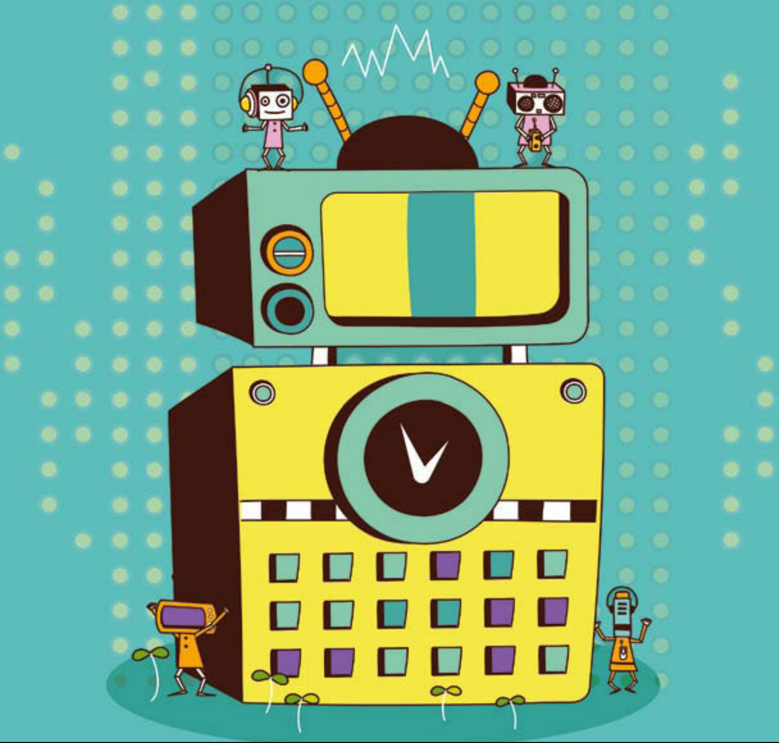


Edit text conveniently in a terminal window with Slap

Leaps and Bounds

The Slap editor supports easy navigation, even through large volumes of data, but the promising software has a couple of hiccups along the way. *By Valentin Höbel*

The choice of text editors on Linux is seemingly infinite – whether for the terminal or with a sophisticated graphical user interface, you are likely to find the right tool for any application if you look hard enough. Although newcomers tend to choose intuitively usable programs like Nano [1] or Pico [2] if they need to edit text-based files at the command line, more experienced users may prefer versatile veterans like Vim or feature-rich classics like Emacs.

Migrating from one editor to another is not easy, mainly because the many keyboard shortcuts differ, stifling your ability to move between editors. The Slap [3] text editor is gathering a fan community rapidly by positioning itself precisely between the newcomer and professional editor camps. Instead of relying on many hotkeys, it uses mouse support and interacts with the Linux clipboard.

Young and Different

Slap is unusual in many respects. Whereas other programs in this field can look back on a long history and are typically the work of many individuals – or even whole communities – Slap is a one-man project by Dan Kaplun [4]. Unlike Nano, Vim, or Emacs, Slap is based on the JavaScript scripting language, which is more typically used for websites or web applications.

The first traces of Slap go back to April 2014, when Kaplun (a.k.a. *beardtree*) posted the first commit on GitHub [5]. Since then, the developer has added many features and shortcuts to the editor. Slap has thus become a lean alternative for editors like Vim or Emacs and has so far avoided the bloat associated with them.

Installation

Because of its fairly short development history – and probably also because of its JavaScript underpinnings – Slap lacks packages in the distribution repositories for now. Instead, you install the libraries required for the tool up front and then install the editor itself via the Node.js package management system (NPM).

Listing 1 shows the installation on Ubuntu 12.04 and 14.04. You need to be root to use all of these commands. During tests in our lab, I used the 64-bit variants of Ubuntu. Line 5 uses the less secure HTTP for the download instead of HTTPS, because Curl on Ubuntu fails to identify the SSL certificate correctly that is stored on NPMJS for Slap. The tool is

LISTING 1: Installing Slap

```
$ apt-get update
$ apt-get install curl
$ curl -sL https://deb.nodesource.com/setup | sudo bash -
$ apt-get install nodejs xclip git
$ npm config set registry http://registry.npmjs.org/
$ npm install -g slap
```

AUTHOR

Valentin Höbel works as a Cloud architect for VoIP specialists NFON AG in Munich. In his free time, you will either find him playing table soccer, or checking out the latest open-source technologies.

also available for other distributions; you will find details of the installation procedures for these distributions online [6].

After running the commands shown in Listing 1, you can launch the editor by typing the `slap` command in a terminal. Optionally, you can pass a file name into the command if you will be editing an existing text file.

Features

Although Slap runs in a console window, when launched, the program comes up with an interface split into two columns; if needed, you can even use the mouse to control it – this is a little unusual for experienced users, but a pointing device does remove the need to learn a number of new, complex hotkeys.

The left half of the window contains a folder and file tree, and the right side shows open files. As you can see in Figure 1, Slap also supports syntax highlighting for numerous scripting and programming languages.

Besides the ability to highlight code, you will find very few aids for developers – only parenthesis matching and automatic indenting help you when typing. You also will look in vain for features such as auto-completion of existing variables or class names.

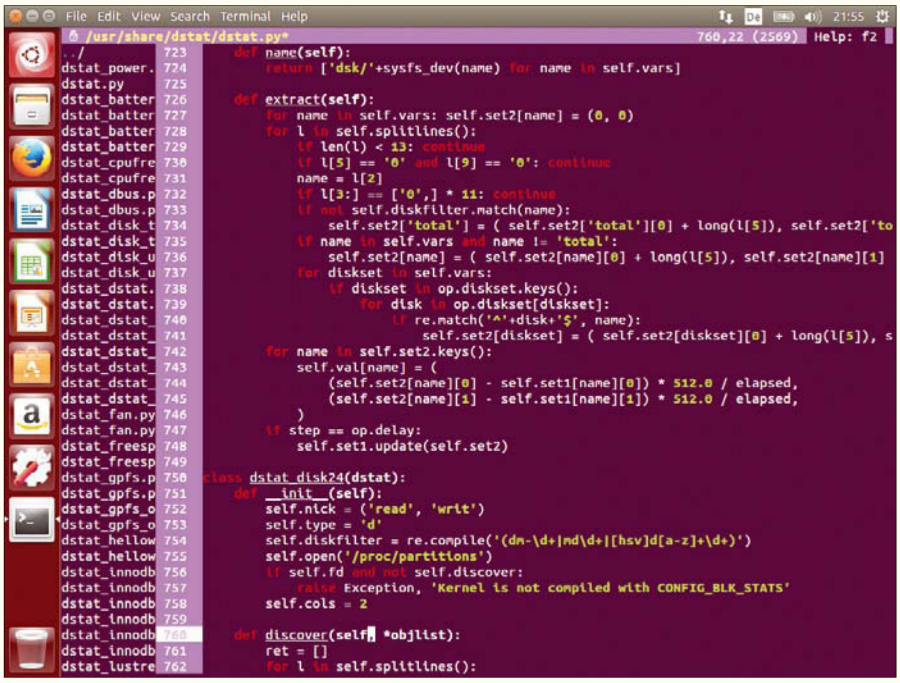
You do, however, have the option of searching open files with the help of regular expressions or sharing content via the Linux clipboard. In tests, cutting and pasting with content from a website opened in Firefox worked pretty well. As you would expect of any good application, you can undo or redo the last action.

Getting Started

Getting started with Slap is not difficult. After launching the program, you can conveniently navigate the file tree with your mouse, including scroll support. However, if you open directories with a large number of files, smooth scrolling suddenly deteriorates into an annoyingly jerky process. The ability to jump to files with names that start with specific letters by pressing the matching keys is something that Slap, in contrast to other applications, currently does not offer.

My tests revealed another weakness when I dumped new files in the background into the directory I was currently viewing in the Slap file view: Slap did not immediately see these files; in fact, I needed to change to a different directory and then back before the files finally appeared.

When you open a file, its content is displayed in the right-hand part of the window. In the lab, opening a file took quite a while in some cases; this was particularly true



```
/usr/share/dstat/dstat.py*                                   760,22 (2569)  Help: f2
../                def name(self):
dstat_power.           return ['dsk/'+sysfs_dev(name) for name in self.vars]
dstat.py
dstat_batter       def extract(self):
dstat_batter           for name in self.vars: self.set2[name] = (0, 0)
dstat_batter           for l in self.splitlines():
dstat_batter               if len(l) < 13: continue
dstat_cpufre               if l[5] == '0' and l[9] == '0': continue
dstat_cpufre               name = l[2]
dstat_dbus.p               if l[3:] == ['0',] * 11: continue
dstat_dbus.p               if not self.diskfilter.match(name):
dstat_disk_t                   self.set2['total'] = ( self.set2['total'][0] + long(l[5]), self.set2['to
dstat_disk_t               if name in self.vars and name != 'total':
dstat_disk_u                   self.set2[name] = ( self.set2[name][0] + long(l[5]), self.set2[name][1]
dstat_disk_u               for diskset in self.vars:
dstat_dstat.                   if diskset in op.diskset.keys():
dstat_dstat.                       for disk in op.diskset[diskset]:
dstat_dstat_                           if re.match('^'+disk+'$', name):
dstat_dstat_                               self.set2[diskset] = ( self.set2[diskset][0] + long(l[5]), s
dstat_dstat_           for name in self.set2.keys():
dstat_dstat_               self.val[name] = (
dstat_dstat_                   (self.set2[name][0] - self.set1[name][0]) * 512.0 / elapsed,
dstat_dstat_                   (self.set2[name][1] - self.set1[name][1]) * 512.0 / elapsed,
dstat_fan.py               )
dstat_fan.py           if step == op.delay:
dstat_freesp               self.set1.update(self.set2)
dstat_freesp
dstat_gpfs.p   class dstat_disk24(dstat):
dstat_gpfs.p       def __init__(self):
dstat_gpfs_o           self.nick = ('read', 'writ')
dstat_gpfs_o           self.type = 'd'
dstat_hellow           self.diskfilter = re.compile('(dm-\d+|md\d+|[hsv]d[a-z]+\d+)')
dstat_hellow           self.open('/proc/partitions')
dstat_innodb           if self.fd and not self.discover:
dstat_innodb               raise Exception, 'Kernel is not compiled with CONFIG_BLK_STATS'
dstat_innodb           self.cols = 2
dstat_innodb
dstat_innodb       def discover(self, *objlist):
dstat_innodb           ret = []
dstat_lustre           for l in self.splitlines():
```

Figure 1: The Slap editor uses a colorful layout with two panes.

TABLE 1: Hotkeys in Slap

Key	Function
F2	Show help
Ctrl + Q	Quit the editor
Ctrl + O	Jump to the navigation panel and open a file
Ctrl + S	Save file
Ctrl + F	Start search
Ctrl + G	Go to the stated line
Ctrl + Left arrow	Jump one word to the left
Ctrl + Right arrow	Jump one word to the right
Ctrl + Up arrow	Move one line up
Ctrl + Down arrow	Move one line down
Ctrl + A	Go to the start of the line
Ctrl + E	Go to the end of the line
Ctrl + M	Go to the matching bracket
Ctrl + W	Delete the word to the left of the cursor
Ctrl + Del	Delete the word to the right of the cursor
Ctrl + K	Delete the current line
Ctrl + C	Copy selected content
Ctrl + V	Insert the clipboard content at the current position
Ctrl + X	Cut the selected content
F10	Show or hide the navigation pane
Ins	Change to edit mode

of files with a large volume of source code, which took Slap a couple of seconds to enable syntax highlighting.

After loading in full, you can start editing directly. Pressing F2 lists the most frequently used hotkeys (Figure 2). Table 1 lists the hotkeys and a couple of other shortcuts as a reference.

After opening an existing file, or after completing a fair amount of typing, you can press the left mouse button to navigate through the lines and words. Alternatively, you can use the arrow keys in combination with the Control key. To jump to a specific location in the file, you can either search or specify the line number.

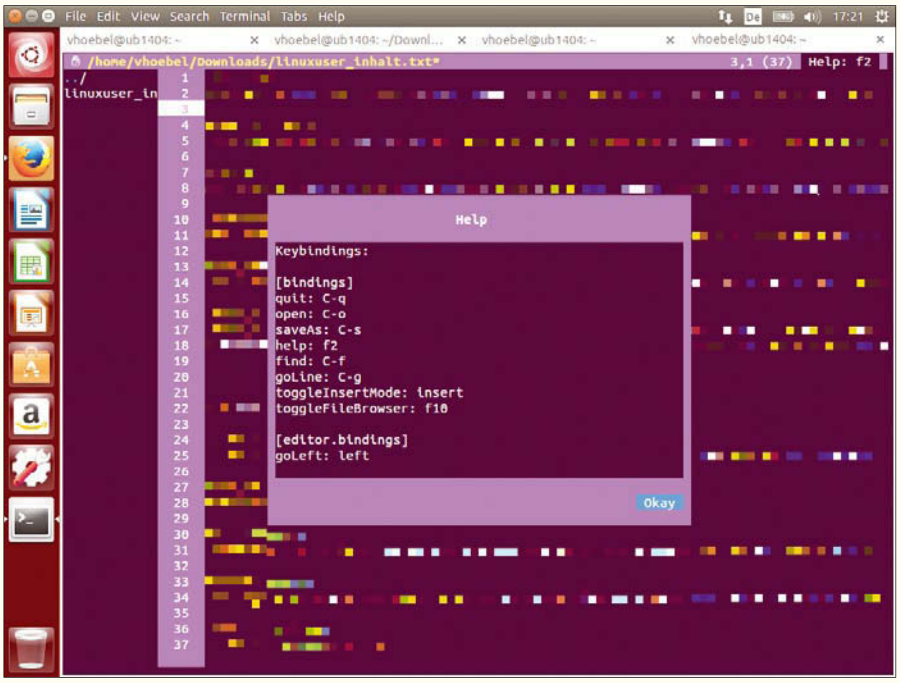
To delete a specific line, use the hotkeys or the mouse to navigate to the line in question and press Ctrl + K to remove it. Alternatively, you can select the whole line with the left mouse button and then press the Delete key twice: The first key press deletes the content, and the second deletes the line itself.

After just a few minutes, you should be familiar with the controls, characterized by a balanced mix of keyboard shortcuts and mouse movements. Editing text is easy enough, but trying to insert long lines of text in the editor can be annoying. Because Slap has no automatic line wrap, you need to move the cursor to the end of the line to see all the content. If you decide to copy and paste content, you will be pleased to see that the familiar keyboard shortcuts Ctrl + C and Ctrl + V take on their normal roles.

Configuration

In the style of Vim, which may be familiar to you, Slap also has a configuration file with various customization options. If you use the editor frequently, take a quick look at `slap.ini` (Figure 3), which you will find in `/usr/lib/node_modules/slap/`. The content of this file defines the hotkeys, the interface colors, and the syntax highlighting.

Editing this file directly is not a good idea. If you want to modify some of the functions to suit your own needs, the cleaner (and – in case of an update – safer) approach is to edit the `.slaprc` file that Slap creates in your own home directory when

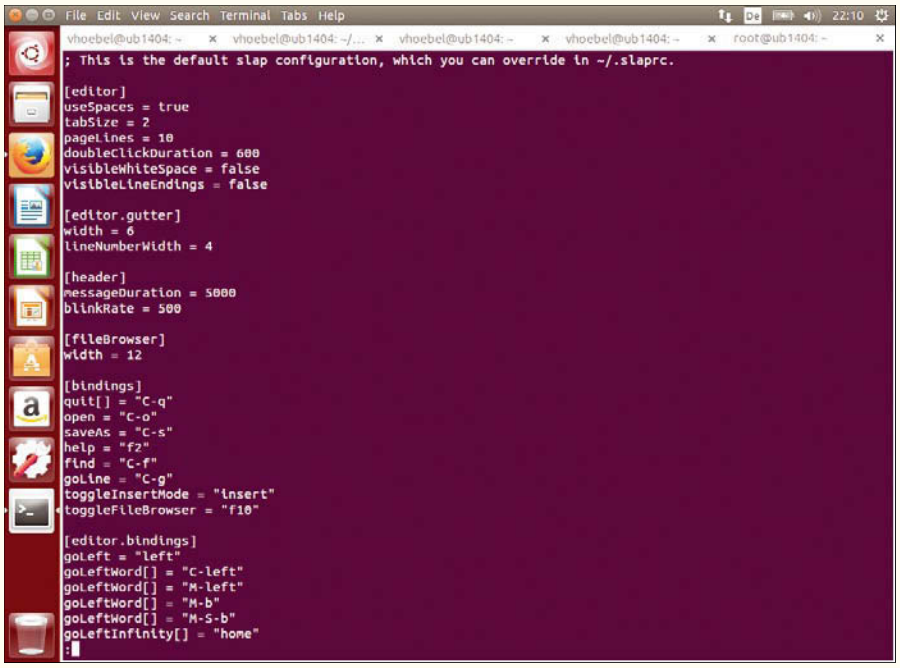


```
/home/vhoebel/Downloads/linuxuser_inhalt.txt*          3,1 (37)  Help: f2
../
linuxuser_in

                     Help
Keybindings:

[bindings]
quit: C-q
open: C-o
saveAs: C-s
help: f2
find: C-f
goLine: C-g
toggleInsertMode: insert
toggleFileBrowser: f10

[editor.bindings]
goLeft: left
```

Figure 2: Pressing F2 displays a short list of frequently used hotkeys.

first launched. To do so, open the file with any editor (e.g., Slap) and type the values you want to change. Make sure you break your personal configuration file down into sections as in the `slap.ini` template. Note that copying the selected content from the system global configuration to `~/.slaprc` and then modifying its content is the fastest way to create your own configuration.

Essentially, you can modify any values – including the hotkeys – to suit your needs. The exceptions are the shortcuts stored in the `slap.ini` file, and anything the terminal would consider to be an escape character. Figure 4 shows an example of what a custom Slap configuration can look like – the modified configuration is already active in the figure.



```
; This is the default slap configuration, which you can override in ~/.slaprc.

[editor]
useSpaces = true
tabSize = 2
pageLines = 10
doubleClickDuration = 600
visibleWhiteSpace = false
visibleLineEndings = false

[editor.gutter]
width = 6
lineNumberWidth = 4

[header]
messageDuration = 5000
blinkRate = 500

[fileBrowser]
width = 12

[bindings]
quit[] = "C-q"
open = "C-o"
saveAs = "C-s"
help = "f2"
find = "C-f"
goLine = "C-g"
toggleInsertMode = "insert"
toggleFileBrowser = "f10"

[editor.bindings]
goLeft = "left"
goLeftWord[] = "C-left"
goLeftWord[] = "M-left"
goLeftWord[] = "M-b"
goLeftWord[] = "M-S-b"
goLeftInfinity[] = "home"
:
```

Figure 3: The Slap configuration file offers users many options for customizing the editor.

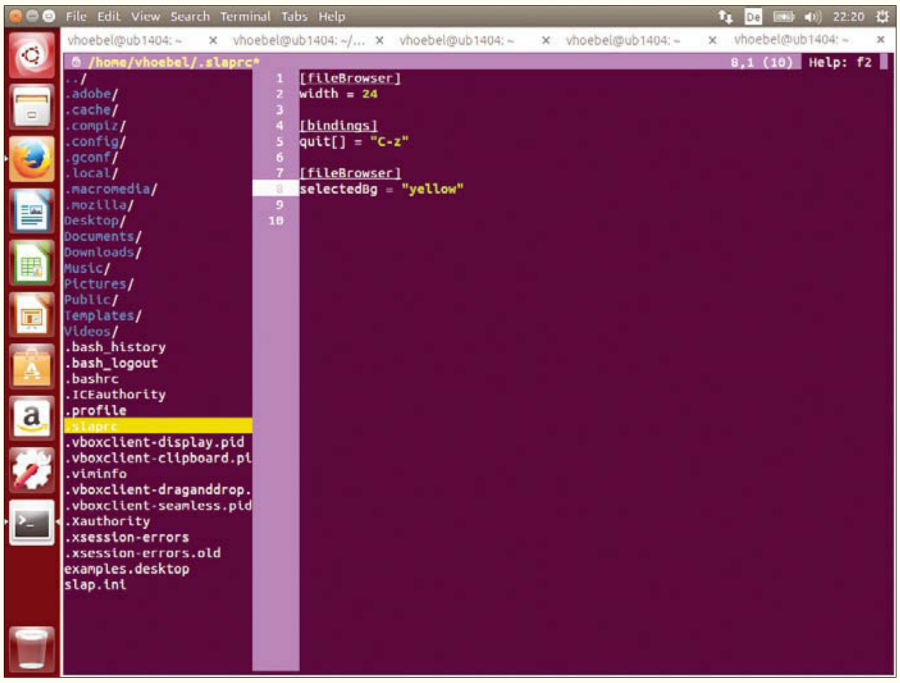
Conclusions

The Slap text editor with its JavaScript underpinnings takes an unusual approach but is a welcome change to text editors without mouse support. The mix of familiar hotkeys and mouse support lends itself to efficient and convenient editing.

Navigation is okay for the most part but shows a couple of weaknesses here and there. This fairly young editor still has some potential for improvement. For example, performance drops when you edit larger files.

Although the author claims similarity to Sublime Text [7], I found very few likenesses in the lab. The original is more powerful and mainly targets experienced power users and developers.

Wherever you look, though, Slap impresses with its adaptability, which is why I can unequivocally recommend the editor for newcomers and experienced users alike – at least on your personal workstation. The software is definitely not designed for servers, and you are very likely to get into trouble with the system administrator if you try to install JavaScript just to run a text editor. ■■■



```
/home/vhoebel/.slaprc*                      8,1 (10)  Help: f2
../                          [fileBrowser]
.adobe/                      width = 24
.cache/
.compiz/                     [bindings]
.config/                     quit[] = "C-z"
.gconf/
.local/                      [fileBrowser]
.macromedia/                 selectedBg = "yellow"
.mozilla/
Desktop/
Documents/
Downloads/
Music/
Pictures/
Public/
Templates/
Videos/
.bash_history
.bash_logout
.bashrc
.ICEauthority
.profile
.slaprc
.vboxclient-display.pid
.vboxclient-clipboard.pi
.viminfo
.vboxclient-draganddrop.
.vboxclient-seamless.pid
.Xauthority
.xsession-errors
.xsession-errors.old
examples.desktop
slap.ini
```

Figure 4: The editing and navigation windows already reflect the configuration shown here.

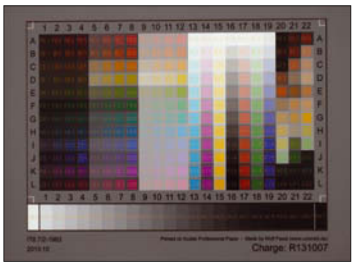
INFO

[1] Nano: *http://www.nano-editor.org*
[2] Pico: *http://www.washington.edu/pine/*
[3] Slap: *https://github.com/slap-editor/slap*
[4] Dan Kaplun: *http://beardtree.com*
[5] First commit for Slap on GitHub: *https://github.com/slap-editor/slap/commit/de718604ca4a2ca85f4cf83cb1aac405458f5cf3*
[6] Slap README: *https://github.com/slap-editor/slap/blob/master/README.md*
[7] Sublime Text: *http://www.sublimetext.com*

HIGHLIGHTS

FEATURES

LINUXUSER



fedora 22™
64-bit

- Improved software installer
- Gnome 3 desktop
- Developer tools
- GNOME Boxes
- Built-in Docker support

manjaro
Linux KDE Edition

- Based on Arch Linux
- Easy software installation
- Auto-detects hardware
- Pre-installed codecs
- Games

SEE PAGE 6 FOR DETAILS

GitHub's configurable editor

Hacker Kitchen

The Atom code editor from GitHub is a highly configurable free application. Just one year old, even at this early stage, the mix looks very promising. *By Kristian Kißling*

When GitHub announced Atom [1] in June 2014, many observers sighed: Does the world really need yet another text editor? Well, the makers of GitHub are convinced it does. Sublime Text [2] might be convenient, but it is not genuinely configurable. On the other hand, Emacs and Vi are highly configurable, but not exactly convenient for the uninitiated.

The makers of GitHub know pretty well how the open source world ticks, and they have gotten very few things wrong thus far. Atom is no exception. Vi and Emacs only work as sustainably and well as they do because they are both open and have a large community. Consequently, GitHub plans to be involved long-term with the editor.

Atom 1.0 was released on June 25, 2015, under an MIT license, but that does not rule out GitHub offering an enterprise variant of the editor at some time.

Atom Model

Atom is intended to help developers program desktop and web applications across multiple platforms. The software comes with syntax highlighting for various programming languages – from JavaScript, through Perl and Python, up to C, C++, or Java. At the end of the day, Atom is a variant of the Chromium browser, and the windows are no more than locally rendered web pages, from which the content can access the Node.js API.

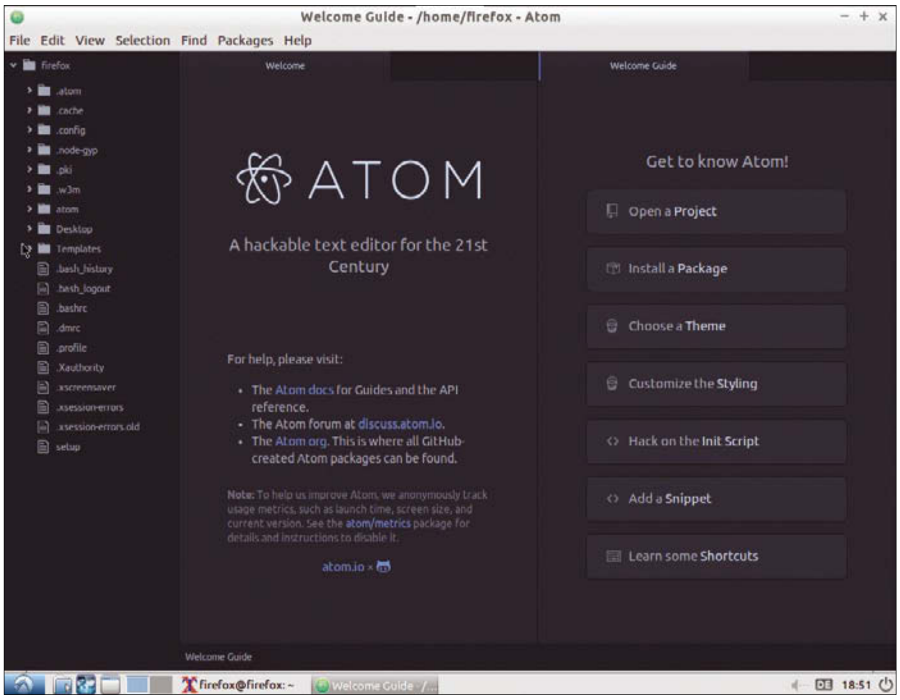
Atom consists of an easily manageable core (Atom Core), with most other components available as Atom packages managed by the Atom Package Manager (APM). The core and the packages run on the Electron framework [3], which was previously known as the Atom Shell. The Atom Shell takes care of automatic updates, includes a Windows installer, creates crash reports, and delivers notifications. The functions are accessible through JavaScript APIs.

For simplicity's sake, the developers use and recommend CoffeeScript instead of JavaScript for working with Atom and Less as a replacement for CSS. That said, the editor can also be extended to include JavaScript and CSS.

Control Panel

After the installation – which could be somewhat less complex (see the box "Installing Atom") – developers are taken to a graphical interface (Figure 1). It comprises various panes that can be zoomed and rearranged as needed. The individual files in which a developer works are known as "buffers" in Atom-speak.

After starting Atom, you find the directory tree for the current project in the left-hand pane. The code belongs in the large field in the middle. Pressing F11 takes you to full-screen mode; Atom opens the settings as additional tabs when you press



A hackable text editor for the 21st Century

For help, please visit:

- The Atom docs for Guides and the API reference.
- The Atom forum at discuss.atom.io.
- The Atom org. This is where all GitHub-created Atom packages can be found.

Note: To help us improve Atom, we anonymously track usage metrics, such as launch time, screen size, and current version. See the atom/metrics package for details and instructions to disable it.

Figure 1: Atom comprises flexibly arranged multiple panes.

Ctrl+, (Ctrl+<comma>), or when you select *Edit* | *Preferences*.

If you are uncertain how to proceed, you can tell the editor to show all available commands with Ctrl+Shift+P. A useful Flight Manual is also available online [6].

In the Reactor

Although the editor is still a fairly young project, it already comes with a number of useful features. To indent one or multiple lines of code, you select the text and pressing Tab. Shift+Tab moves the block of code to the left.

Auto-completion is also implemented, and the developers promise to expand this in the future. Right now, you can press Ctrl+Space to enable the feature and display alternatives to the element underneath the cursor.

When you access the settings (Ctrl+,) and mark the *Soft Wrap* checkbox below the Editor Settings section, the editor wraps the line at the end of the screen. Checking *Soft Wrap At Preferred Line Length* wraps the code after *n* characters. The line break border is marked by an invisible line; by default the length is 80 characters, but you can change this in the *Preferred Line Length* text box in the same section.

If you enable the *Soft Tabs* option, the editor automatically converts tabs to space characters; you can even define the number of space characters in the settings (*Tab Length* drop-down box).

Folding means hiding blocks of code (e.g., functions, loops) to simplify your view of the code. Folding is accomplished either with the small triangles that appear when mousing over the gutter or with Alt+Ctrl+[(fold) or Alt+Ctrl+] (unfold). The triangles are sometimes difficult to hit, causing the editor to jump suddenly to other snippets of code.

Weaknesses and Strengths

Atom still has a couple of issues, the most noticeable being that Atom cannot handle files greater than 2MB; the developers are working on a solution. They are also looking for a way to speed up the editor's slowish launch behavior and GUI reaction speed. The response was a tad too slow in our lab, but that might have been the result of using a virtual machine.

One feature that really works (surprise, surprise) is integrated version management with Git and GitHub. For example, using Ctrl+Alt+Z restores the

INSTALLING ATOM

Installation on Ubuntu 14.04 with an LXDE desktop involved some overhead, mainly attributable to the Node.js environment that Atom needs. Prebuilt packages exist, but if you want to compile the software yourself, you first need to install the matching files:

```
sudo apt-get install build-essential git ⮐
  libgnome-keyring-dev fakeroot gconf-service libnss3
```

The next step is to download the latest version of Node.js, which is available on NodeSource.com [4]:

```
curl -sL https://deb.nodesource.com/setup | sudo bash -
```

The command sequence basically fetches a setup script for the current node version and runs it with root privileges. More cautious users will probably want to insert `more` after the pipe to watch the script run. The command

```
sudo apt-get install -y nodejs
```

installs the latest variant of the framework – if it runs without any complaints.

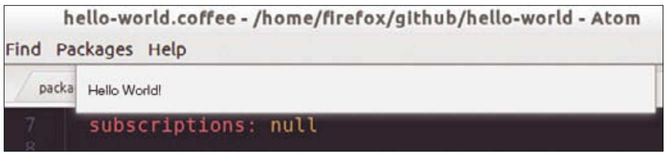
If the `which node` command does not show any results, the binary below `/usr/bin/` that Atom is looking for might be named `nodejs` instead of `node`. The command

```
sudo update-alternatives --install /usr/bin/node node ⮐
  /usr/bin/nodejs 10
```

provides a remedy. The next set of commands clones the Atom repository on GitHub and then checks out the latest Atom release from the `atom` directory:

```
git clone https://github.com/atom/atom
cd atom
git fetch -p
git checkout $(git describe --tags `git rev-list ⮐
  --tags --max-count=1`)
```

Finally, you need to run the build script and then launch the JavaScript Task Runner Grunt [5] to install the executable `atom` file and the `apm` package manager in `/usr/local/bin`. Incidentally, you can also use Grunt to create a Debian package via `script/grunt mkdeb`.


hello-world.coffee - /home/firefox/github/hello-world - Atom

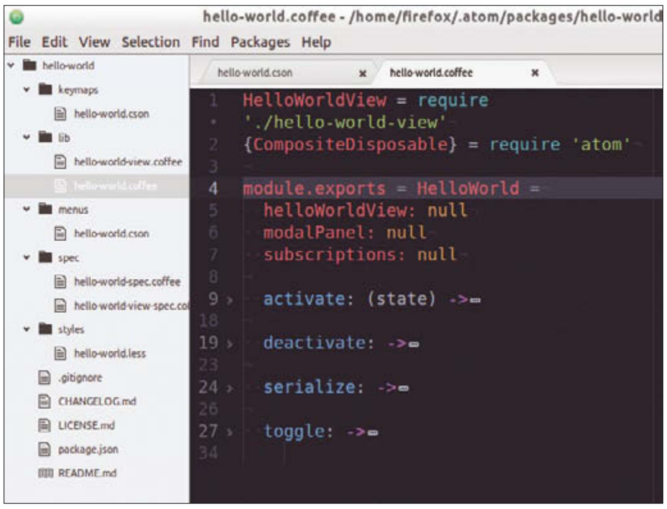
Hello World!

```
subscriptions: null
```

Figure 2: **The** `hello-world` **package conjures up an unspectacular window with the standard message on the screen.**

TABLE 1: GitHub Keyboard Shortcuts for Atom

Shortcut	Action
Alt+G+O	Open a file on GitHub
Alt+G+B	GitHub info relating to the last changes of a file
Alt+G+H	View GitHub history of a file
Alt+G+C	Copy the GitHub URL of the current file
Alt+G+R	Branch comparison with GitHub


hello-world.coffee - /home/firefox/.atom/packages/hello-world

```
HelloWorldView = require
'./hello-world-view'
{CompositeDisposable} = require 'atom'

module.exports = HelloWorld =
  helloWorldView: null
  modalPanel: null
  subscriptions: null

  activate: (state) ->

  deactivate: ->

  serialize: ->

  toggle: ->
```

Figure 3: **In the tree view on the left is the structure of the automatically generated package. On the right is the code that handles the program logic.**

last good version of a file and is equivalent to:

```
git checkout HEAD -- /<path>
git reset HEAD -- /<path>
```

If you want to know which of your files is not being tracked, you can press Ctrl+Shift+B (i.e., `git status`). With the help of the `language-git` package, you can highlight Commit, Merge, and Rebase messages for Git. The `git-diff` package allows the editor to highlight modified lines.

If you're looking for information about a file that you are currently editing on GitHub, a number of keyboard shortcuts (Table 1) can help.

Hackable to the Core

Another of Atom's benefits is the ability to customize the software. A Package Generator creates a skeleton for the package; in our lab, I used the classic `hello-world` as its title. This package does nothing but display window with a *Hello World!* message (Figure 2).

Pressing Ctrl+Shift+P then starts the command search and calls *Package Generator: Generate Package*. The boilerplate code created by this ends up in the `github/<Projectname>` directory; the project name in this example is `hello-world`. At the same time, the directory tree generated by this process is shown in the tree view on the left; Figure 3 shows the skeleton of the `hello-world` package.

The second to last file in the treeview is `package.json`, which contains a variety of metadata for the package (Figure 4), including domain, the version number, the potential GitHub repository, as well as the path to the executable files in the `lib` subdirectory.

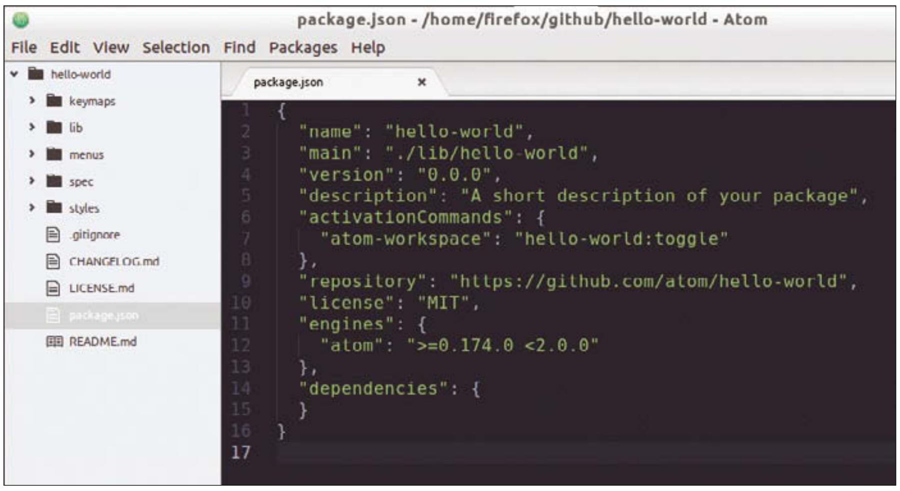
The `lib` subdirectory contains two files: `hello-world.coffee` and `hello-world-view.coffee`, which organizes the graphical user interface (Listing 1). Figure 3 shows the code with the package logic. The methods `activate`, `deactivate`, `serialize`, and `toggle` have been folded to show the structure; Listing 2 contains the invisible code.

Listing 1 generates the root element (`@element`) and assigns it as a child of the `message` element (line 12); then, the code defines how the element responds to `getElement()` and `destroy()` calls.

The first three of the four methods in Listing 2 are typically part of a package. The `activate` method generates a modal UI element (`ModalPanel`) and pins it to Atom's workspace (line 4). The `subscriptions` method assigns the menu entry defined in Listing 3 to the `@toggle()` method (Listing 2, line 20), which is optional. It ensures that an entry for `hello-world` is displayed below the Packages menu. If you click on `Toggle` in this venue, or press Alt+Ctrl+O, you will see the window from Figure 2 either appear or disappear.

The details in the menu are customizable; Listing 3 shows the simple schema. It lives in the `menus` folder in the `hello-world.cson` file and creates an entry for the context menu (line 1) and for the Atom menu (line 8).

Moreover, developers who pay attention to style can change the style in a `.less` file (below `styles`), define a keyboard shortcut (`keymaps`), and create specifications for tests (`spec`). The specs also lets you perform regression tests [7]; the Atom makers rely on their


package.json - /home/firefox/github/hello-world - Atom

```
{
  "name": "hello-world",
  "main": "./lib/hello-world",
  "version": "0.0.0",
  "description": "A short description of your package",
  "activationCommands": {
    "atom-workspace": "hello-world:toggle"
  },
  "repository": "https://github.com/atom/hello-world",
  "license": "MIT",
  "engines": {
    "atom": ">=0.174.0 <2.0.0"
  },
  "dependencies": {
  }
}
```

Figure 4: **The** `package.json` **that resides in the root directory of the package contains metadata, such as the path to the executable file and the version number.**

LISTING 1: lib/hello-world-view.coffee

```
module.exports =
class HelloWorldView
  constructor: (serializedState) ->
    # Create root element
    @element = document.createElement('div')
    @element.classList.add('hello-world')

    # Create message element
    message = document.createElement('div')
    message.textContent = "Hello World!"
    message.classList.add('message')
    @element.appendChild(message)

  # Returns an object that can be retrieved when package is activated
  serialize: ->

  # Tear down any state and detach
  destroy: ->
    @element.remove()

  getElement: ->
    @element
```

own Jasmine [8] JavaScript test framework for this.

Conclusions

Given that it is only version 1.0, Atom already gives a robust impression, and it is genuinely easy to configure comprehensively. Users of the Sublime Text editor will probably be familiar with many elements; the developers seem to have borrowed heavily from the proprietary Python software. The developers are probably confident that the tie to GitHub will ensure them a new and sustainable basis for their code platform.

Even Atom has a learning curve. The editor still has some deficits in terms of user interface response time and in the way it handles large files. Additionally, the installation is a bit tricky. However, Atom has a very active user community after just one year and has been downloaded 750,000 times. The 803 developers have added 1,861 packages, so it looks like the editor meets a genuine demand. ■■■

LISTING 2: lib/hello-world.coffee

```
[...]
  activate: (state) ->
    @helloWorldView = new HelloWorldView (state.
      helloWorldViewState)
    @modalPanel = atom.workspace.addModalPanel(item: @
      helloWorldView.getElement(), visible: false)

    # Events subscribed to in atom's system can be easily
      cleaned up with a CompositeDisposable
    @subscriptions = new CompositeDisposable

    # Register command that toggles this view
    @subscriptions.add atom.commands.add 'atom-workspace',
      'hello-world:toggle': => @toggle()

  deactivate: ->
    @modalPanel.destroy()
    @subscriptions.dispose()
    @helloWorldView.destroy()

  serialize: ->
    helloWorldViewState: @helloWorldView.serialize()

  toggle: ->
    console.log 'HelloWorld was toggled!'

    if @modalPanel.isVisible()
      @modalPanel.hide()
    else
      @modalPanel.show()
```

LISTING 3: menus/hello-world.cson

```
'context-menu':
  'atom-text-editor': [
    {
      'label': 'Toggle hello-world'
      'command': 'hello-world:toggle'
    }
  ]
'menu': [
  {
    'label': 'Packages'
    'submenu': [
      'label': 'hello-world'
      'submenu': [
        {
          'label': 'Toggle'
          'command': 'hello-world:toggle'
        }
      ]
    ]
  }
]
```

INFO

[1] Atom: *https://atom.io*
[2] Sublime Text: *http://www.sublimetext.com*
[3] Electron: *http://electron.atom.io*
[4] NodeSource distribution of Node.js: *https://github.com/nodesource/distributions*
[5] Grunt: *http://gruntjs.com*
[6] Atom Flight Manual: *https://atom.io/docs/latest/*
[7] Writing specs for Atom: *https://atom.io/docs/latest/hacking-atom-writing-specs*
[8] Jasmine: *http://jasmine.github.io*