

★

★

like this?
there are more
zines at:
<http://jvns.ca/zines>

→ LINUX ←

DEBUGGING TOOLS

you'll ♥



A SMALL WIZARD TOOL HANDBOOK

FOR ANYONE WHO WRITES (OR RUNS!!)
PROGRAMS ON LINUX COMPUTERS

CC-BY-NC-SA

Julia Evans, wizard debugging industries

BY: JULIA EVANS

05 X tool!

wireshark

Wireshark is an amazing GUI tool for network analysis. Here's an exercise to learn it! Run this:

```
sudo tcpdump port 80 -w http.pcap
```

While that's running, open metafilter.com in your browser. Then press Ctrl+C to stop tcpdump. Now we have a pcap file to analyze!

Explore the Wireshark interface! Questions you can try to answer:

- ① What HTTP headers did your browser send to metafilter.com? (hint: search frame contains "GET")
- ② How long did the longest request take? (hint: click Statistics → Conversations)
- ③ How many packets were exchanged with metafilter.com's servers? (hint: search `ip.dst == 54.186.13.33` (pinging metafilter.com from `ip.from`))

05 X tool!

tcpdump

tcpdump is the most difficult networking tool we'll discuss here and it took me a while to v it. I use it to save network traffic to analyze later!

See jvns.ca/zines for a zine all about tcpdump!

awesome thing

```
port 8997" is actually a tiny program in the "Berkeley Packet Filter" (BPF) language. BPF filters get compiled and they run really fast!
```

a "pcap file" ("packet capture") is the standard for saving network traffic. Everything understands pcap

```
sudo tcpdump port 8997 -w service.pcap
```

→

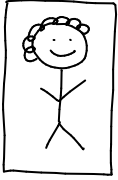
Everything understands pcap

Some situations where I'll use tcpdump:

- * I'm sending a request to a machine and I want to know whether it's even getting there. (tcpdump port 80 will print every packet on port 80)
- * I have some slow network connections and I want to know whether to blame the client or server. (we'll also need Wireshark!)
- * I just want to print out packets to see them (tcpdump -A)

what's this?

Hi! This is me:



JULIA EVANS
blog: jvns.ca ☺
twitter: @b0rk

and in this zine I want to tell you about

how I got
better at
debugging

These are 5 ways I've changed how I think
about debugging:

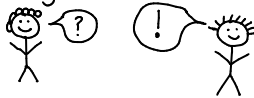
- Remember the bug is happening
for a logical reason.

It's never magic. Really. Even when it makes no sense.

- Be confident I can fix it

before:  "maybe this
is too
hard" now:  "well I've fixed
a lot of hard
bugs before"

- Talk to my coworkers



I found out that everything was ok ☺

`sudo ngrep some-id`

it was working, so I ran:

requests. I wanted to make sure

{ "some-id": "... } with all its

to a client so that it sent

Recently at work I'd made a change

output! We are SPIES ☺

matching network packets in ngrep's

in your browser. You should see

Then go to `http://metafilter.com`

`sudo ngrep -d any metafilter`

spy tool! Try it right now! Run:

ngrep is my favourite starter network

OS X
too!

ngrep

grep your
network!

* valgrind
* the Java ecosystem's fantastic
tools (Jstack, VisualVM, YourKit)
which your language is probably jealous of
* ftrace (for Linux kernel tracing)
* LTTng (d.tto)
* eBPF

(in general, my aim in this zine is to showcase
tools that I think don't get enough love ☺)
Some things I didn't have space for in this
section but wanted to mention anyway:

This section is about using `perf`
to answer that question. `perf` is a
Linux-only tool that is extremely
useful and not as well-known as
it should be.

Your programs spend a lot of time
on the CPU! Billions of cycles.
What are they DOING?!

section 3: CPU + `perf`
Linux
only

know my debugging toolkit



most importantly: I learned to like it



what you'll learn

I can't teach you in 20 pages to ♥ debugging (though I'll try anyway!) I can show you some of my debugging toolkit though!

These are the tools I reach for when I have a question about a program I want to know the answer to. By the end of this, I hope to have given you a few new tools to use!

☺

☺

So! I ♥ netstat because it tells me which processes are running on which ports. On OS X, use `lsof -i -P` instead.

proto	local address	PID / program name
tcp	0.0.0.0:5353	2993 / python

Here's what you'll see:

```
(sudo netstat -tunapl)
```

also known as

★ "tuna, please!"

★ "tuna, please!"

ports is really easy. It's just out which programs are listening on which needs to be "listening" on the port. Finding receive a request, a program (aka "server") a port (like 80) on a computer. To Every network request gets sent to

OS X tool

★ netstat

perf is not simple or elegant. It is a weird multi-tool that does a few different, very useful things. First, it's a sampling profiler.

Try running:

```
$ sudo perf record python (press ctrl+c after a few seconds)
```

saves a "perf.data" file

You can look at the results with:

```
$ sudo perf report
```

Mine says it spent 5% of its time in the PyDict-GetItem function. Cool! We learned a tiny thing about the CPython interpreter.

works everywhere ☺

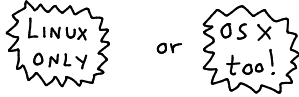
if you use perf to profile a Python program, it'll show you the C functions (symbols) from the CPython interpreter, not the Python functions. depend on your kernel version.

Section 1: I/O and

★ system calls ★

Hello, dear reader! In this zine, there are 3 sections of tools that I love.

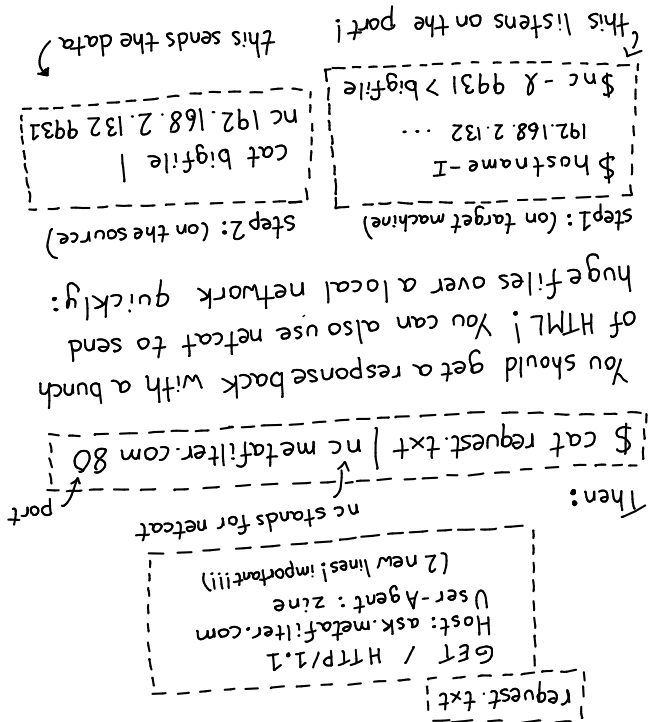
For each tool, I'll tell you why it's useful and give an example. Each one is either



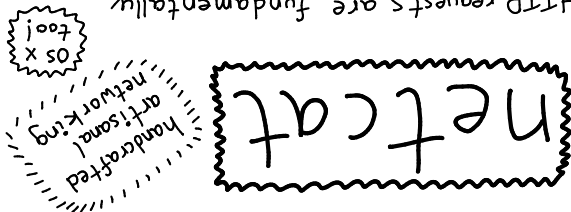
Some of the most basic questions you might have when you log into a misbehaving machine are:

- is this machine writing to or reading from disk? The network?
- are the programs reading files? Which files?

So, we're starting with finding out which resources are being used and what our programs are doing. Let's go!



HTTP requests are fundamentally really simple — they're just text! To see that, let's make one by hand! First, make a file:



Remember when I said perf only knows C functions? It's not quite true. node.js and the JVM (java, scala, clojure...) have both taught perf about their functions.

≡ node ≡
Use the perf-basic-prof command line option

≡ Java ≡
Look up 'perf-map-agent' on GitHub and follow the directions

... especially Java and node devs !

process	PID	%	function
ruby	1957	77	match-at

\$ sudo perf top

perf top doesn't always help. But it's easy to try, and sometimes I learn something!

* Ruby's internal regexp matching function

One day, I had a server that was using 100% of its CPU. Within about 60 seconds, I knew it was doing regular expression matching in Ruby. How? 'perf top' is like top, but for functions instead of programs.

dstat

LINUX ONLY

I love dstat because it's super simple. Every second, it prints out how much network and disk your computer used that second.

Once I had an intermittently slow database server. I opened up dstat and stared at the output while monitoring database speed.

```
$ dstat
send | rcv
```

0	} during this period, everything is normal
3k	
5k	
0	} DATABASE GETS SLOW
300 MB	
48 MB	} back to normal
0	
0	

Could 300MB coming in over the network mean... a 300MB database query?!
 $\hat{=}$ YES $\hat{=}$

This was an AWESOME CLUE that helped us isolate the problem query

Let's go!

- * my program is slow. whose fault is that?
- * is my service even running?
- * was the request wrong, or was it the response?

answer questions like:

is a nice language-independent place to answer questions like:

Every programming language uses the same network protocols! So the network

```
request => { GET /cats/42"
              name: "fruffy",
              colour: "blue" }
response => { my program }
```

I've devoted a lot of space in this zine to networking tools, and I want to explain why. A lot of the programs I work with communicate over HTTP.

section 2: networking

Here's what they look like:

They're constructed from collections (usually thousands) of stack traces sampled from a program. The one above means 80% of the stack traces started with "main" and 10% with "panda" and 10% with "alligator".

You can construct them from 'perf' recordings (see Brendan Gregg's flamegraph github for how) but lots of other unrelated tools can produce them too. I ♥ them.

Flamegraphs are an awesome way to visualize CPU performance, popularized by Brendan Gregg's Flamegraph.pl tool.

flamegraphs

github.com/brendangregg/flamegraph

! strace !!

LINUX ONLY

(I have a strace sticker on my phone)

strace is my favourite program. It prints every system call your program used. It's a cool way to get an overall picture of what your program is doing, and I ♥ using it to answer questions like "which files are being opened?"

```
$ strace python my_program.py
```

file descriptor ↓

```
read(3, "the contents of the file")
... hundreds of lines ...
connect(5, "172.217.0.163")
sendto(5, "hi!!!")
```

networking!



WARNING
strace can make your program run 50x slower. Don't run it on your production database

I can't do justice to strace here, but I have a whole other zine about it at

jvns.ca/zines

phew

I hope you learned something new.
Thanks for reading ♥

Thanks to my partner kamal for help reviewing and to the amazing Monica Dinculescu (@notwaldorf) for the cover art.

To learn more, see:

- ★ my blog: jvns.ca
- ★ my other zines: jvns.ca/zines
- ★ brendangregg.com

But really you just need to experiment. Try these tools everywhere. See where they help you track down bugs and where they don't.



strace really helped with that problem!



that didn't tell me much, oh well!

It takes practice, but I find these tools both fun and a useful job skill. I hope you will too!

There's also an opensnoop on OS X & BSD! That one is powered by DTtrace.

there are lots of eBPF-powered tools! Check out that GitHub repo to learn more!

Requires: Ubuntu 16.04+ or a ~4+ kernel version
Installation instructions at: github.com/lorvisor/bcc

opensnoop is a script that uses a new kernel feature called eBPF. eBPF is fast!

opensnoop won't slow you down. opensnoop run 10x slower. strace can make your program run 10x slower. But strace can do this

strace -e open -p \$PID
too! I just use

it will print out in real time every file being opened by a program. You might think...

```
opensnoop -p $PID
```

When you run

(kind of) 0.5x tool

opensnoop!
eBPF!

Hardware is cool. I've never used perf stat in earnest but I think it's awesome you can get so much info from your CPU.

how it works
Your CPU can track all kinds of counters about what it's doing. perf stat asks it to count things (like L1 cache misses) & report the results.

how to use it
perf stat is
This runs 'ls' and prints a report at the end.



perf stat!

how do I know if my program is using those caches?

If you're trying to do an operation in microseconds, CPU cache usage matters!

tip! google "latency numbers every programmer should know"

Your CPU has a small cache on it (the L1 cache) that it can access in ~0.5 nanoseconds! 200 times faster than RAM!

spy on your CPU!