PRACTICAL PORTS

Programmers Programming Potpourri

BY BENEDICT REUSCHLING

This column covers ports and packages for FreeBSD that are useful in some way, peculiar, or otherwise good to know about. Ports extend the base OS functionality and make sure you get something done or, simply, put a smile on your face. Come along for the ride, maybe you'll find something new.

One of Unix's strengths is that even though it involves a lot of typing, it grew a number of helper utilities over the years. People developed neat little shortcuts to avoid typing or repeating the same keystrokes. One such invention was the shell history. Why type something again when you can retrieve it from way back when you typed it successfully the last time. For the uninitiated, typing the right keystrokes to invoke searching in the shell history will make them fold a wizard hat in no time—mouth still open in astonishment. Once they learn how to do that, they might become less impressed. Then it is time to install misc/mcfly and race through your shell history just like Marty did in the movies. The intelligent search takes your current directory into account or the context within which you used the program and offers you the proper commands. It does not mess with your normal shell history file, allowing you to get comfortable with mcfly.

Why type something again when you can retrieve it?

Programmers like to listen to audio when hacking on the latest code. A soothing visualization of the sound waves hitting your ears has been popular since the days of Winamp. If you want the same on the console, audio/cava is there for you. No matter what you use: Pulseaudio, fifo (mpd), sndio, squeezelite or portaudio, jumping bar graphs will appear

to match your tunes. Sweet, but don't forget to wear headphones or your neighbors get angry. Talk about eavesdropping.

Programmers not only write code, but they also need to test it. Performance is still crucial for the user experience, even today with hardware the Apollo guidance computer would only have dreamed of. To ensure a code change did not make things run more slowly, benchmarks are used. One such benchmark for the commandline is benchmarks/hyperfine. It supports arbitrary shell commands for statistical analysis across multiple executions. A warmup phase ensures that caches don't interfere with measuring the right things. Outliers caused by some other program running in the background are detected and hyperfine can run with varying numbers of threads. The results are output in CSV, JSON, Markdown, and AsciiDoc. Your thesis basically writes itself these days...

Did you ever have the need to securely transfer a file or string like a password over to another computer? If SSH is too complex for you with its 740-character, public-key exchange, but you don't want to compromise on the security front, send your files through net/py-magic-wormhole instead. When sending a file, a short, humanly pronounceable character string is generated. Input this one-time key on the receiving side and the wormhole does its magic using Rendezvous Message Exchange and PAKE-based security. It even allows tab-completion for the key, saving you some keystrokes. The magic wormhole can also serve as a replacement for ssh-copy-id for the initial SSH key exchange. In the example below, I transferred my backup.zip over to another machine. On the receiving side, I entered the code and confirmed the file. Moments later, the file was on the other side, just like a 30,000 light-years trip through the delta quadrant.

```
$ wormhole send backup.zip
Sending 1.2 GB file named 'backup.zip'
Wormhole code is: 1-specialist-apple
On the other computer, please run:

wormhole receive 1-specialist-apple

Sending (<-[redacted]:60188)..
100%|[redacted]| 1.23G/1.23G [00:14<00:00, 87.6MB/s]
File sent.. waiting for confirmation
Confirmation received. Transfer complete.
$
```

```
$ wormhole receive
Enter receive wormhole code: 1-specialist-apple
 (note: you can use <Tab> to complete words)
Receiving file (1.2 GB) into: backup.zip
ok? (y/N): y
Receiving (->tcp:[redacted]:31976)..
100%|[redacted]| 1.23G/1.23G [00:14<00:00, 86.7MB/s]
Received file written to backup.zip
$
```

A common exercise for my undergraduate computer science students is to let them parse text files in Unix. Comma separated values (CSV) are still popular file types in these assignments. This usually happens before students get introduced to databases and SQL. If they knew before, they'd probably all use the next tool I'm introducing without ever looking back: textproc/csvq. Read, update, and delete CSV files to your heart's content. An interactive shell or commandline-mode did not let the developers stop there. Execution of multiple operations in sequence is possible in managed transactions. Variables, cursors and yes, temporary tables are also supported. Oh, did I mention that besides CSV, JSON is also supported in full UTF-8 and UTF-16 variants? Because why not?

Another tool is textproc/dasel (data selector), which supports JSON, YAML, TOML, and XML in addition to CSV. It can convert between the different formats, put new content in and delete existing entries. No need to learn a new tool each time a new language comes around. Chances are, someone has it already covered with "just another supported tool."

Developers usually have a lot of machines, both at work and home. They also want to have their familiar configuration everywhere, so that they are productive wherever they are. To stop copying the dotfiles (that hold all the configuration magic) from one machine to the next, there

is sysutils/chezmoi. It synchronized the dotfiles across machines in a secure way. A single command can pull down all these configuration files on a new machine. When changing settings on one machine, a single "chezmoi update" will ensure the change is present on another machine, too.

A quickstart guide and other documentation helps you use chezmoi in no time.

If you like syntax highlighting and line numbers in your output and wondered why cat(1) never evolved this feature, consider using textproc/bat. A cat clone with wings, capable of showing you non-printable characters, does automatic paging, and includes git integration.

Some developers like to brag about how much code they've written. But how much of it are comments and blank lines? If you want to find out, devel/tokei can tell you all about it. Tokei understands multi-line and nested comments and ignores comments within strings. It supports over 150 languages, colors, and is very fast, even with big projects.

Some developers like to brag about how much code they've written. But how much of it are comments and blank lines?

When I have not used a tool for a while, I tend to forget all the command line switches. Then I have to look up the man page and build my command line again (especially with multiple such flags). I should probably have written it down in a cheatsheet to save time. Something similar must have gone through the head of the misc/cheat developer. How about all there is to know about using tar? You can find this example on its github project page:

```
$ cheat tar

# To extract an uncompressed archive:
tar -xvf '/path/to/foo.tar'

# To extract a .gz archive:
tar -xzvf '/path/to/foo.tgz'

# To create a .gz archive:
tar -czvf '/path/to/foo.tgz' '/path/to/foo/'

# To extract a .bz2 archive:
tar -xjvf '/path/to/foo.tgz'

# To create a .bz2 archive:
tar -cjvf '/path/to/foo.tgz' '/path/to/foo/'
```

Very handy and sometimes I find some new nuggets of wisdom in there, too. Try it with commands like rsync. Run

```
cheat -l
```

to list all available cheatsheets. Of course, you can add cheatsheets of your own and share them with the community. People will still think you are a Unix wizard by sharing these, perhaps even more then.

And if you thought that bat was the only port named after an animal mentioned in this column, prepare to meet two new: dns/dog and dns/doggo. The latter was inspired by the former, but since the author did not know Rust (which dog is written in), he rewrote it in go (dog + go, you get the idea). Both are a modern CLI DNS clients (like dig) and support protocols like DNS over HTTPS (DoH), DNS over TLS (DoT), DNS over TCP/UDP, and DNSCrypt. JSON support is integrated in both dns/dog and dns/doggo and at least one of them supports multiple resolvers at once. I'll let you find out which, so that you can also enjoy the colors it produces. What a dark world we used to live in. But then LSCOLORS slowed things down too much, but that's a story for another time.

But while we're on the subject of colors, if you like them too much, why don't you mix some sysutils/lsd into your Unix work? This pretty ls alternative has both colors and icons and was written in Rust (is there anything left that isn't?). It certainly looks ... colorful. Don't blame me for headaches or other woes incurred by using any kind of lsd.

Back to the subject of this Journal issue, but staying with the colors: have you ever looked at your git diff output and wondered if that could be improved? Wonder no more, devel/git-delta comes to the rescue (even if you did not think you needed help). Layout and style for diffs is what delta allows you to spend hours on, just to get the right color scheme. It includes a pager and of course themes so that you don't have to start from scratch. Line numbers, side-by-side views, boxes with customized lines are all possible. Chances are, you will spend even more time reviewing code with this tool. See what they tricked you into?

When your disk is overflowing with code snippets, patches, and uncommitted work, it's high time to open sysutils/duff. No, not the fictional beer, the duplicate file finder. It styles itself to be a prettier du utility, but you'll be the judge of that. It certainly is fast and gives you an overview of where disk space was eaten the most. I'm sure that duffs grouping into local, network, and special devices gives sysadmins valuable information at a glance. Better run

```
git gc
```

once in a while to garbage collect some temporary files no longer needed.

Lastly, you could wake me up in the middle of the night and I would still be able to tell you the find commandline syntax to search for a file or directory. It might take me some time to spell out

```
find / -name "*foo*"
```

for you and I will be grumpy at you for waking me up. I value my beauty sleep, so I'll tell you about sysutils/fd. This tool is so easy to use that it only requires

```
fd foo
```

to list the results. Much easier, more sleep for me. Goodnight!

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. He is one of the hosts on the bsdnow.tv podcast.