ADMIN

Network & Security

In Partnership with TuxCare

10 Terrific Tools

FOR THE BUSY ADMIN

2021 EDITION

Discover a free tool to help you:

- Enforce stronger passwords
- Manage multiple databases with a single app
- Add QR stickers to your hardware for easy inventory

And much more!

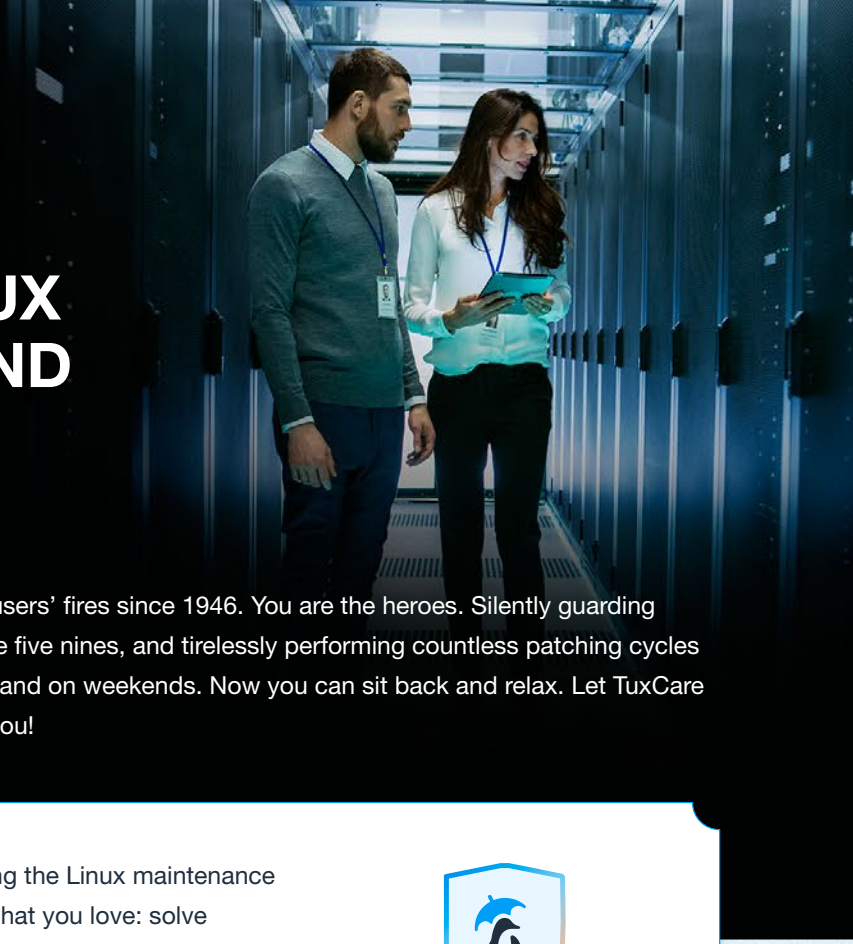
Bonus articles

- Two-Factor Authentication
- Ugrep

WWW.ADMIN-MAGAZINE.COM US$ 7.95

TuxCare

AUTOMATE LINUX AVAILABILITY AND SECURITY



HAPPY SYSADMIN DAY!

As a SysAdmin, you have been putting out users' fires since 1946. You are the heroes. Silently guarding companies of all sizes, keeping watch on the five nines, and tirelessly performing countless patching cycles all year round, sometimes through the night and on weekends. Now you can sit back and relax. Let TuxCare take care of Linux support and security for you!

We are TuxCare, and we are automating the Linux maintenance routine so you have more time to do what you love: solve complex riddles, learn about new technologies and maybe even build some LEGO!

TuxCare Live patching services eliminate maintenance windows with automated updates for Linux kernels, shared libraries, VMs, databases, and embedded devices used in the office. No more Friday night security updates!

TuxCare Linux Support Services maintain and prolong the life of your current or end-of-life Linux distributions, giving you time to migrate to the newest version. Unload your sysadmin workload and keep Linux secure without all the fuss.

Learn more

tuxcare.com

AlmaLinux redhat ORACLE LINUX ubuntu amazon webservices Linux

CentOS CloudLinux debian PROXMOX

10 Terrific Tools

FOR THE BUSY ADMIN 2021 EDITION

Dear Readers,

Experts know that every task has a tool. Commands like `ping` and `top` are well known to system admins, but the experts have many other powerful utilities for managing the daily challenge of keeping systems up and ready. We round up another list of little-known gems in this year's lineup of *Terrific Tools for the Busy Admin*.

ADMIN Special

Editor in Chief – Joe Casad

Copy Editors – Amy Pettle, Aubrey Vaughn

Layout / Graphic Design – Dena Friesen, Lori White

Advertising
Brian Osborn, bosborn@linuxnewmedia.com
Phone: +49 8093 7679420

Publisher – Brian Osborn

Customer Service / Subscription
For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(toll-free from the US and Canada)

www.admin-magazine.com

Printed in Nuremberg, Germany by hofmann infocom GmbH.

Distributed by Seymour Distribution Ltd, United Kingdom

ADMIN is published by Linux New Media USA, LLC, 4840 Bob Billings Parkway, Ste 104, Lawrence, KS 66049, USA.

Published in Europe by: Sparkhaus Media GmbH, Bialasstr. 1a, 85625 Glonn, Germany

Table of Contents

As a special bonus, we're also including two more articles describing other great tools for the busy admin's toolkit:

Strong Passwords

Regular password changes are a thing of the past: Strong passwords for each individual service provide more protection. Charly pimped his Ubuntu accordingly with a suitable PAM module. By Charly Kühnast

Changing the password regularly, about every 60 or 90 days, is now considered obsolete. It is better to use a separate strong password for each service and each login. The requirement for how strong (i.e., how complicated) a password must be is something that – at least on your own systems – you can define yourself.

On my test machine with Ubuntu, I can use almost any simple password I want – that has to change. To make sure it does, I first have to install the pwquality PAM library:

```
$ sudo apt install libpam-pwquality
```

Then I have to add a line to the `/etc/pam.d/common-password` configuration file. On Ubuntu 18.04 "Bionic Beaver," the default looks like this (this may be slightly different on other systems):

```
password [success=1 default=ignore] ➥
  pam_unix.so obscure sha512
```

This line can remain as a fallback, but in front of it – and this is important – I need to insert the line from Listing 1. This is a single line, which I just wrapped for Listing 1 to improve readability. With the individual parameters (Table 1 breaks them down), the password requirements can be easily controlled.

Listing 1: Password Requirements

```
password requisite pam_pwquality.so \
retry=4 minlen=9 difok=4 lcredit=-2 \
ucredit=-2 dcredit=-1 ocredit=-1 \
reject_username enforce_for_root
```

```
bob@influx:~ $ passwd
Changing password for bob.
Current password:
New password:
BAD PASSWORD: The password is shorter than 8 characters
New password:
BAD PASSWORD: The password fails the dictionary check -
is based on a dictionary word
New password:
Retype new password:
passwd: password updated successfully
```

Figure 1: After the change, the system rejects overly simple passwords.

After restarting the system, the new password rule takes effect. To test it, I changed the password of the user *bob* (Figure 1). In doing so, I intentionally entered a password that was too short in the first round and one that can be found in common dictionaries in the second. The system categorically rejected both – and that's the way it should be.

As my third attempt, I entered a new password that complied with the modified rules: `Cm1.Sya-n`. This seems complicated, but it is mnemonic. It's the first letters and punctuation of the first words of Melville's *Moby Dick* [1], with a *1* instead of an *I*, because I need a digit according to the new password rule. The system accepted the password without complaint. ■

Info

[1] "Call me Ishmael. Some years ago – never mind how long precisely ...": [http://www.online-literature.com/melville/mobydick/2/]

Author

Charly Kühnast manages Unix systems in a data center in the Lower Rhine region of Germany. His responsibilities include ensuring the security and availability of firewalls and the DMZ.

Table 1: pwquality Parameters

Parameter	Meaning
retry	Number of incorrect attempts
minlen	Minimum password length
difok	Number of characters that can match the old password
lcredit	Minimum number of lowercase letters
ucredit	Minimum number of uppercase letters
dcredit	Minimum number of numbers
ocredit	Minimum number of non-standard characters
reject_username	Password and username cannot be identical
enforce_for_root	Rules also apply for root

Lean Bookkeeper

Tools that measure the network throughput on an interface and provide a history are not easy to find. VnStat manages this balancing act and finds favor with Charly. By Charly Kühnast

There are many small tools that measure and display the network throughput on an interface, and I have already introduced some of them here. If you also want a history of the data traffic volume, you have many choices, but then these tools are often not exactly lightweight. There is one, though, that manages the balancing act: VnStat [1].

After installing vnStat on my test computer, which runs Ubuntu, vnStat automatically created a database for each interface found (Listing 1, lines 1 to 4). If the tool does not do this automatically, the databases can be created manually (line 6). The `--delete` parameter is used to delete a database if necessary.

The `-i` parameter is used to track the load on an interface (line 8) in live (`-l`) operation. If you interrupt the output by pressing Ctrl + C, an easily understandable overview of the measured values appears (Figure 1). When displaying the network throughput, I prefer to read the values in bits per second rather than in bytes. You can do this by appending the `-ru` 1 parameter to the command (`ru` stands for "rate unit").

The databases mentioned at the beginning are used to provide the desired history about the network load. The `-h`, `-d`, `-w`, and `-m` parameters help me to display the data traffic history for an hour, a day, a week, or a month.

The `-t` parameter stands for "Top 10" and returns the 10 days with the most traffic. Another handy parameter is `--oneline`, which gives you minimal output that can be parsed easily for rehashing in your own scripts, for example.

Finally, vnStat has a colorful counterpart named vnStati, which illustrates the results in easily understandable diagrams. I have never looked at it before, because the monochrome version has always been fine for my needs. ■

```
enp2s0  /  traffic statistics

                           rx         |       tx
--------------------------------------+------------------
  bytes                    10.92 MiB  |        6.30 MiB
--------------------------------------+------------------
          max          8.65 Mbit/s    |     9.48 Mbit/s
      average        738.65 kbit/s    |   425.99 kbit/s
          min            1 kbit/s     |        0 kbit/s
--------------------------------------+------------------
  packets                      12828  |             9969
--------------------------------------+------------------
          max              740 p/s    |         786 p/s
      average              103 p/s    |          80 p/s
          min                1 p/s    |           0 p/s
--------------------------------------+------------------
  time                 2.07 minutes
```

Figure 1: VnStat lets you monitor the throughput on a network interface in real time. Exiting the tool by pressing Ctrl+C displays a practical summary of the measurement results.

Info

[1] vnStat: [https://humdi.net/vnstat/]

Listing 1: Monitoring an Interface

```
01 charly@glas:~$ ls -l /var/lib/vnstat
02 total 8
03 -rw-r--r-- 1 vnstat vnstat 2272 Jun  8 14:30 enp1s0
04 -rw-r--r-- 1 vnstat vnstat 2272 Jun  8 14:30 enp2s0
05
06 charly@glas:~$ vnstat --create -i eth0
07
08 charly@glas:~$ vnstat -i enp1s0 -l
09 Monitoring enp2s0...    (press CTRL-C to stop)
10
11    rx:       204 kbit/s    351 p/s
12    tx:        34 kbit/s     39 p/s
```

Measure system runtime with tuptime Stopwatch

How long has the Linux server been running without rebooting? And how often has the system rebooted without you noticing? These questions and more are answered by the tuptime tool. By Tim Schürmann

If a Linux system has been running for a long time, this is definitely proof of its stability, but – depending on the distribution – some updates might be waiting to install. Conversely, if the system reboots very frequently, there may be a configuration error – or maybe a hardware component is slowly deteriorating. Such restarts are quickly noticed on a workstation computer, but not necessarily on a remote server that is running quietly and well away from the action. How long a system has been running continuously can be determined at the command line by a call to `uptime`. But you might also want to try tuptime [1], a similar tool whose name is based on a contraction of "total uptime." It outputs far more information, including valuable information in the form of the number of reboots and the kernel version used.

Installation

While `uptime` is available on almost every system, you need to install tuptime separately. Some distributions, such as Debian and Ubuntu, already have the tool in their repositories. Arch Linux users will find tuptime in the AUR and CentOS users in the EPEL repository. On any Linux system, you can install the tool with the following:

```
$ curl -Ls https://git.io/tuptime-install.sh | sudo bash
```

Tuptime itself consists of a Python script and requires Python 3 with the modules `sys`, `os`, `optparse`, `sqlite3`, `locale`, `platform`, `datetime`, and `logging`. These modules should be preinstalled on most distributions. For more information about installation, see the box "In the Background."

In the Background

The system calls `tuptime` briefly during the boot and shutdown processes. In each case, the tool notes the system time. It is from the timestamps collected in this way that tuptime ultimately calculates the total runtime and all other values.

To make tuptime start automatically at boot and shutdown time, `tuptime-install.sh` sets up some startup scripts. For example, it sets up a corresponding service unit on a system with systemd. Besides systemd, the installation script also supports SysVinit and OpenRC. When using the installation script, tuptime also always runs under its own user account, named `_tuptime`.

If at some point the power should suddenly fail, tuptime would not notice anything. As a consequence, the timestamp for the shutdown would be missing, and this would prevent the tool from calculating the correct total runtime. For this reason, the installation script additionally sets up a cron job, which in turn ensures that tuptime is launched at regular intervals. All timestamps noted by tuptime end up in a small SQLite database, which is located below `/var/lib/tuptime/tuptime.db` by default.

Time Measurement

After setting up the program, query the current status via `tuptime` in the terminal. The tool already provides some basic information (Figure 1): At the very top it shows the number of system starts (at System Startups), as well as the time and date since the software started counting the starts. Tuptime doesn't capture events before this date. To get a complete picture, set up the program as soon as possible after system installation.

Next, tuptime tells you how often the system has been shut down (System Shutdowns). Pay special attention to the number of uncontrolled shutdowns labeled *bad*. If this figure increases rapidly, there is a major problem, such as an unstable power supply. System life indicates how long tuptime has been monitoring the system. If you installed tuptime directly after the installation of the distribution, this value corresponds to the time since the first boot.

This is followed by the total system uptime (System Uptime) and the total time the computer was powered off (System Downtime). If the percentage of downtime exceeds 50 percent, the computer was off longer than in operation. How long the system was in operation on average is indicated by the value following Average Uptime. Similarly, Average Downtime indicates how long the computer was switched

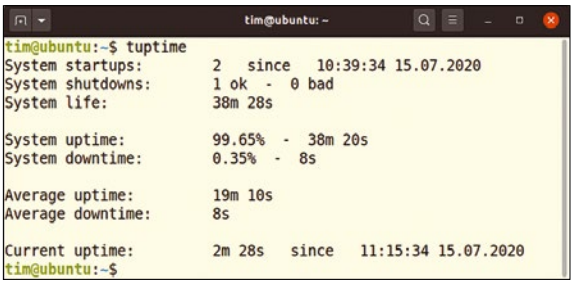


Figure 1: Shortly after the install tuptime cannot give you very much information, but over time the data starts to give you a better impression of the computer's behavior.

off on average. The final value listed is the time since the last system start (Current Uptime).

Display Options

The `tuptime -l` command returns a list with all startup operations. In the list, you can see in detail when the system was in operation and for how long. If the list seems too confusing, `tuptime -t` converts the data to tabular form, shown in Figure 2. The additional `-r` parameter reverses the order; the last system start appears at the top.
If you still feel overwhelmed by the volume of information, the display can be limited to a certain period of time. For example, if you want to know when and for how long the system was active between July 27 and July 28, 2020, just call `tuptime` with the command:

```
$ tuptime --tsince=$(date
--date="2020-07-27" +%s) --tuntil=$(date
--date="2020-07-28" +%s)
```

The `--tsince` and `--tuntil` parameters each expect a timestamp that is calculated by date. Tuptime also outputs these timestamps with the `-s` parameter. Alternatively, you can limit the output to very specific startup operations. For example, if you are interested in how long the system ran from the very first to the fifth logged startup, use the following:

```
$ tuptime --since 1 --until 5
```

More Details

If you append the `-k` parameter to the call, tuptime will tell you the kernel version used at each startup; if you add the `-b` option, it will also provide the unique identification numbers of the individual boot processes (Figure 3). The parameter `-p` tells tuptime to show you how long the system slept in each case (after Sleeping). Similarly, the active time is noted after Running. These values are only available on systems that have Python 3.6 or higher in place.

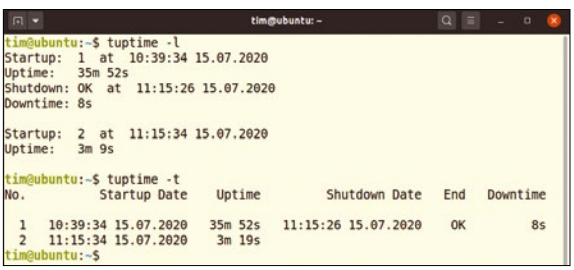


Figure 2: Tuptime offers different display formats. Here, there has been a reboot since the tool was installed, so two system starts have been counted.

A Question of Format

If so desired, `tuptime -csv` will deliver all the information in CSV format. If you append a `-t`, you are given a table that you can redirect to a file and then open with a spreadsheet program.
If you don't like the date and time format in the output, you can change it using

```
$ tuptime -d '%H:%M:%S %m-%d-%Y'
```

Tuptime replaces the placeholders beginning with a `%` with the corresponding values. `%H` stands for the hours, `%M` for the minutes, `%S` for the seconds. Similarly, `%m` returns the month as a number, `%d` the day, and `%Y` the year. Details of other placeholders are explained in the tuptime documentation, which you will find on GitHub in the `tuptime-manual.txt` file [1]. ■

Info

[1] tuptime: [https://github.com/rfrail3/tuptime]

The Author

Tim Schürmann is a freelance computer scientist and author. Besides books, Tim has published various articles in magazines and on websites.

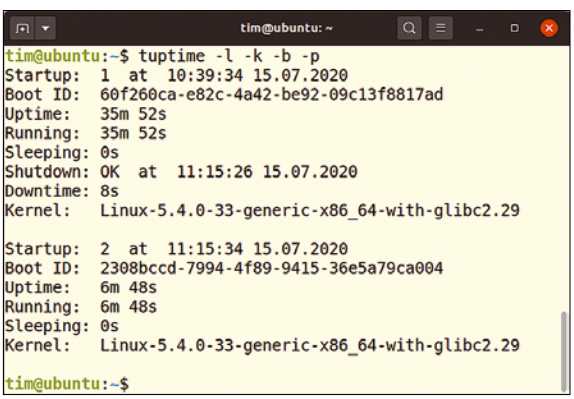


Figure 3: This system used the same Linux kernel for both system starts. After the first start, it ran for 35 minutes without going to sleep.

Sorting the Harvest

In order to keep up to date with security, Charly uses RSS feeds, among other things. He lets rss2email send the most important feeds directly to his mailbox to ensure that nothing is overlooked. By Charly Kühnast

RSS feeds are still an essential part of my daily information refueling plan. I sort my feeds in Miniflux, a web-based RSS aggregator, which I wrote about in this column back in 2014 [1]. I am happy to say that Miniflux is still being actively developed.

I have a very small number of RSS feeds emailed to me directly. These are feeds that warn me of acute security problems. By having them delivered directly to my inbox, I can avoid the risk of missing an important article – Miniflux can handle between 200 and 250 feeds, most of which I just skim.

I use rss2email – a very logical choice of name – to email me the feeds. It can be found in the package sources of almost all the popular distributions. On Ubuntu, for example, I installed it with the following call:

```
$ sudo apt install rss2email
```

Listing 1: rss2email.cfg

```
[DEFAULT]
from = rss@mydomain.com
force-from = True
html-mail = False
to = charly@mydomain.com
```

Before using rss2email for the first time, you have to create a configuration file; the approach is very clear-cut. If there is no folder named `.config/` in your home directory, you have to create one and `cd` to it:

```
$ cd
$ mkdir .config
$ cd .config/
```

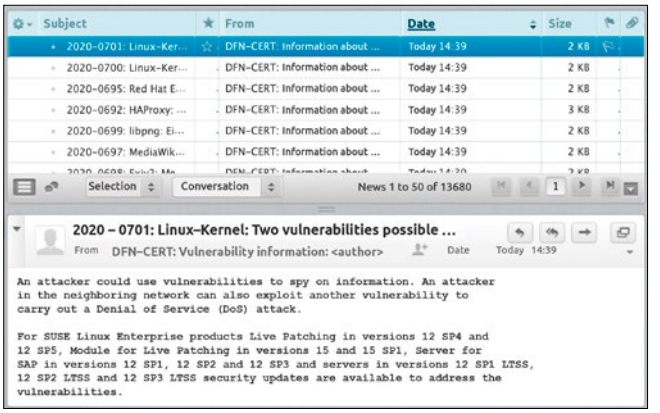


Figure 1: Charly uses rss2email to deliver information from RSS feeds about acute vulnerabilities directly to his mailbox.

When you get there, use the editor of your choice to create the `rss2email.cfg` configuration file with the content from Listing 1; the `force-from = True` entry is of special importance here. Normally, rss2email would use a sender address from the RSS feed if it found one there. However, on many mail systems, this leads to the spam filter rightly becoming suspicious. The `force-from = True` entry now causes rss2email to always use the address stored in the `from` line. Whether you set `html-mail` to `True` or `False` is a matter of taste.

Now it's time to choose one or more RSS feeds that you want rss2email to check out regularly. The syntax for this is:

```
r2e add <title> <URL>
```

For the DFN Certificate RSS feed, which informs admins of current security problems, look at the first line in Listing 2. Whether the `add` action worked can be verified using `r2e list` (line 2). It worked in my example.

The `r2e run` command traverses the feeds and dispatches the emails. Now all I need to do is write a crontab entry to run the command at specified intervals (e.g., every 30 minutes). To do so, I type `crontab -e` and add the crontab line from Listing 3.

Sure enough, the first messages start to slowly arrive in my mailbox (Figure 1). The important thing here is to be careful about the rss2email feeds you choose; otherwise you might flood your inbox – just staying with Miniflux would be preferable to that. ■

Listing 2: Enter and check RSS feed

```
01 $ r2e add DFNCert https://adv-archiv.dfn-cert.de/rss/latest
02 $ r2e list
03 0: [*] DFNCERT (https://adv-archiv.dfn-cert.de/rss/latest
   -> charly@mydomain.com)
```

Listing 3: Cron job for rss2email

```
*/30 * * * * /usr/bin/r2e run > /dev/null 2>&1
```

Info

[1] "The sys admin's daily grind: Miniflux" by Charly Kühnast, *Linux Magazine*, issue 164, July 2014, p. 50, [https://www.linux-magazine.com/Issues/2014/164/Charly-s-Column-Miniflux/(language)/eng-US]

Best Laid Plans

The dstask personal tracker lets you manage your to-do list from the command line. By Tim Schürmann

The dstask personal tracker can help you prioritize tasks and track completion. With a short, succinct command, you can add a new task or mark off a completed one. On request, dstask provides a list of all pending tasks, sorted by urgency. Filters help you stay on track. As an added benefit, you can use your task list to show customers or your boss your completed work. Under the hood, dstask stores the pending tasks in the Git version management system, letting you sync tasks across all your devices.

Kickstart

To get dstask up and running, first install Git using your package manager. Then go to the dstask project page on GitHub [1] and click on the current version number. Under *Assets*, download the version for Linux, `dstask-darwin-amd64`. Rename the resulting program `dstask` and make it executable. You do not have to actually install dstask, but the developer does recommend storing dstask in `/usr/local/bin`, which lets all of the system's users call the program directly by typing `dstask`.
After that, you still have to create the Git repository where dstask stores all the tasks and becomes acquainted with Git (Listing 1).

Come On In

To add a task to dstask, use the call shown in line 1 of Listing 2. The `add` command tells the program to create a new task, which is described in the following brief summary (`clean up basement`). Dstask adds all words that follow a plus sign (+) to the task as keywords (i.e., `basement` and `private` in Listing 2). These tags will help you later on when searching for a specific task. You can also add the tags to the summary, as shown in line 2 of Listing 2. And you can set the task priority, shown by the number following `P`. Dstask supports priority levels `P3` to `P0`, with `P0` for urgent tasks and `P2` for normal priority tasks. In Listing 2, I've set my basement cleanup task to `P2` because it's not time critical. Dstask automatically defaults to `P2` if you don't include the priority level.

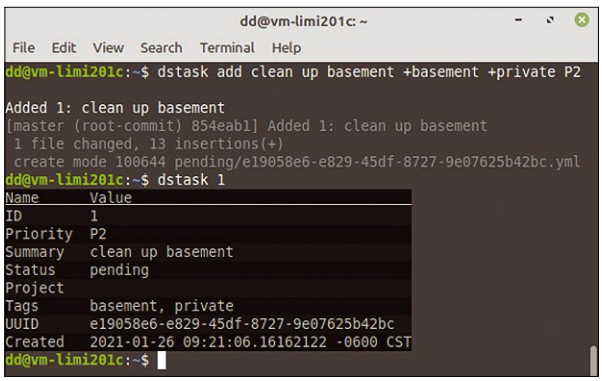


Figure 1: Dstask removes the tags from the task name. To see an individual task's detailed information (including tags), enter `dstask` followed by the task number (1). (You can ignore the messages in light gray from Git.)

Roll Up Your Sleeves

To see what dstask has on your to-do list, just call `dstask`. The program assigns a consecutive ID to each task. Entering `dstask 1` shows detailed information about the task (Figure 1). Dstask ignores case sensitivity in the tags.
The detailed information also shows the task `status`. Immediately after creation, the task will be shown as `pending`. When you head down to the basement and pick up the first box, call

```
dstask start clean up basement
```

to change the status. This switches the task to the `active` state, signifying that the task is in progress. If you take a break, stop processing with

```
dstask stop clean up basement
```

which toggles the task back to the `pending` state.
Once the basement is finally clean, mark the task as done by entering

```
dstask done clean up basement
```

Even though dstask is done reminding you about this task, it

Listing 1: Creating a Git Repository

```
$ mkdir ~/.dstask && git -C ~/.dstask init
$ git config --global user.email "editorial@linux-user.en"
$ git config --global user.name "Tim Schürmann"
```

Listing 2: Adding Tasks

```
01 $ dstask add clean up basement +basement +private P2
02 $ dstask add +private basement +clean up basement P2
```

Listing 3: Organizing Tasks in Projects

```
01 $ dstask add clean up garage +private P3 project:spring cleaning
02 $ dstask add declutter shoe closet +Private P1 project:Spring Cleaning
03 $ dstask modify 1 -Private +Books P1 project:Spring cleaning
```

Listing 4: Templates

```
01 $ dstask template clean up desk +Private P3 project:administration
02 $ dstask add template:4 sort pencils
```

will still store the task, thereby creating a record of your completed tasks for your boss or client. (See Table 1 for more "Display Formats and Commands.") To get a report on all completed tasks, enter

```
dstask show-resolved
```

Dstask automatically sorts the tasks by weeks. If you want to remove a task from the dstask repository, use the following command:

```
dstask remove clean up basement
```

Project Work

While you're cleaning up the basement, you might as well tackle the garage and declutter the shoe closet. All three cleaning tasks could be grouped together to create a `spring cleaning` project. Dstask lets you add individual tasks to a project by adding another parameter when you create a task. The tasks in Listing 3 are part of the `spring cleaning` project specified after the `project:` parameter. (Note again that dstask is not case sensitive.) As shown in Figure 2, a call to `dstask` reveals that the original task `clean up basement` does not yet belong to any project. To change this, you can quickly modify the information by typing

```
dstask modify 1 project:Spring cleaning
```

The number again stands for the corresponding task ID (where 1 is the basement cleanup task). You can use `modify` to adjust the priority, keywords, and the project itself; a minus sign removes the tag in question (Listing 3, line 3).

Targeted Intervention

Use the `dstask edit 1` command to make a correction in a note. Dstask then opens all the details of the specified task in a text editor (Figure 3). As shown in Figure 3, modify the project name to the right of the `project:` parameter. When you are done, save your adjustments and exit the editor (in nano, press Ctrl + O followed by Ctrl + X).

Templates

You probably need to clean up your desk far more often than your basement. For recurring tasks, templates can save you some typing (Figure 4). Templates are initially conventional tasks, but you create them using the `template` keyword (Listing 4, line 1). ■

Info

[1] dstask: [https://github.com/naggie/dstask]

Table 1: Display Formats and Commands

Action	Function
next	List of major tasks
show-projects	List of all projects including the resolved tasks
show-tags	List of all assigned tags
show-active	List of all started tasks
show-paused	List of all tasks that were started and then paused
show-open	List of all tasks not yet finished
show-resolved	List of all completed tasks
show-templates	List of all templates
show-unorganised	List of all tasks that do not have a tag or are not assigned to a project

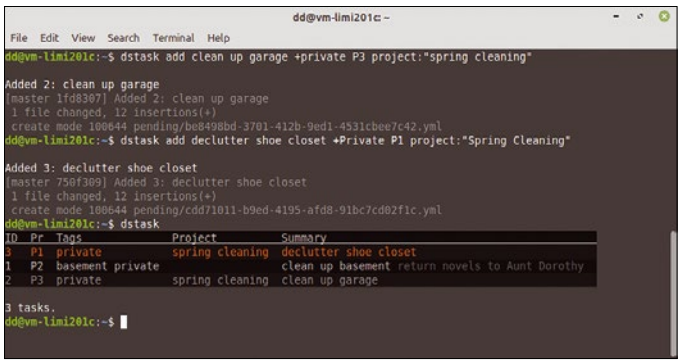


Figure 2: When you call `dstask`, you see a list of pending tasks. Dstask displays any existing notes in light gray in the `Summary` column.

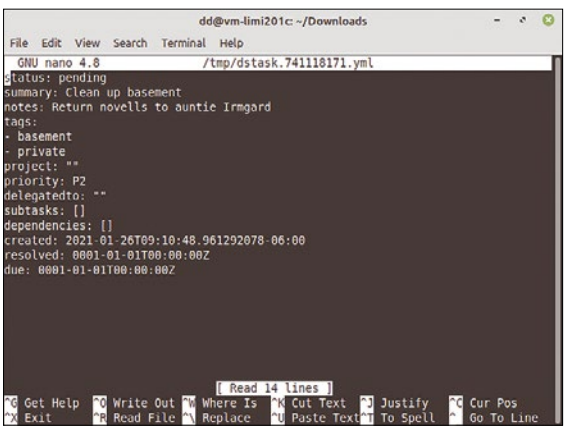


Figure 3: The editor shows you how dstask stores the task under the hood: a text file in YAML format.

Eureka!

Fd is an uncomplicated find replacement that discovers lost treasures in the filesystem in next to no time. Charly would love to deploy an amazing tool like this in the analog world of his office. By Charly Kühnast

I'm not very good at sorting things sensibly and then finding them again – both in my office and on my computers' filesystems. For the latter, at least I have electronic help in the form of tools like `find` and, more recently, `fd`. The `find` command existed on Unix systems long before Linux was invented – in fact, it's older than most of the people who use it. On many of my systems, there is a directory named `/test` where I try things out. Anything that proves useful is sent to Git; the rest just hangs around gathering dust until the cron job in Listing 1 sweeps it away without write access after 365 days.

While doffing a hat to the now impressive power of the GNU implementation of `find` [1], you still sometimes find yourself wishing for a tool that can perhaps do a little less, but one that is more intuitive to use. This is where `fd` [2] jumps into the breach. The compact younger sibling of `find`, `fd` has already made its way into many distributions, but often only recently. In Ubuntu, it is available starting with version 19.04, for example. After installing `fd` on my test Ubuntu, I now have an `fdfind` command. But the developers make it quite clear that their tool is named `fd` and use this name in all the examples. In order to permanently teach my system the short form, I just added an `alias fd=find` entry to my `.bashrc`.

Quickly perusing the man page reveals that `fd` can definitely do less than `find`, but it does what it does well, intuitively, and quickly. Typing `fd` without any further parameters returns the current directory's contents including all its subfolders, but without the hidden files and directories – like `ls`, but recursively. If the environment variable `LS_COLOR` is set (which is the default on most systems), the output will be in color.

Things become more interesting if you are searching for a file name or name component.

```
$ fdfind rng
run/rngd.pid
dev/hwrng
usr/sbin/rngd
usr/bin/rngtest
etc/rc4.d/S01rng-tools
etc/rc6.d/K01rng-tools
etc/default/rng-tools
etc/rc0.d/K01rng-tools
etc/init.d/rng-tools
etc/rc5.d/S01rng-tools
etc/rc1.d/K01rng-tools
etc/rc3.d/S01rng-tools
etc/rc2.d/S01rng-tools
sys/module/rng_core
usr/share/doc/rng-tools
usr/include/linux/virtio_rng.h
run/systemd/generator.late/rng-tools.service
run/systemd/units/invocation:rng-tools.service
etc/logcheck/ignore.d.server/rng-tools
etc/logcheck/violations.ignore.d/rng-tools
usr/lib/python2.7/lib2to3/PatternGrammar.txt
```

Figure 1: Without concrete instructions, `fd` ignores the case, but it can even handle regular expressions if necessary.

Listing 1: Crontab Entry

```
find /test/* -mtime +365 -exec rm {} \;
```

In Figure 1, I told `fd` to search the root directory `\` for `rng`. As you can see, it also found `PatternGrammar.txt` (at the very bottom). This is because `fd` is not case-sensitive by default. However, if an uppercase letter is stipulated as the search term, it switches its behavior and only returns case-specific results.

You can search for file extensions with the `-e` parameter. For example, to find all PNG images in and below the current directory, just type:

```
fd -e png
```

I use regular expressions for fine tuning. By way of an example, the command

```
fd '^a.*png$'
```

finds file names that start with `a` and end with `png`. The GitHub page [2] for the tool explains many more applications and parameters.

Now all I really need is a physical `fd` counterpart to tidy up my office … ■

Info

[1] GNU find: [https://www.gnu.org/software/findutils/manual/html_mono/find.html]

[2] fd: [https://github.com/sharkdp/fd]

Shell Screencasts

Asciinema lets you record events at the command line and publish the resulting terminal movie on the web. By Christoph Langner

A screencast (i.e., a movie of what is happening on screen) helps developers demonstrate their programs to users and is useful for people seeking a way of explaining their problems to a support specialist. On Linux, there are many different solutions for this, such as recordMyDesktop and OBS Studio, or – as in the case of Gnome – the feature is integrated into the desktop environment. But if you only want to record shell commands and their output, you're using a sledgehammer to crack a nut. Asciinema [1] can be a good, lean alternative for these cases.

Asciinema consists of three components. The first is the actual recording tool for the command line. The second is a web-based hosting platform for asciinema videos, which is similar to YouTube or image hosts such as Imgur.com or Gfycat.com. The third component is a JavaScript player that plays the asciinema videos [2]. You only need the recorder, unless you want to host your asciinema videos on the web yourself. In that case, you would have to set up the server components on a web server.

And ... Action

Most current distributions include the screencast recorder for asciinema in their package sources. The application version counter is currently at 2.0.2. Ubuntu 18.04, Debian 10 "buster" (`sudo apt install asciinema`), and Fedora 28 (`sudo dnf install asciinema`) at least give you asciinema 2.0.0. More information about the installation, such as for the Python package manager Pip, can be found in the application documentation [3].

After the install, the easiest way to start recording is to type the `asciinema rec` command. Doing so opens a new shell in which asciinema records everything you type, and the system displays it on the screen. It continues until you stop recording by typing `exit` or pressing Ctrl + D. You can then either press Enter to upload the video to *asciinema.org*, or save it locally in the `/tmp/` directory using Ctrl + C. The file name is always `tmp<random code>-ascii.cast`. Alternatively, you can pass a file name and path directly to asciinema at startup:

```
$ asciinema rec <example>.cast
```

To play the locally saved recording, call asciinema again, this time with the `play` option and the movie file as a parameter. A typical call looks something like this:

```
$ asciinema play /<path>/<to>/ <example>.cast
```

Asciinema plays the screencast directly in the shell, but without executing the recorded commands locally – it is in fact a video, not a script. If necessary, you can use the space bar to pause playback or press "." to skip through the video frame. Press Ctrl + C if you want to stop the playback of the terminal movie completely.

When you're recording, keep in mind that asciinema records the events on the command line unchanged. It not only stores the commands and their output, but if you correct input at the command line, this is also shown in the asciinema video. Similarly, if you do nothing during the recording (e.g., because you need to look up the details of a command) the video pauses.

To avoid what are often involuntary and (for the viewer) boring breaks, add the `--idle-time-limit=<seconds>` option or the shortcut `-i<seconds>` to the call to record the asciinema video. This limits the timeout to the number of seconds specified in the option. For example, `asciinema rec -i1` gives you a maximum timeout of one second.

Managing Shell Videos

Alternatively, you can upload the terminal video directly to *asciinema.org* on the web without any detours. You don't have to register with the service to do this; the video is uploaded automatically if you do not press Ctrl + C to cancel the action at the end of the recording.

Asciinema's terminal recorder then displays a URL in the style of `https://asciinema.org/a/jl0[...]2wN`, where you (or others) can play back the movie in a web browser (Figure 1). The link remains active for seven days, after which the system automatically

archives the recording, and access to the movie is lost.

If you want to keep the movie accessible for a longer period of time, you have to authenticate against *asciinema.org* on each computer where you want to create and upload recordings. To do this, enter the `asciinema auth` command and then open the link displayed in the terminal in a browser. You are then given the option to create an account on the asciinema website or to log in with an existing account. Asciinema will then automatically assign the terminal videos previously uploaded from this computer to your account. The web portal lets you manage your recordings. For instance, you can change the metadata (such as the title) or insert a description. The terminal theme and the preview image can also be modified. Settings | Make public lets you publish your video on a list [4] accessible to all users. With the default setting, however, your uploads remain private. Unsuccessful movies, or ones you no longer want, can be deleted from the overview.

Embedding asciinema

The Share button gives you information on how to integrate the selected terminal video into any website, such as your own WordPress blog or the GitHub page of one of your projects. The page automatically generates HTML or Markdown tags that display an image in the web page linked to the video on *asciinema.org*. You will also find a short script snippet below, which can be used to embed the video directly into a website (if the content management of the site allows it).

Listing 1 demonstrates an asciinema-enriched website. The first section uses the script; the second uses the universal image. Figure 2 shows the resulting and still very rudimentary website in the browser. The asciinema movie played back in the web browser supports copy and paste actions at any point, so that the viewers can, for example, copy commands from the video directly into the terminal. However, both variants have the disadvantage of embedding external resources into your own website (the script and the data loaded by asciinema or the embedded image). To avoid external data connections, download the image of the terminal movie, upload it locally to your website, and then link only to the asciinema video. ■

Listing 1: asciinema-Enriched Website

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1>Included as script</h1>
    <script id="asciicast-jl0NXr88Lm7c7DGOIf8Jh92wN" src="https://asciinema.org/
     a/jl0NXr88Lm7c7DGOIf8Jh92wN.js" async></script>
    <h1>Included as image</h1>
    <a href="https://asciinema.org/a/jl0NXr88Lm7c7DGOIf8Jh92wN" target="_blank">
      <img src="https://asciinema.org/a/jl0NXr88Lm7c7DGOIf8Jh92wN.svg" /></a>
  </body>
</html>
```

Info

[1] asciinema: [https://asciinema.org]

[2] asciinema project on GitHub: [https://github.com/asciinema]

[3] Installation instructions: [https://asciinema.org/docs/installation]

[4] Public list of all asciicasts: [https://asciinema.org/explore/public]

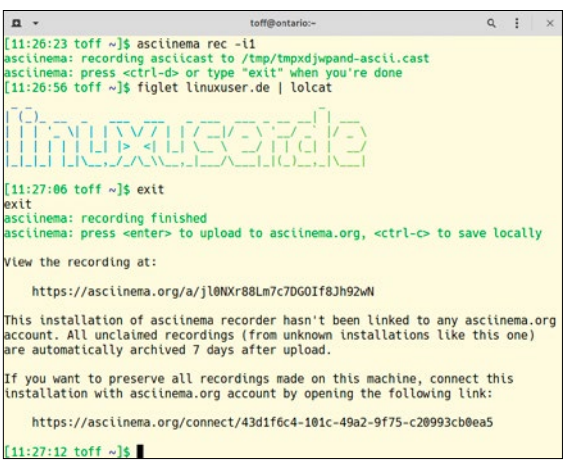


Figure 1: In the default configuration, asciinema automatically uploads the recorded movies to its in-house "YouTube alternative" for terminal movies.

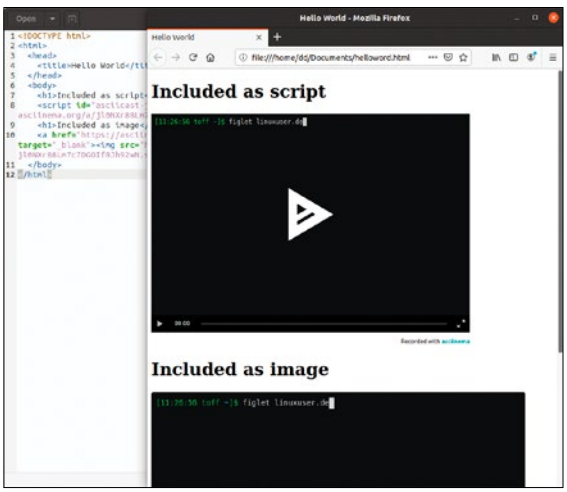


Figure 2: The terminal videos hosted by asciinema can be easily integrated into your own web pages, such as a blog or GitHub page.

Checklist

Doing a hardware inventory in a data center is anything but a piece of cake. In order to quickly assign devices to the appropriate database entry, Charly provides each newly acquired system with a QR code sticker with the help of Zint. By Charly Kühnast

When you need to manage large numbers of devices, there is no avoiding centralized data management. In the simplest case, this can be a wiki, with one entry per system. This will include, for example, the date of purchase, the length of the warranty period or maintenance contract, any repairs that have already been made, and the rack number where the device is installed (finding the hardware in a larger data center can be time-consuming). I then encode the URL of the wiki entry as a barcode or QR code, print it on self-adhesive film, and stick it on the device.

I generate the codes for this with Zint [1]. Many distributions have Zint on board; if not, it is quickly compiled from the GitHub repository. You must have libpng in place; otherwise, Zint will not generate images. Those who now want to generate codes are spoiled for choice: Zint knows dozens of variants (Figure 1). With `zint -t`, I can display their names. I know a few of these codes, like EAN and QR, from everyday life. PDF417 (Figure 2) and its relatives can be found on the boarding passes of many airlines. And there just happens to be a cold medicine bottle on the table in front of me that has a PZN barcode. I can see from a web page [2] for generating barcodes that this is used on pharmaceuticals in Germany. On the same website – funnily enough, it uses Zint itself – there are examples of the other types of code.

Listing 1: QR Code with Zint

```
$ zint -o ~/qr/linmagurl-qr.png -b 58 -d https://linux-magazine.com
```

Figure 2: PDF417 is the format often used in the transportation industry.

```
 1: Code 11            52: PZN                 96: DPD Code
 2: Standard 2of5      53: Pharma Two-Track    97: Micro QR Code
 3: Interleaved 2of5   55: PDF417              98: HIBC Code 128
 4: IATA 2of5          56: Compact PDF417      99: HIBC Code 39
 6: Data Logic         57: Maxicode           102: HIBC Data Matrix
 7: Industrial 2of5    58: QR Code            104: HIBC QR Code
 8: Code 39            60: Code 128-B         106: HIBC PDF417
 9: Extended Code 39   63: AP Standard Customer 108: HIBC MicroPDF417
13: EAN                66: AP Reply Paid      110: HIBC Codablock-F
14: EAN + Check        67: AP Routing         112: HIBC Aztec Code
16: GS1-128            68: AP Redirection     115: DotCode
18: Codabar            69: ISBN               116: Han Xin Code
20: Code 128           70: RM4SCC             121: RM Mailmark
21: Leitcode           71: Data Matrix        128: Aztec Runes
22: Identcode          72: EAN-14             129: Code 32
23: Code 16k           73: VIN                130: Comp EAN
24: Code 49            74: Codablock-F        131: Comp GS1-128
25: Code 93            75: NVE-18             132: Comp DataBar Omni
28: Flattermarken      76: Japanese Post      133: Comp DataBar Ltd
29: GS1 DataBar Omni   77: Korea Post         134: Comp DataBar Exp
30: GS1 DataBar Ltd    79: GS1 DataBar Stack  135: Comp UPC-A
31: GS1 DataBar Exp    80: GS1 DataBar Stack Omni 136: Comp UPC-E
32: Telepen Alpha      81: GS1 DataBar Exp Stack 137: Comp DataBar Stack
34: UPC-A              82: Planet             138: Comp DataBar Stack Omni
35: UPC-A + Check      84: MicroPDF           139: Comp DataBar Exp Stack
37: UPC-E              85: USPS Intelligent Mail 140: Channel Code
38: UPC-E + Check      86: UK Plessey         141: Code One
40: Postnet            87: Telepen Numeric    142: Grid Matrix
47: MSI Plessey        89: ITF-14             143: UPNQR
49: FIM                90: KIX Code           144: Ultracode
50: Logmars            92: Aztec Code         145: rMQR
51: Pharma One-Track   93: DAFT Code
```

Figure 1: Zint can generate these codes.

In Listing 1, I use `-b 58` to select QR as the code type. The parameter `-d` for `data` always has to be at the end: Zint blithely ignores all the options that follow. As long as I stick to this, the barcode generation routine works like clockwork, which gives me one less excuse to put off the pesky inventory process. ■

Info

[1] Zint: [https://github.com/zint/zint]
[2] Barcode generator: [http://www.barcode-generator.org]

Figure 3: The *Linux Magazine* URL as a QR code.

Databases

Usql is a useful tool that lets you manage many different databases from one prompt. By Marco Fioretti

Linux offers many tools for creating, populating, and querying databases. I will introduce you to usql, a little tool that is a lifesaver for many users who work with databases.

The most ubiquitous and flexible way to work with any database is in a text-based interface. Inside a client application, you type queries at a prompt. The syntax might vary depending on the implementation of Standard Query Language (SQL) [1] the database is using. Depending on the database type, the client either executes the query directly or, much more frequently, forwards it to a server that actually handles the data. The result of the query is then printed out, usually in a tabular format. Alternatively, you can store sequences of queries or commands in a text file and pass it to the client that will execute them automatically, possibly saving the result to a file.

If you always work with one type of database (for example, only SQLite), you can just choose a client for that specific database and get good at using it. However, if you frequently switch back and forth between different database clients, each with its own personality and feature set, it can get very confusing.

Usql [2] is a single database client that works with several different database systems. Although the syntax of the actual queries might vary slightly depending on the database, other commands for operating the client are unified in a convenient way that will simplify your database experience and lower the learning curve for adding new database systems to your repertoire. Usql is a clone of psql, the standard command-line client for PostgreSQL databases. The goal of usql is to "support all standard psql commands and features" and to extend those commands to include other database systems.

If you try to access 10 different databases with usql, you will still need to know all the SQL dialects and how they differ from each other. With usql, however, you will be able to query all those databases in the same session of the same terminal, and you'll have access to some extra features built into usql, such as syntax highlighting.

Installing usql

Usql is written in the Go language. The easiest way to use it on Linux is to download the tar archive for amd64 systems from the release page [3] (version 0.7.8 at time of writing). Uncompress the tarball and place the resulting binary file, unsurprisingly called usql, in a directory of your path. At this point, you should be able to type `usql` at a command prompt and start using the program. If your Linux system has a version of a library that is older than what usql expects, it won't run. Of course, many Linux applications fail to execute if the libraries are out of sync. The problem with the usql binary from the website is that the dependencies are not documented.

Another option for obtaining usql is to build it yourself in Go.

Getting Started

Usql provides two sets of commands: a big family of "normal" commands, plus the internal "meta" commands of usql itself. The normal commands are nothing more than standard SQL queries that you would use in other clients to insert, edit, or fetch data. I won't describe the syntax of those queries in this article because this is not a general introduction to SQL, and the commands vary depending on the database. (You will find many good SQL tutorials online.)

The usql meta commands are all prefixed by a backslash, and the first two commands to learn are the ones that tell you what is available in your usql installation. The `\drivers` command first lists all the database drivers that were compiled into your copy of usql. The backslashed question mark `\?` lists the available meta commands if typed without arguments. Add the name of a command to learn what it does, or use the `options` and `variable` keywords to view available options and variables.

Configuration

It is possible to define the general configuration and start-up behavior of usql by writing the meta commands in the `$HOME/.usqlrc` file.

Usql will also execute all the commands contained in a file passed to it with the `-f` or `-file` switches:

```
#> usql -f some-database-script.txt
```

You might be wondering what happens if the file loaded using the `-f` switch contains commands and settings that conflict with the general `$HOME/.usqlrc` configuration file. This issue is important, especially if you want to prepare reusable usql scripts. Luckily, usql offers an easy solution: Just add the `-X` or `--no-rc` option when launching usql at the prompt, and usql will completely ignore (for that session only!) the default configuration file. During an interactive session, you can load and execute a command file as follows:

```
\i FILE
\ir FILE
```

`\i` executes the contents of `FILE`, and `\ir` is similar except it looks for the file in the directory of the current script. `\ir` is helpful because, if you use usql on a regular basis, sooner or later you will end up creating your own library of usql scripts that could be organized in several folders and might even call each other.

Connecting to Databases

Usql opens a database connection by parsing a string and passing its content to the appropriate database driver. Database connection strings (aka "data source names" or DSNs) can be passed to usql directly on the command line or at any moment during an interactive session. In an interactive session, you can pass all the necessary parameters (driver, database name, etc.) separately via the `\c` meta command. In general, DSNs that connect to multi-user database servers like PostgreSQL or MariaDB have the following structure:

```
driver+transport://user:pass@host/dbname
```

The `driver` part is the name of the driver you wish to use (which corresponds to the type of database) or any of its aliases allowed by usql. When connecting to a PostgreSQL database, for example, the driver may be `postgres` or `pg`, whereas with MySQL, you should use `mysql` or just `my`.
To connect to a database from the Linux command line, use the following command:

```
#> usql my://marco:mypassword@localhost/customers
```

or, after launching usql, from its own prompt:

```
\c my://marco:mypassword@localhost/customers
```

The steps are slightly different if you want to connect to a serverless, file-based database such as SQLite3. In this case, the DSN has a much simpler structure:

```
driver:/path/to/file-on-disk
```

where the driver could be `sqlite3`, `sq`, or `file`, or the command might not specify a driver. Either of the following commands would open an already existing SQLite3 database contained in the single file `$HOME/my-sqlite-db.sqlite3`:

```
#> usql sqlite3://$HOME/my-sqlite-db.sqlite3
#> usql $HOME/my-sqlite-db.sqlite3
```

The second command will work without errors only if the `$HOME/my-sqlite-db.sqlite3` file already exists and is, indeed, an SQLite3 database. If your intention is to make usql create a new, empty SQLite3 database and save it into a file with the specified name, you must use the first command, which includes a recognizable driver name.
To close an open connection from inside usql, just type `\Z`. In any moment, you can also verify the parameters of the connection you are using by typing `\conninfo`.

Editing and Reusing Queries

Once you are connected to a database from inside a usql session, you can communicate with it just as if you were using a native client. Usql also keeps the current query in a dedicated buffer that you can edit and reuse. Use the `\e` meta command to edit the buffer and the `\w` command to write a query into the buffer. The `\p` command shows the buffer contents and `\r` resets the buffer.
Use the `\g` command to re-execute the query in the buffer. If you also want to execute every value of the result, type `\gexec`. The `\gset` command stores the result of the query in usql variables for further reuse. ■

Info

[1] Introduction to SQL: [http://www.w3schools.com/sql/]
[2] Usql: [https://github.com/xo/usql]
[3] Usql release page: [https://github.com/xo/usql/releases]

Author

Marco Fioretti is a freelance author, trainer, and researcher based in Rome, Italy, who has been working with Free and Open Source Software since 1995 – and on open digital standards since 2005. Marco is a Board Member of the Free Knowledge Institute ([http://freeknowledge.eu]), and he blogs about digital rights at [http://stop.zona-m.net].

History Lesson

For admins like Charly, who try to avoid typing at all costs, the shell offers an excellent opportunity to avoid wear on your fingertips in the form of built-in history. By Charly Khünast

There are commands that I type several dozen times a day – `grep <something> /var/log/syslog` is such a classic. The shell keeps a history of all my entries; thanks to the `history` command, I can always see in a numbered list which commands I typed last.

The `history` command is not a separate tool; typing `which history` at the command line just drops you into a black hole. Instead, `history` is a part of the shell, a built-in keyword. `history`'s killer feature, for which lazy people like me are eternally grateful, is the interactive search. You enable it with Ctrl + R, changing the command-line prompt to `(reverse-i-search)`':`.

If you start typing now, for example, the word *net*, the shell will show you the last command typed containing *net*. When you press Ctrl + R again, the history feature shows you an increasing number of older commands that contain *net* (Figure 1).

There are a number of other ways to execute commands stored in the history one more time. To repeat just the last command entered, you can do any of the following:

- Press the up arrow
- Press Ctrl + P ("previous" on keyboards without arrow keys)

```
(reverse-i-search)`net': sudo nethogs enp1s0
```

Figure 1: Interactive search with Ctrl+R.

- Type `!!`
- Type `!-1`

Sometimes using relative addressing backwards through history proves helpful. In the example from Figure 2, I reran the third-to-last command from the history by typing `!-3`. If you wanted to repeat the last command that started with *echo*, you would just need to type `!echo`.

You can also access the parameters from previous commands. If you just typed `ls .bashrc`, you can enter `vim !!:$` to open `.bashrc` in the editor. If you have a command that requires root privileges, `sudo !!` does the trick. In the meantime, I defined `but` as an alias (Figure 3).

```
charly@glas:~$ echo "brave"
brave
charly@glas:~$ echo "new"
new
charly@glas:~$ echo "world"
world
charly@glas:~$ !-3
echo "brave"
brave
```

Figure 2: Going back three commands.

```
charly@glas:~$ alias but='sudo $(history -p !-1)'
charly@glas:~$ ls -lha /root/.bashrc
ls: cannot access '/root/.bashrc': Permission denied
charly@glas:~$ but
-rw-r--r-- 1 root root 3.1K Oct 22  2015 /root/.bashrc
```

Figure 3: No? But! Oooh...

Occasionally, however, I find the history's length problematic, as it only stores 1,000 entries on my test system. This is not enough for me, so I added a `HISTSIZE=10000` line to the `/.bashrc` file to multiply the history size by 10. I also added `HISTCONTROL=erasedups` to `/.bashrc`. This means that the `history` command, which I type several times, is only saved once – this saves space and gives a better overview. ■

Smartphone-based two-factor authentication

Double Your Security

Protect your system from unwanted visitors with two-factor authentication. By Charly Kühnast

If the only protection between an attacker and a user account is a password, security-conscious administrators start to get nervous – and rightly so. Although strong passwords can be enforced, carelessness cannot be ruled out. Two-factor authentication (2FA) provides additional protection against unwanted visitors, even if a user chooses a weak password. While the user's password remains as the first authentication factor, a six-digit numerical code with a limited validity period generated by a smartphone authenticator app adds a second factor.

In this article, I will show how to require a one-time code at login (in addition to the user's password) by creating an app on the user's smartphone. This procedure was developed by the Initiative For Open Authentication (OATH) and has been an Internet Engineering Task Force (IETF) standard since 2011.

Listing 1: Installing Authentication Module

```
01 $ sudo apt install libpam-google-authenticator
02 $ google-authenticator
```

Getting Started

For this article, I am using Ubuntu 20.04, but the procedure is very similar on other distributions. You have a Linux client and a server. On the server, which goes by the name of *influx* in this example, I have an account belonging to user *bob*. Bob has been logging in with a password only. However, his organization now wants to switch Bob's account to 2FA.

I'll start by installing the authentication module on Bob's client (Listing 1, line 1) and then log in as *bob* and start the module (line 2)

The module first prompts you to decide whether the authentication should be time-based. It wants to know if the identical time – in terms of Coordinated Universal Time (UTC) [1] – exists on the two systems involved (smartphone and computer console). Reply yes since all systems today use Network Time Protocol (NTP) to synchronize their time.

Next a QR code (Figure 1) appears, which you scan with a One-time password (OTP) app that you install on your smartphone; an OTP is only valid for a single

Figure 1: The QR code generated by Google Authenticator can be scanned using an OTP app like FreeOTP.

Photo by Scott Webb on Unsplash

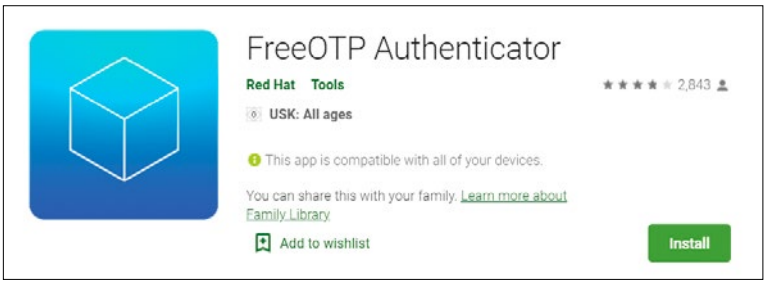


Figure 2: Unlike Google Authenticator, Red Hat's FreeOTP is an open source application.

use. There are plenty from which to choose; you can use any app that uses the Time-based One-time Password (TOTP) protocol. TOTP generates time-limited, one-time passwords based on the Hash-based Message Authentication Code (HMAC). For example, Google Authenticator is a very popular OTP app, although it is not open source.

For this example, I will install the FreeOTP app developed by Red Hat, which is available for both iOS [2] and Android [3], on the smartphone (Figure 2). After you scan the code, a new button will appear in the app that lets you generate a one-time password on demand with a validity period of 30 seconds.

Now set aside the smartphone and return to the console. Below the QR Code in Figure 1, you will find a number of emergency scratch codes. If you lose your smartphone, you can still log in with these codes to generate a new QR code and start over. Each of the emergency scratch codes can only be used once. Keep these codes in a safe place.

Google Authenticator will now ask you a series of security questions, all of which you can safely answer with *y* (Figure 3). The idea is to limit the number of logins per time interval, but at the same time ensure a certain tolerance for time differences between client and server.

You need to complete these steps for each user on the system who will be using 2FA. On the client side, all the work is done; time to work on the server.

Modifying PAM

To enable 2FA access, you need to modify two configuration files, for which you need root privileges. First, modify the `/etc/ssh/sshd_config` file (Listing 2). Find the two lines that begin with `UsePAM` and `ChallengeResponseAuthentication` and make sure that both end with `yes`.

Listing 2: Modifying /etc/ssh/sshd_config

```
UsePAM yes
[...]
ChallengeResponseAuthentication yes
```

Listing 3: Editing /etc/pam.d/sshd

```
[...]
@include common-auth
auth required pam_google_authenticator.so
[...]
```

Next, edit the `/etc/pam.d/sshd` file, again working as root. After the `@include common-auth` line at the top of the file, add the following line:

```
auth required pam_google_authenticator.so
```

The file should now look like Listing 3.

Now type the command

```
systemctl restart ssh
```

to start the SSH service. At the next login attempt via SSH (Figure 4), the server now not only

```
Do you want me to update your "/home/bob/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y

By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) y

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
You have new mail in /var/mail/bob
```

Figure 3: Yes (*y*) is the right response to all of Google Authenticator's questions.

```
bob@gw:~ ssh bob@influx
Password:
Verification code:
Linux influx 4.19.97-v7l+ #1294 SMP Thu Jan 30 13:21:14 GMT 2020 armv7l

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Sat Sep 19 13:15:00 2020 from 10.0.0.254
```

Figure 4: In addition to the user password (`Password:`), the login dialog now also prompts you for the one-time password (`Verification Code:`).

Listing 4: Modifying /etc/pam.d/login

```
[...]
@include common-auth
session optional pam_motd.so noupdate
# insert this line:
auth required pam_google_authenticator.so
[...]
```

Listing 5: Modifying /etc/pam.d/gdm-password

```
[...]
@include common-auth
# insert this line:
auth required pam_google_authenticator.so
[...]
```

prompts for the user password (`Password:` in Figure 4), but also the one-time password (`Verification Code:`), which you generate with Google Authenticator.

Console Login

My changes so far only apply to access via SSH. If you want to enable 2FA for the local login (the console) in addition to the remote login (the smartphone), you need to change the `/etc/pam.d/login` file (Listing 4).
To do this, insert the following line

```
auth required pam_google_authenticator.so
```

after the `@include common-auth` line. The `session optional pam_motd.so noupdate` line is used to display notifications (Message of the Day); it is not available on all systems.

Gnome Display Manager

If your console system uses a Gnome graphical user interface, you can also enable 2FA authentication at login time. To do this, you make the same changes previously discussed, but in a different file: `/etc/pam.d/gdm-password` (Listing 5). After a restart, Gnome will now prompt you for the second factor at login time.

Passwordless Login

Going back to logging in via SSH, many users prefer passwordless access via public key authentication. To do this, the user *bob* enters the command

```
ssh-keygen -t rsa -b 4096
```

on their client to generate a key pair (Figure 5).
After that, the command

```
ssh-copy-id bob@influx
```

is sufficient, followed by the input of the current password. Bob can now log on to the *influx* server without entering a password. Passwordless login can also be combined with 2FA. To do this, change the two configuration files on the server that I discussed previously. First open `/etc/ssh/sshd_config` and enter the following line at the end of the file:

```
AuthenticationMethods ➥
  publickey,keyboard-interactive
```

Second, edit `/etc/pam.d/sshd`. Here you need to disable the line that reads `@include common-auth` by adding a hashtag (`#`) at the start of the line:

```
#@include common-auth
```

Then run the `systemctl restart ssh` command to restart the SSH service. When Bob now logs on to the server, he does not have to enter a password, but he does have to enter the one-time password from the smartphone app.

Conclusions

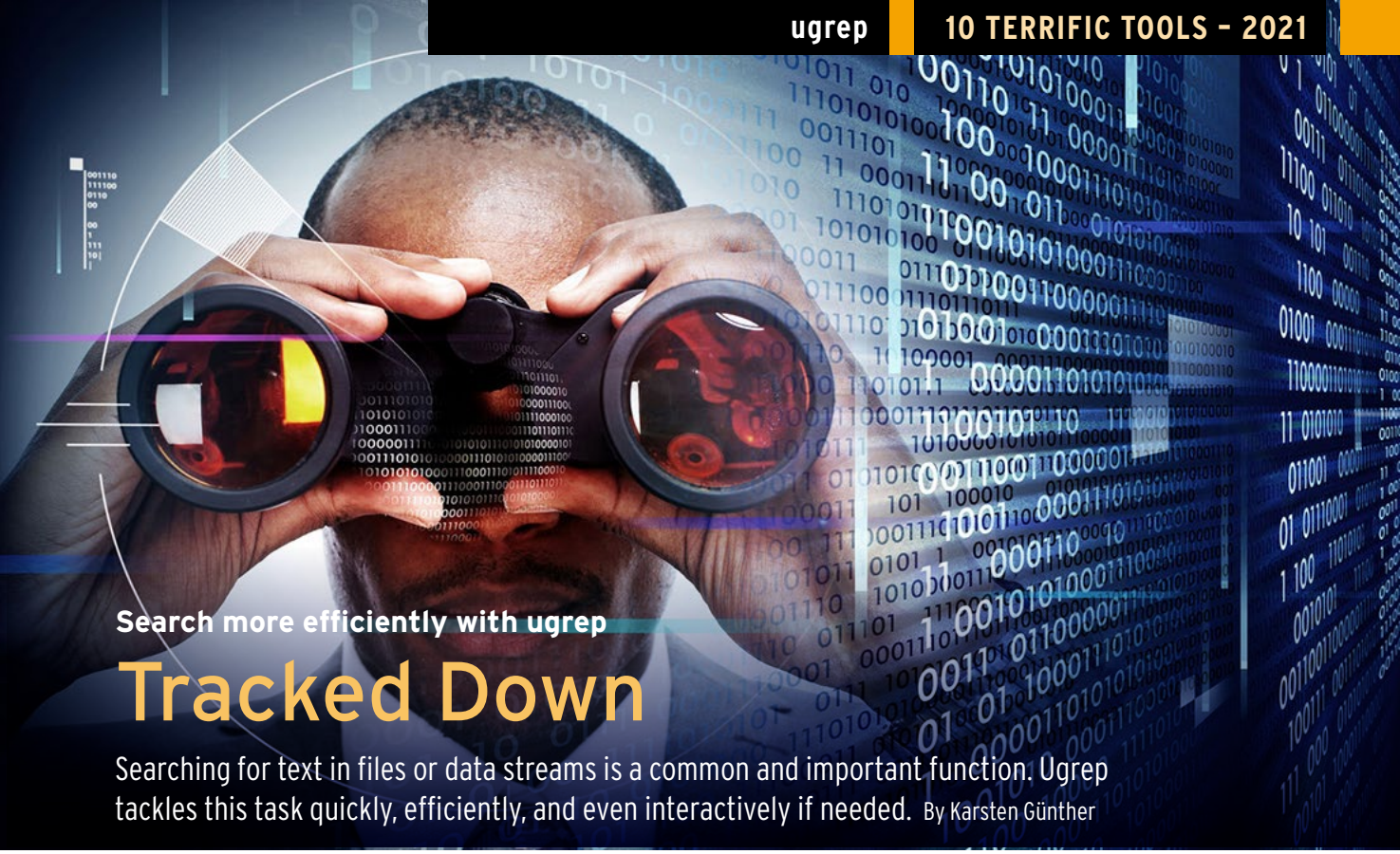
Security is not witchcraft. As shown here, even simple mechanisms such as 2FA can make logging on to a system far more secure. 2FA gives you additional protection against unwanted visitors, even if users choose weak passwords. ■

Info

[1] UTC: [https://www.timeanddate.com/time/aboututc.html]
[2] Apple iOS: [https://apps.apple.com/us/app/freeotp-authenticator/id872559395]
[3] Android: [https://play.google.com/store/apps/details?id=org.fedorahosted.freeotp]


```
bob@gw:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_rsa):
Created directory '/home/bob/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_rsa.
Your public key has been saved in /home/bob/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0Tgf869HRhlXX07SeKbqqzkMuWHdFr+Hj5iKJmQNqKc bob@gw
The key's randomart image is:
+---[RSA 4096]----+
|               .o=|
|          o    o+B|
|      .  + +    Bo|
|     . .  + +. +  |
|    .   oSo...=   |
|   . . o * . +.+  |
|    o o . = o o.o |
|   E   . o.o..++..|
|       o. ++=o.o.|
+----[SHA256]-----+
```

Figure 5: For a passwordless login, Bob generates a key pair on the client.



Search more efficiently with ugrep

Tracked Down

Searching for text in files or data streams is a common and important function. Ugrep tackles this task quickly, efficiently, and even interactively if needed. By Karsten Günther

Grep is one of the oldest Unix commands. The abbreviation "grep" stands for Global/Regular Expression/Print or Global search for a Regular Expression and Print out matched lines. It picks up on the syntax of the original Unix editor, QED, which used `g/re/p` to search for patterns in text files. In addition to fixed search terms, it can also search for patterns with wildcard characters. The GNU variant of grep is normally installed on Linux. It extends the features of the original grep in some places, for example, allowing recursive searching in directories.

Another variant of grep, agrep (approximate grep) [1], extends text searching to include fuzzy searches. It also finds near misses as long as the differences are below a specified threshold, known as the word distance. This is calculated from the necessary permutations, deletions, and additions of letters that convert the search pattern into the actual data.

In addition, there are some variants of grep that also find search patterns in certain archive types, such as ZIP files. These programs are relatively slow, since they first need to unpack the archive. However, all grep variants used on Linux can also read data from pipes via the standard input channel and write the results to the standard output channel for searching in archives (Listing 1).

ugrep

Ugrep can do all of this and more without explicitly unpacking the data streams. In addition, the program is known for its exceptionally fast processing speed. To speed up the search, it uses multiple threads if necessary.

On Debian and Arch Linux, setting up ugrep is easy. Debian has the tool in its repositories; with Arch Linux, you can use the AUR. For all other distributions, you will have to install ugrep from the source code [2]. The commands required for this are shown in Listing 2.

Ugrep is programmed in C++, has been around for several years, and is available not only on Linux, but also on other operating systems. Search patterns specified as regular expressions can span consecutive lines, a thing that many other grep variants cannot do. By default, ugrep assumes Unicode as the encoding for the search data.

Ugrep supports archive types including CPIO, JAR, PAX, TAR, and ZIP, compressed with all common methods (BZIP, GZ, LZ, and XZ). In addition, you can use filters to prepare data in special formats in advance. For example, PDF documents can be converted to text with a filter, before ugrep performs the search.

Like all grep variants, the program is largely controlled by options. For most options, as usual, there is a short form (`-<O>`) and a long form (`--<Option>`). Table 1 summarizes the most important options. Besides all of this, the developer suggests a number of alias con-

Listing 1: Archive Search

```
$ zcat archive.gz | grep <pattern>
```

Listing 2: Installing ugrep

```
$ git clone https://github.com/Genivia/ugrep
$ cd ugrep && ./build.sh
$ sudo make install
```

structs for the .bashrc to ensure compatibility with GNU grep, for example (see Table 2). Some of these short forms rely on the ug command variant. In this form, ugrep reads in a configuration file (by default $HOME/.ugrep) which can contain special settings. This means that important presets can be applied implicitly without having to specify them at the command line every time.

Ugrep supports several search pattern variants, which you enable through appropriate options (see the "Patterns" box). Besides simple and extended regular expressions like GNU grep, ugrep also supports Perl regexes and word patterns. In addition to these default patterns, which always define positive patterns, ugrep can also use negative patterns (exclusion patterns). They let you, for example, ignore matches if they occur in comments. Files whose names match a certain pattern can also be excluded from the search. The --not option has a special effect: All patterns to the right of it are used by ugrep as exclusion patterns.

Table 1: Important Options

-a	Interpret data as text
-c	Match count
-e <pattern>	Search for specified pattern (can specify multiple patterns)
-E	Interpret search patterns as extended regular expressions (default)
--encoding=<encoding>	Set encoding for data
-f <file>	Load search pattern from specified file
-F	Interpret search pattern as string (special characters are considered as text)
--filter=<filter>	Pre-filter based on specified filter criteria
-G	Interpret search patterns as simple regular expressions
-i	Ignore case in pattern
-N <pattern>	Define negative search pattern
--not	Interpret all of the following search patterns as exclusion patterns
-O <extension>	Edit only files with the specified extension
-P	Interpret search patterns as Perl expressions
--pager=<pager>	Set pager for terminal output
-Q[<delay>]	Incremental search with optional delay
-R	Recursive search
-w	Word search
-X	Output in hexadecimal form
-z	Unpack compressed data streams in advance
-Z<Criteria>	Fuzzy search with set criteria for allowed deletions, insertions, or substitutions

Table 2: Suggested Alias Constructs

Alias	Function
alias uq = 'ug -Q'	Interactive, incremental search
alias ux = 'ug -UX'	Binary search
alias uz = 'ug -z'	Search in (compressed) archives
alias ugit = 'ug -R --ignore-files'	Grep for Git
Compatibility with classic variants	
alias grep = 'ugrep -G'	Search with simple regular expressions
alias egrep = 'ugrep -E'	Search with extended regular expressions
alias fgrep = 'ugrep -F'	Search without regular expressions
alias pgrep = 'ugrep -P'	Search with Perl regular expressions
Search in compressed data	
alias zgrep = 'ugrep -zG'	Archive search with simple regular expressions
alias zegrep = 'ugrep -zE'	Archive search with extended regular expressions
alias zfgrep = 'ugrep -zF'	Archive search for strings
alias zpgrep = 'ugrep -zP'	Archive search with Perl regular expressions

Extensions

In many places ugrep extends the other, classic program versions. The new features for patterns in file names ("globbing") are particularly interesting. For example, **/ stands for any number – even zero – directories. At the end of a path definition, /** stands for any number of files. The special case \\? addresses zero characters or one. In the man page, the globbing section summarizes these features and also gives numerous examples.

Special environment variables let you additionally control the behavior of ugrep. $GREP_PATH simplifies access to so-called pattern files (i.e., files that define search patterns); the -f option enables this feature. Patterns in external files are a good way to keep complex search patterns permanently. Some options, including -Q, can use an external editor that the key combination Ctrl + Y starts. If the $GREP_EDIT environment variable is set, ugrep uses the editor defined

Patterns

The term "pattern" usually appears in multiple contexts with different meanings in search programs like ugrep. Patterns in file names determine which files the program processes. The file content patterns are the actual search patterns for which it searches the processed files. With ugrep, these may also be across lines. Ugrep and some other search programs also support negative patterns. They are used to exclude files or not to display corresponding matches. In fact, ugrep takes this procedure quite far: In the program's documentation, there is a separate section, *Search this but not that with -v, -e, -N, --not, -f, -L, -w, -x*, that deals with the finer points of this subject.

there; otherwise the one defined in `$EDITOR` is used.
The `$GREP_COLOR` and `$GREP_COLORS` environment variables let you specify when and how ugrep color highlights matches when using the `--color` option. The `GREP_COLORS` section in the man page describes this in more detail.
But the really outstanding extensions in ugrep are the incremental search feature and the user interface.

User Interface

Grep programs are usually used interactively in command lines, scripts, or pipes; in many cases the results then act as input for further commands. This also works without any restrictions in ugrep. In addition, the developer has also paid great attention to extended interactive usability. For example, incremental searching is currently an absolutely unique selling point of ugrep. The user interface used for this was modeled on editors such as Emacs and is normally reserved for GUI programs.

With this type of search, each additional letter specified further refines the search and reduces the number of matches. All lines that match the previous entries are then displayed. For this form of search, ugrep provides a special interface that you enable using the `-Q` option. As an argument of `-Q`, you can specify a small delay that ugrep waits for before evaluating the input.
The `Q>` prompt now appears in the upper left corner of the terminal. Everything you type is interpreted by ugrep as a search pattern; each additional keystroke refines the search. Typos can be corrected with the backspace key. In the example from Figure 1, we called ugrep with the `-ZQ` (fuzzy, interactive) options and searched for "alles" ("everything" in German). Due to the fuzzy search, ugrep also finds "alpes", "alls", "ales," and so on.
This feature is so powerful that ugrep in this mode can sometimes even replace a pager for displaying output. For example, `man ugrep | ugrep -Q` displays the man page of ugrep and lets you define exactly which search term it should display. The output can also be shifted vertically with the arrow keys; Esc ends the mode again.
On top of that, this option can be combined with others. In case you need more than the ability to see just the line with the match, you can add two context lines before and after the match to the output using `-C2`. In this form, ugrep is extremely useful as an alias (`alias q2='ug -C2 -G '`), shell function, or script.
The ability to search archives is a similar case. Many modern documents are in complex formats like EPUB, ODF, etc. There, the options usually only act on metadata in the document containers – often ZIP archives. To search in the actual contents, you have to unpack these archives, which is done either by a filter (more on that later) or the `-z` option, often combined with `-r` for recursive.
Ugrep supports fuzzy searching with the `-Z` option, which may be followed by a number appended directly without spaces. The latter determines the degree of fuzziness, that is, the permissible number of errors (omitted, added, swapped characters). The default is 1. Larger values quickly lead to many additional hits, but this sometimes makes the results unusable.
However, the type of allowable errors can be specified: With a prefix of `+` or `-`, the specification refers only to additions or omissions, respectively. The tilde (`~`) groups several errors. `-Z~-2` means that up to two omissions or swaps are allowed. The `--sort=best` option sorts the output so that the files with the best matches appear first.
Ugrep uses some function keys for special tasks in interactive mode.

```
Z>alles
alpes                463/tcp
alpes                463/udp
h323hostcallsc      1300/tcp
h323hostcallsc      1300/udp
mt-scaleserver      2305/tcp
mt-scaleserver      2305/udp
call-sig-trans      2517/tcp
call-sig-trans      2517/udp
sonuscallsig        2569/tcp
sonuscallsig        2569/udp
allstorcns          2901/tcp
allstorcns          2901/udp
caller9             2906/tcp
caller9             2906/udp
stonefalls          2986/tcp
stonefalls          2986/udp
ewinstaller         4091/tcp
ewinstaller         4091/udp
parallel            4989/tcp
parallel            4989/udp
alesquery           5074/tcp
alesquery           5074/udp
h323callsigalt     11720/tcp
h323callsigalt     11720/udp
(END)
```

Figure 1: Ugrep enables interactive, fuzzy, and incremental searches.

For example, F1 activates the online help (Figure 2) where ugrep displays the current keyboard shortcuts. You can enable additional options by calling them in this mode. For example, after pressing F1, the key combination Alt-Left + Shift + Z activates fuzzy searching.

Invoked with the `--save-config` option, the program creates the `$HOME/.ugrep` configuration file. If necessary, you can create another file using `--save-config=/<path>/<file>`). Similarly, `--config` reads configuration files. Calling ugrep as `ug` automatically parses the configuration. Since configuration files are a powerful means of controlling ugrep, there is also the shorthand `---<file>` for loading. You can create configuration files with certain preset options with the following command:

```
$ ugrep -<option> [...] --save-config
```

The configuration files are well commented and can be easily customized with a text editor if needed.

Filters

Ugrep tries to determine the type of an examined file based on the data it contains, the file name extension, and the signature (the "magic byte"). In this way, the search can be specially prepared for certain file types (i.e., filtered). Here the filter extracts the text components from the data streams. These filters execute a command, a script, or a specific function, with pipes if necessary. They are prepended to the search process via the `--filter=<Filter>` or `--filter-magic-label=<Label>:<MagicByte>` option. In the form `--filter=<filter>`, the `<filter>` consists of an expression of the form `<Ext>:<command line>`. `<Ext>` is a comma-separated list of file name extensions for which you want the filter to apply, such as `.doc,.docx,.xls`. The `*` character is a special case that acts on all files, especially those for which there are no other filters.

The `<command>` line must be constructed to read input via the standard input channel and write the results to the standard output channel. Typical commands include `cat` (pass everything) and `head` (pass the first lines of text), but tools like `exiftool` (extract and pass metadata) or `pdftotext` (extract text from PDFs) can also be included this way. Some commands, like `pdftotext`, require options to work correctly – in this case `pdftotext % -`. You then need to quote spaces in the command lines to protect them:

```
--filter='pdf:pdftotext % -'
```

The `--filter-magic-label=<Label>:<Magic>` option lets you extend the filtering mechanism to data streams that ugrep then classifies by reference to the magic byte. Details can be found in the man page.

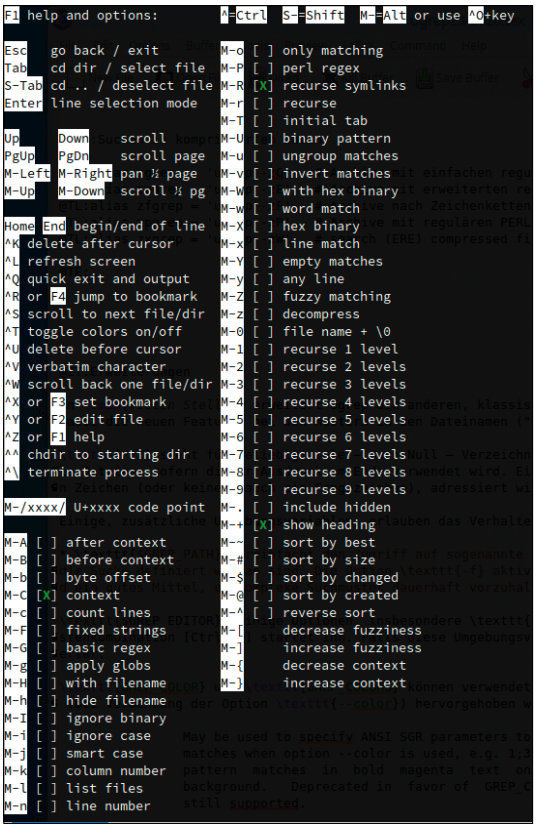


Figure 2: The ugrep help function conveniently comes with a built-in configuration mode.

Multiple filters can be specified as comma-separated lists. A combined definition for PDF and Office documents might look like the one shown in Listing 3.

Listing 3: Combined Filter Definition

```
--filter="pdf:pdftotext % -,odt,doc,docx,rtf,xls,xlsx,ppt,pptx:soffice --headless --cat %"
```

Conclusions

Ugrep belongs on every computer. It replaces and complements the standard commands quite excellently, and anyone who has to deal with text searches should familiarize themselves with it. The incremental search alone is so useful that it more than justifies the minimal training time. ■

Info

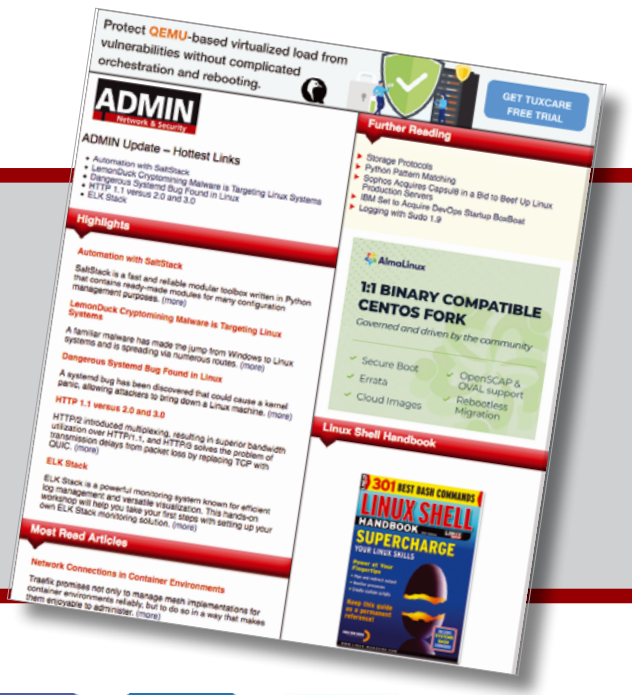
[1] agrep: [https://linux.die.net/man/1/agrep]
[2] ugrep: [https://github.com/Genivia/ugrep]

GET TO KNOW ADMIN

***ADMIN Network & Security* magazine is your source for technical solutions to real-world problems.**

ADMIN is packed with detailed discussions aimed at the professional reader on contemporary topics including security, cloud computing, DevOps, HPC, containers, networking, and more.

Subscribe to *ADMIN* magazine and get 6 issues delivered every year



Want to get ADMIN in your inbox?

Subscribe free to ADMIN Update

and get news and technical articles you won't see in the magazine.

ADMIN Network & Security