

→ LINUX ←

DEBUGGING TOOLS

you'll ♥



A SMALL <sup>wizard</sup> <sub>tool</sub> HANDBOOK  
FOR ANYONE WHO WRITES (OR RUNS!!)  
PROGRAMS ON LINUX COMPUTERS

BY: JULIA EVANS

Hi! This is me:



JULIA EVANS  
blog: [jvns.ca](http://jvns.ca) ☺  
twitter: @b0rk

and in this zine I want to tell you about

how I got better  
at debugging

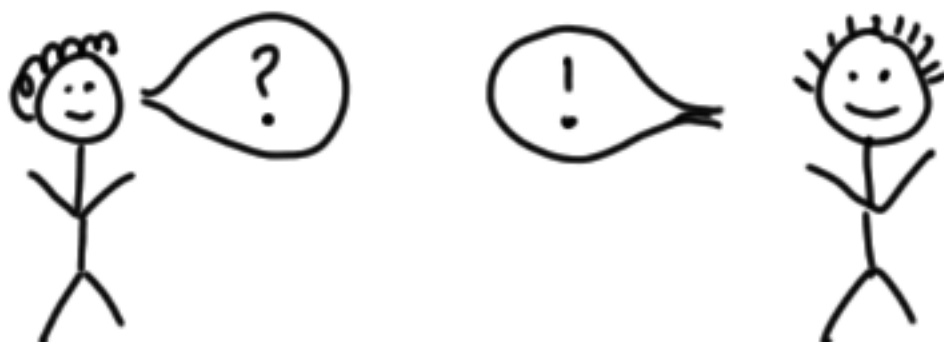
These are 5 ways I've changed how I think about  
debugging:

Remember the bug is happening  
for a logical reason. There's no magic.

Be confident I can fix it



Talk to someone



Know my debugging toolkit

before



I want to know  
\$THING but I don't  
know how to find it out

now



I know!  
I'll use  
tcpdump!

most importantly: I learned to like it ☺

before



oh no  
a bug

now



I think I'm about  
to learn something

← determined Julia

## what you'll learn

I can't teach you in 20 pages to be  
confident or to ♥ debugging. I can

show you some of my

debugging  
toolkit

the tools I reach for when I have  
a question about what my programs  
are doing. I hope at the end to have given  
you 1-2 new microscopes to use.

# 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

LINUX  
ONLY

or

OS X  
too!

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!

# 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 | recv
```

pro dstat tip:  
the -t flag prints  
the time every  
second

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

Could 300MB coming in over the network mean... a 300MB database query?!

≡ YES! ≡

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



# 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
```

... hundreds of lines ...

file  
descriptor  
↓

read  
a  
file!  
{ open("/home/bork/.config\_file") = 3  
  read(3, "the contents of the file")  
  ... hundreds of lines ...

networking!  
{ connect(5, "172.217.0.163")  
  sendto(5, "hi!!")



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](http://jvns.ca/zines)

opensnoop !

execsnoop !

eBPF !

OS X  
too!  
(kind of)

When you run

```
opensnoop -p $PID
```

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

oo strace can do this too! Just use  
strace -e open -p \$PID

... and you would be right. But strace can make your program run 10x slower.

opensnoop won't slow you down. execsnoop tells you what programs are being started.

≡ how to get it ≡

Requires: Ubuntu 16.04+  
or a ~4.4+ kernel version

Installation instructions:

[github.com/iovisor/bcc-tools](https://github.com/iovisor/bcc-tools)

This won't work on many servers today, but keep it in mind! One day you'll have servers running a newer kernel.

≡ how it works ≡

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

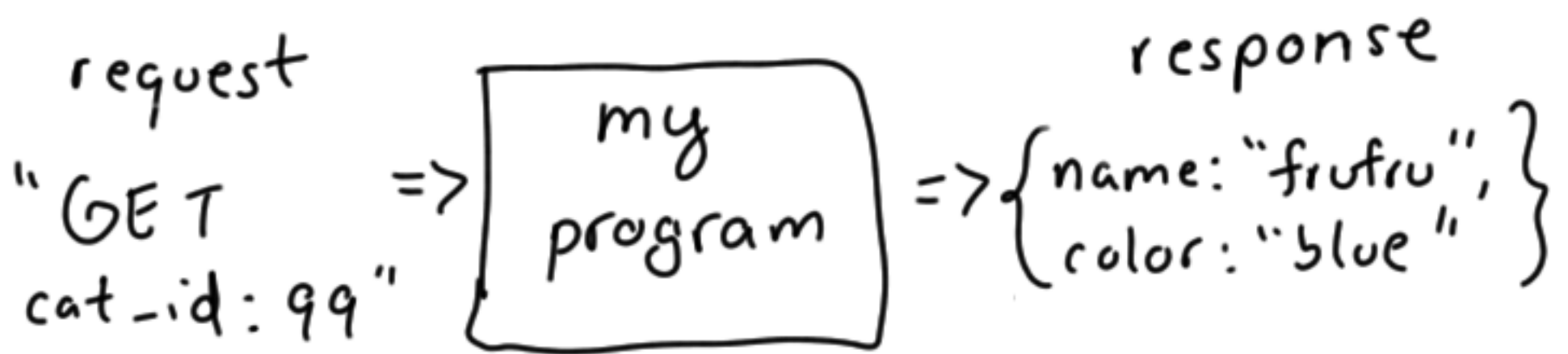
There's also an opensnoop on OS X & BSD!

That one is powered by DTrace. eBPF is super powerful. Read Brendan Gregg's blog to learn more!

# section 2::networking;

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.



Every programming language uses the same network protocols! So the network is a nice language-independent place to answer questions like:

- was the request wrong, or was it the response?
- is my server even on?
- my program is slow. Whose fault is it?

Let's go!



# netcat

handcrafted  
artisanal  
networking

OS x  
too!

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

request.txt

GET / HTTP/1.1

Host: ask.metafilter.com

User-Agent: zine

(2 newlines! important!!)

nc stands for net cat

Then:

\$ cat request.txt | nc metafilter.com 80

You should get a response back with a bunch of HTML! You can also use netcat to send huge files over a local network quickly:

step 1: (on target machine)

\$ hostname -I

192.168.2.132 ...

\$ nc -l 9931 > bigfile

step 2: (on the source)

cat bigfile |

nc 192.168.2.132 9931

↑ this listens on the port!

this sends the data ↑

# ☆ netstat ☆

OS X  
too!

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

☆ "tuna, please!" ☆

also known as

sudo netstat -tunapl

thanks to  
@icco for  
thetuna!

Here's what you'll see:

<u>proto</u>	<u>local address</u>	PID / program name
tcp	0.0.0.0:5353 port ↗	2993 / python

So! I ♥ netstat because it tells me which processes are running on which ports.

On OSX, use lsof -i -P instead.



# ngrep

OS X  
too!

grep your  
network!

ngrep is my favourite starter network spy tool! Try it right now! Run:

```
sudo ngrep -d any metafilter
```

Then go to <http://metafilter.com> in your browser. You should see matching network packets in ngrep's output! We are SPIES 😊

Recently at work I'd made a change to a client so that it sent

`{"special-id": ...}` with all its requests. I wanted to make sure it was working, so I ran

```
sudo ngrep special-id
```

I found out that everything was ok 😊

# ☺ tcpdump ☺

OS X  
too!

tcpdump is the most difficult networking tool we'll discuss here, and it took me a while to ♥ it.

I use it to save network traffic to analyze later!

```
sudo tcpdump port 8997
```

```
-w service.pcap
```

! 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 ♥

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`)



# 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](http://metafilter.com) in your browser. (or [jvns.ca](http://jvns.ca)!). Then press Ctrl+C to stop tcpdump. Now we have a pcap!

```
wireshark http.pcap
```

Explore the Wireshark interface!

Questions you can try to answer:

- ① What HTTP headers did your browser send to [metafilter.com](http://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](http://metafilter.com)'s server?

(hint: search 'ip.dst == 54.186.13.33')

IP address from  
ping [metafilter.com](http://metafilter.com)

LINUX  
ONLY

## section 3: CPU + perf

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

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

(in general, my aim in this zine is to showcase tools that I don't think get enough ♥♥♥)

Some things I didn't have space for in this section but I want to mention anyway:

- valgrind
- the Java ecosystem's fantastic tools (jstack, VisualVM, Mission Control, YourKit) which your language is probably jealous of
- ftrace (for Linux kernel perf problems)
- eBPF

# ♥ perf ♥

perf is not simple or elegant. It is a weird multitool that does a bunch of different, very useful things. First! It's a

sampling profiler

Try running:

```
$ sudo perf record python
```

(press Ctrl+C after 2 seconds)

saves a file "perf.data"

This records, every few milliseconds, what the python process is doing. Let's see the results!

```
$ 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!

just C functions

If you're a Python/Ruby/Java/Node programmer, you might be getting antsy.

"I want to know which Ruby function is running! Not the C function!

Stick with me though. I get you.

works everywhere ♥

perf can be installed on pretty much any Linux machine. The exact features it has will depend a little on your kernel version.

# perf is for everyone

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?

```
$ sudo perf top
```

process	PID	%	function
ruby	1957	77	match-at

perf top doesn't always help. Far from it. But it's an easy tool to try, and it's awesome when it does help.

★ "Ruby's internal" ★  
"regexp matching function"

... especially Java and node devs! 📢

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



# spy on your CPU!

Your CPU has a small cache on it (the L1 cache) that it can access in  $\sim 0.5$  nanoseconds! faster than RAM!

200 times

\* tip:  
google "Latency Numbers every programmer should know"!

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



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

perf stat!



how to use it

```
perf stat -e L1-dcache-load-misses ;  
ls
```

This runs 'ls' and prints a report at the end.

how it works

Your CPU can keep all kinds of counters about what it's doing. `perf stat` asks it to count things + then collects the result.

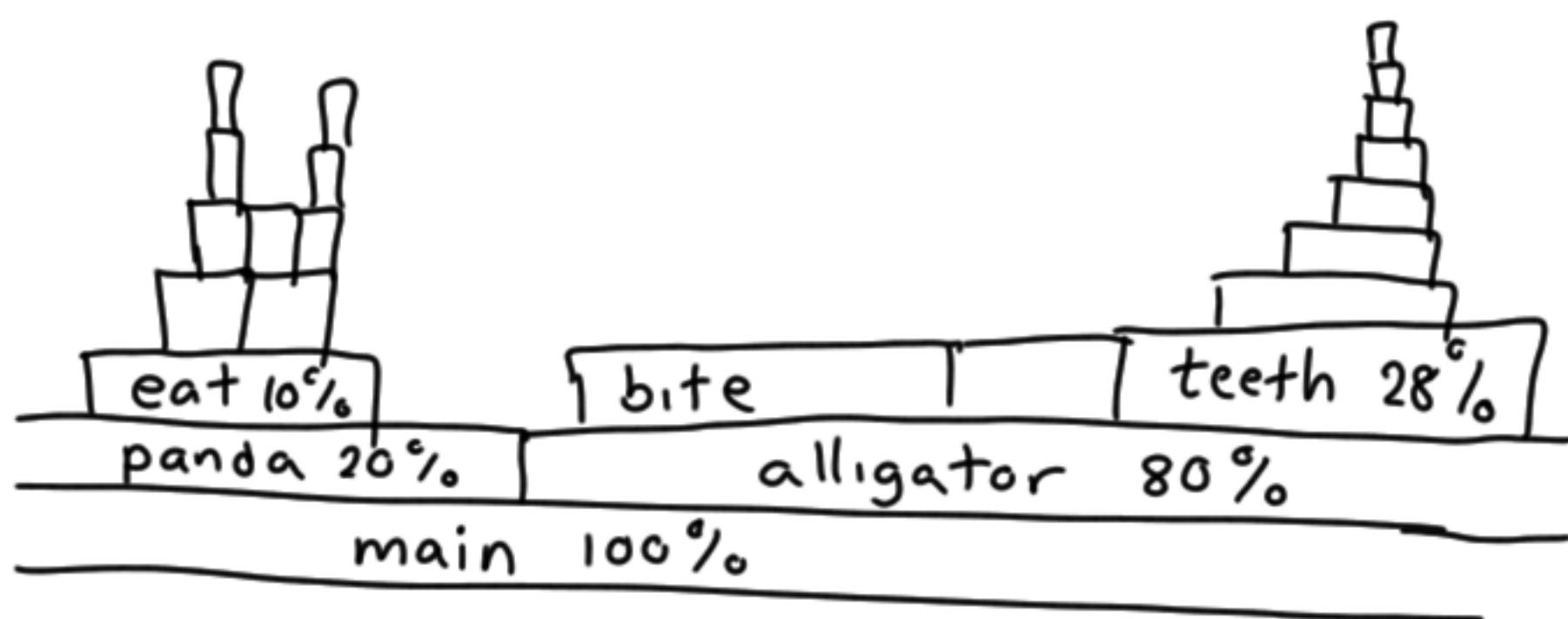


Hardware is cool. knowing more about how your hardware works can really pay off ★

# flame graphs

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

Here's what they look like:



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

You can construct them from perf recordings (look up "Brendan Gregg flamegraph" for how), but lots of other unrelated tools can produce them too. I ♥ them.



I hope you learned something new.

Thanks for reading ♡

Thanks to my partner Kamal for endless reviews, to the amazing Monica Dinculescu (@notwaldorf) for the cover art, and many others.

If you want to know more - my site has a lot ([jvns.ca](http://jvns.ca)) and [brendangregg.com](http://brendangregg.com) does too.

But really you just need to experiment. Try these tools everywhere. See where they help and where they don't. It takes a lot of practice to use these tools to debug real problems.

I've been learning them for 2 years, and I've gotten pretty far, but there's a long way to go. It's really fun ☺☺☺

liked this?

You can print more!  
for free ♡

≡ <http://jvns.ca/zines> ≡