_init();
019
020 if (!-d $spotty_dir) {
021  mkdir $spotty_dir, 0755
022    or LOGDIE "Cannot mkdir ",
023         "$spotty_dir ($!)";
024 }
025
026 # Init database
027 my %DB_FILE;
028 tie %DB_FILE, "DB_File",
029   "$spotty_dir/db_file.dat"
030   or LOGDIE "$!";
031
032 # Application window
033 my $top = MainWindow->new();
034 $top->geometry("-0+0");
035
036 my ($input, $first_match,
037     $label_text);
038
039 my $label = $top->Label(
040  -textvariable =>
041    \$label_text,
042  -width => 20
043 );
044
045 my $entry = $top->Entry(
046  -textvariable    => \$input,
047  -validatecommand =>
048    \&validate,
049  -validate => "key",
050 );
051
052 $entry->pack(
053  -side => "left");
054 $label->pack(
055  -side => "left");
056
057 $entry->bind(
058  "<Control-Key-q>", \&bail);
059 $entry->bind("<Return>",
060  sub { launch($input) });
061 $entry->bind("<Tab>",
062  \&complete);
063
064 $entry->focus();
065 MainLoop;
066
067 ##################################
068 sub bail {
069 ##################################
070
071  $top->destroy();
072 }
```

error, the *bail* function (line 68) helps to clean up by calling the top window's *destroy* method, thus removing the GUI.

Choosing a keyboard shortcut for your Spotty desktop hotkey that's not in use by any other application program makes sense; remember that pressing it shifts the keyboard focus to the tool. This might not be easy in the case of key hogs, such as Vim. I opted for Ctrl + U because it's easy to press and is not on my list of frequently used Vim commands. (Of course, Vim does use the combination to scroll the edited text up, but I use Ctrl + B instead.)

Unfortunately, the Gnome desktop on my Ubuntu installation thinks it knows better and only lets me bind selected applications to hotkeys, not just any old program. To resolve this issue, I had to run *gconf-editor* (Figure 2). If the tool is missing, you can issue *sudo apt-get install gconf-editor* to install the package.

Below *Apps*, you will see an entry for Metacity (the Gnome Window manager), and below this, the items *global_keybindings* and *keybinding_commands*. Set *run_command_1* below *global_keybindings* to the required hotkey combination – for example, Ctrl + U – and then add the path to Spotty below *keybinding_commands*.

Extensions

Launching a program that needs root privileges causes a minor problem because you would normally have to enter your password. The Ubuntu package manager, *synaptic*, is one example of this. The program will run, but without root privileges, which means that it can query packages but cannot install new ones.

Spotty solves this with the hash defined in line 7: *%sudo_programs*. If the program selects one of the programs listed here, line 130 not only issues an *exec*, but Spotty also launches an Xterm terminal that calls *sudo* to launch the program in question. The effect of this is that the shell in the *xterm* that pops up onscreen first prompts you for a password, and if you enter the right one, it runs the program with root privileges.

If you would like to search for commands in paths outside of your *$PATH* environment variable, you can add entries to the *@misc_paths* array in line 11. *path_search()* automatically finds programs in the extended path.

Also, if you prefer to use the cursor keys instead of typing letters, you can extend the Perl code to draw a list box populated with matches to the right of the input box and select an entry from the list.

Regardless of which approach you prefer, if you know what you are looking for, you will find it much faster with a little help from Spotty. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/resources/article_code*

Listing 1: Spotty (continued from p77)

```
073
074 #################################
075 sub validate {
076 #################################
077  my ($got) = @_;
078
079  $label_text = join "\n",
080                matches($got);
081  return 1;
082 }
083
084 #################################
085 sub matches {
086 #################################
087  my ($got) = @_;
088
089  my @all =
090    sort keys %DB_FILE;
091  my @matches =
092    grep { /^$got/ } @all;
093  if (@matches) {
094   $first_match = $matches[0];
095  } else {
096   $first_match = undef;
097  }
098  return @matches;
099 }
100
101 #################################
102 sub complete {
103 #################################
104
105  $input = $first_match;
106  launch($input);
107 }
108
109 #################################
110 sub launch {
111 #################################
112  my ($program) = @_;
113
114  my $path =
115    path_search($program);
116
117  LOGDIE
118    "$program not found ",
119    "in path ($ENV{PATH})"
120    unless defined $path;
121
122  $DB_FILE{$program}++
123    if defined $path;
124  DEBUG "Launching $path";
125  untie %DB_FILE;
126  if (
127   exists
128   $sudo_programs{$program})
129  {
130   exec "xterm", "-e", "sudo",
131        "$path";
132  } else {
133   exec $path;
134  }
135  LOGDIE
136    "exec $path failed: $!";
137 }
138
139 #################################
140 sub path_search {
141 #################################
142  my ($program) = @_;
143
144  DEBUG "PATH is $ENV{PATH}";
145
146  for my $path (
147   split(/:/, $ENV{PATH}),
148   @misc_paths)
149  {
150   if (-x "$path/$program") {
151    DEBUG "$program found ",
152          "in $path";
153    return "$path/$program";
154   }
155  }
156
157  ERROR "$program not found";
158  return undef;
159 }
```