Structured Log Pipes: A Modern Take on UNIX Philosophy

Published: December 2025

Introduction

The UNIX philosophy has guided software design for decades: write programs that do one thing well, work together, and handle text streams. But what if we could improve on this foundation?

This post explores how structured logging can modernize the pipe paradigm while maintaining its core benefits.

The Problem with Text

Traditional UNIX pipes rely on unstructured text. Consider this simple pipeline:

```
ps aux | grep python | awk '{print $2}' | xargs kill
```

This works, but it's fragile. What if a username contains "python"? What if the column layout changes?

A Mathematical View

Let's formalize this. A traditional pipe is a composition of functions:

$$f_n \circ f_{n-1} \circ \ldots \circ f_1 : \text{Text} \to \text{Text}$$

Where each f_i must parse and serialize text. The error rate grows with each composition:

$$P(\text{error}) = 1 - \prod_{i=1}^{n}(1 - p_i)$$

Where p_i is the error rate of parser i.

The Structured Alternative

Instead, we use JSON Lines (JSONL) as our universal interface:

```python
#!/usr/bin/env python3
import json
import sys

for line in sys.stdin:
    event = json.loads(line)
    # Process structured data
    event['processed'] = True
    print(json.dumps(event))
```

This gives us:

- Type safety through schemas
- Composability through structured data
- Debuggability through readable formats

Conclusion

By replacing text with structured data, we preserve UNIX philosophy while fixing its core limitation. The future of composable software is structured.