Perl scripts add useful options to extend cut-and-paste

Grab And Go

With a couple of lines of code, you can create simple Perl scripts that enhance the Linux desktop's cut-and-paste feature, adding buffers for time-saving editing. By Mike Schilli

Cutting text on the desktop with the mouse and pasting it elsewhere is a very useful feature, and not even the most ardent supporters of the command line will deny this fact. If you take a closer look, the Linux desktop window manager actually supports two different buffer mechanisms: primary selection and the clipboard [1].

If you use the mouse to select a line of text, you can drop the content of this primary selection elsewhere by pressing the middle mouse button. In contrast to this, the clipboard expects users to trigger the copy mechanism after selecting something (typically using Ctrl + C or the *Edit* | *Copy* menu in a graphical application) and to call the paste function when they reach the target (Ctrl + V or *Edit* | *Paste*).

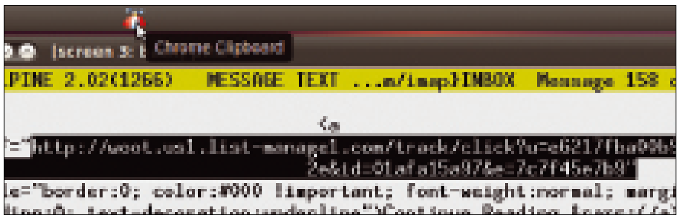
MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

Confused Users

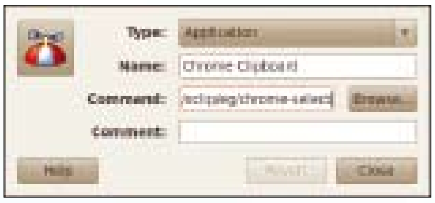
Applications can implement one or both of these approaches, which can cause confusion and frustration among users in the wild. Although, for example, a simple `xterm` only supports primary selection, the Adobe Flash player only supports the clipboard mechanism. If you want to transfer text from an `xterm` to a form rendered in Flash player, you're out of luck. The Google Chrome browser supports both primary and clipboard selection, but if the evernote.com web application is running in Chrome, primary selection no longer works. At this point, confusion reigns.

If you copy an URL from a text window and want to use it to access and display the corresponding website in a browser, you may discover that the URL in the text contains a line break and that



```
|screen 3: ł Chrome Clipboard
PINE 2.02(1266)   MESSAGE TEXT ...m/imap}INBOX  Message 158
=“http://woot.us1.list-managel.com/track/click?u=e6217fba9b
                             2e&id=01afa15a97&e=7c7f45e7b9”
le="border:0; color:#000 !important; font-weight:normal; margi
```

Figure 1: The Perl script hiding behind the propeller hat grabs the URL from the primary text selection and launches Chrome with it.



Type: Application
Name: Chrome Clipboard
Command: /eclipseg/chrome-select
Comment:

Figure 2: The Chrome Launcher, which launches the Chrome browser with the URL highlighted in X Window's primary selection.

the browser only understands the first part. Now, I must admit to being a dyed-in-the-wool, hardcore Unix user. I read my emails in `pine` in a `screen` window, and to make sure that URLs of this kind are transferred correctly, I set the value of `editor.singleLine.pasteNewlines` to 3 in Firefox's `about:config`.

Help for Chrome

Unfortunately, Chrome can't do this, and that explains why I have a propeller hat icon in my Gnome panel at the top edge of the screen with the script from Listing 1 hiding behind it (Figure 1). This script [2] picks up the URL from the primary selection, removes any blanks and line breaks and passes the cleaned URL to the Chrome browser.

To access the primary desktop selection, Listing 1 relies on the CPAN Clipboard [3] module; under the hood, the module uses the `xclip` program, which belongs to Linux's graphical underpinnings, the X Window system. You can install `xclip` on Ubuntu by typing `sudo apt-get install xclip`. Compared to using `xclip` directly, the CPAN module offers the benefit that it will also work on a Mac or Windows computer, because it automatically switches to the native clipboard mechanisms on these two platforms.

Propeller Hat Take-Off

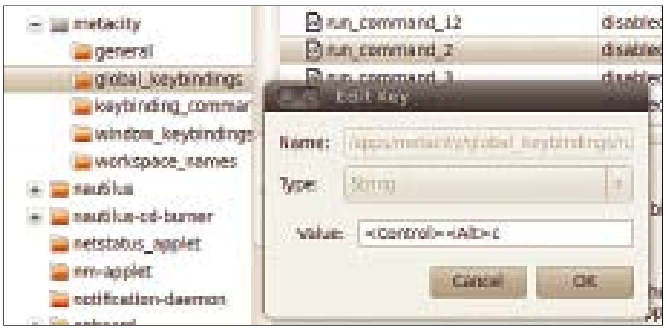
Because the clipboard module was installed using the CPAN shell with my user ID, in my home directory, and not under `root`, `local::lib` points the Perl interpreter to the local directory. The `Clipboard` class's `paste` method picks up the content of the current primary selection. Line 10 uses a regular expression to remove undesirable line breaks, and the `system` command calls the Chrome browser with the URL as its argument. If the script in the form of a “Custom Application Launcher” is added to the panel with an easily identifiable icon (Figure 2) by right-clicking and selecting `Add to Panel`, the user only needs to click on the propeller hat after selecting the URL (without the copy command) to tell the Chrome browser to start up with the sanitized URL so that the browser can reluctantly follow a link.

Desktop Button Copy

To solve the communication problem that I mentioned earlier on between `xterm` and Flash player, Listing 2 grabs the primary selection and pushes it onto the clipboard. To allow users to run this command while working in an application that doesn't understand the clipboard mechanism, the `gconf-editor` program in Figure 3 defines a global key map in Gnome's Metacity window manager. If the user presses the keyboard shortcut, `CTRL-ALT-c` (defined in the `global_keybindings` section) after making a selection, Metacity triggers the configurable `run_command_2` command,

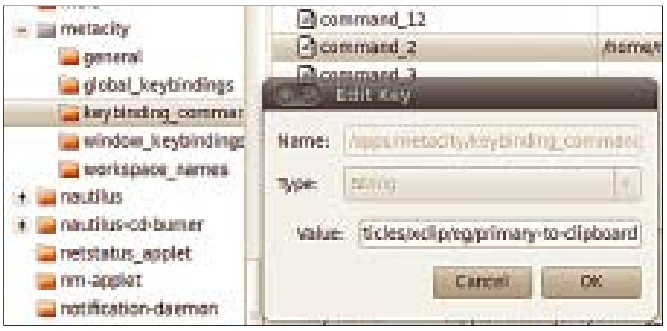
LISTING 1: chrome-select

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use local::lib;
04 use Clipboard;
05
06 my $browser =
07   "/usr/bin/google-chrome";
08 my $url = Clipboard->paste;
09
10 $url =~ s/\s+//g;
11
12 system($browser, $url);
```



Name: /apps/metacity/global_keybindings/r
Type: String
Value: <Control><Alt>c

Figure 3: In Gnome's Metacity window manager, pressing the keyboard shortcut Ctrl+Alt+c runs the command 2 …



Name: /apps/metacity/keybinding_command
Type: String
Value: ticles/xclip/eg/primary-to-clipboard

Figure 4: … which in turn executes the primary-to-clipboard script in order to copy the mouse selection to the clipboard.

LISTING 2: primary-to-clipboard

```
#!/usr/local/bin/perl
use local::lib;
use Clipboard::Xclip;

my $primary =
  Clipboard::Xclip
  ->paste_from_selection(
 "primary");

Clipboard::Xclip
  ->copy_to_selection(
 "clipboard", $primary);
```

LISTING 3: clipboard-stack

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use local::lib;
04 use Clipboard;
05 use YAML
06   qw(DumpFile LoadFile);
07
08 my ($home) = glob "~";
09 my $clipboard =
10   "$home/.clipboard";
11
12 my $stack = [];
13 $stack = LoadFile($clipboard)
14   if -f $clipboard;
15
16 my ($command) = @ARGV;
17
18 die "usage: $0 [push|pop]"
19   if !defined $command;
20
21 {
22  no strict 'refs';
23  &$command($stack);
24 }
25
26 DumpFile($clipboard, $stack);
27
28 #############################
29 sub push {
30 #############################
31  my ($stack) = @_;
32
33  push @$stack,
34    Clipboard->paste;
35 }
36
37 #############################
38 sub pop {
39 #############################
40  my ($stack) = @_;
41
42  Clipboard->copy(
43   pop @$stack);
44 }
```

LISTING 4: multipaste

```
01 #!/usr/local/bin/perl -w
02 use strict;
03
04 my @words =
05   qw(yes no maybe so);
06
07 for my $word (@words) {
08
09  open my $pipe, "|-",
10    qw(xclip -verbose
11    -selection CLIPBOARD -loops 2);
12
13  print $pipe $word;
14  close $pipe or die;
15 }
```

which points to the script from Listing 2, as set in the `keybinding_commands` section (Figure 4).

This lets the user select text in an application that does not support the clipboard and then bundle the text off into the clipboard by pressing Ctrl + Alt + c. In the target application, which doesn't support the primary selection, all you need to do is to press Ctrl + V to insert the desired text. The script uses the undocumented `paste/copy_from_selection` methods from the derived Clipboard::Xclip class, because the base class `Clipboard` doesn't allow any distinctions between the two buffers.

Cut and Paste on Steroids

Some readers might wish for several cut-and-paste buffers at times to collect N different sections and then deposit them all in another window. On a normal Gnome desktop, this would involve the user jumping back and forth N times between the source and the target, taking a snippet from the source and dumping it into the target each time.

To solve this problem, Listing 3 implements a persistent stack onto which the user keeps pushing the content of the primary selection. For this to happen, `clipboard-stack push` creates more space in the buffer for more selections with each new one. To replace the current selection with one recently stored in the stack, the user calls `clipboard-stack pop`. The script stores the values in the `@$stack` array, using the CPAN `YAML` module to write the array content persistently to the `.clipboard` file in the user's home directory.

When the script is called the next time, the YAML module's `LoadFile()` method reads the stored data. Depending on whether the user calls the script with the `push` or `pop` argument, the call to `&$command( $stack )` in line 23 jumps to one of the functions defined below. Because the script is not allowed to dynamically call functions given as text strings in `strict` mode, the `no strict 'refs'` pragma in line 22 temporarily lifts this restriction. The `push()` function defined in line 29 ff. uses the `Clipboard` class' `paste()` method to read the selected text and push it onto the stack. The `pop()` function defined in line 38 copies the archived content to the primary selection by calling the `copy()` method. When the user presses the middle mouse button, the archived content tumbles out.

To avoid the need for users to call Perl scripts in some convoluted way while juggling with selection content, there are two entries in the Gnome panel for the selection stack, represented by two very attractive icons in the form of arrows (which I discovered under `/usr/share/icons/Human/48x48/actions` as `edit-redo.png` and `edit-undo.png`) taking care of `push` and `pop` (Figures 5 and 6).

Note that a calling `push` and `pop` in quick succession doesn't accomplish anything useful. You can paste the primary selection anywhere right away, but if you want to save the selection in the archive before making an additional selection, the stack offers the required functionality in the form of `push`. If you no longer need the current selection, you can retrieve the next archived selection by issuing a `pop`.

Click, Click, Click

Instead of retrieving the next chunk lined up in the clipboard archive after

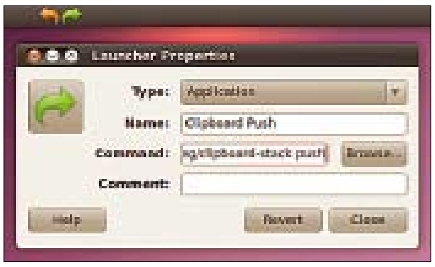

Launcher Properties
Type: Application
Name: Clipboard Push
Command: ...g/clipboard-stack push
Comment:

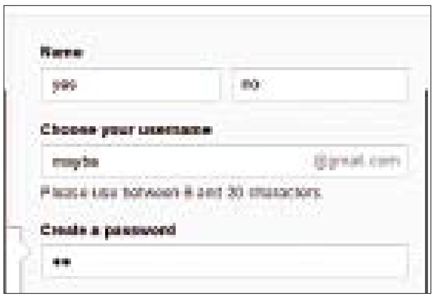
Figure 5: Adding the Clipboard Stacker to the Gnome panel.

```
local::lib;
Clipboard;
YAML  qw(DumpFile LoadFile);
```

Figure 6: The Green Arrow pushes the current selection onto the stack and the orange arrow retrieves it from there.

every paste, it would be downright revolutionary to allow the user to click N times and each time automatically follow the primary selection with the next archived item. This would help, say, by filling out web or PDF forms if you know the order of the fields and have organized the data in extended clipboard buffers in the correct order. In this case, the user would just need to jump from field to field and issue a paste command in each one.

Listing 4 implements this approach by calling `xclip` directly. In contrast to the script introduced earlier, instead of using the primary selection, it uses the clipboard buffer, which is filled by pressing Ctrl + C (*Edit* | *Copy*) and emptied by pressing Ctrl + V (*Edit* | *Paste*). The reason is just because I originally wrote the script for filling out form data in a Flash player application. If you leave out `-selection CLIPBOARD`, it will instead access the primary selection; then, rather than pressing Ctrl + V multiple times, you can just keep hitting the middle mouse button.



Name
yes
no
Choose your username
maybe
@gmail.com
Please use between 6 and 30 characters
Create a password
••

Figure 7: The user first called `multipaste`, then moved from field to field, and pressed Ctrl+V in each case.

The script iterates over a row of words (yes, no, maybe so), pushing them one by one onto the clipboard and waiting (`-loops 2`) rounds until the user releases the current selection with the paste command. This action reanimates the blocking `xclip` command and the Perl script loop enters the next round. A subsequent call to `xclip` is then handed the next word via the pipeline opened by `open`.

In Figure 7, the user has accessed the various fields in a registration form using the mouse or the tab key, after calling `multipaste` in another window, and has pressed Ctrl + V to trigger a paste process in each case. One after another, the words bundled into the buffer come tumbling out.

In other words, if you need to regularly fill out of the same web forms, and you can't automate the process by using APIs or a screenscraper, you can use scripts such as `multipaste` to prepare the values in the right order. Then, later you can zip from field to field and enter the correct values without having to think, without worrying about typos, in fact without any worries at all. ■■■

INFO

[1] "X Selections, Cut Buffers, and Kill Rings" by Jamie Zawinski: *http://www.jwz.org/doc/x-cut-and-paste.html*

[2] Listings for this article: *ftp://www.linux-magazin.de/pub/listings/magazin/2012/03/Perl*

[3] CPAN clipboard module: *http://search.cpan.org/~king/Clipboard-0.13/*

**1/2 Ad
with bleed**