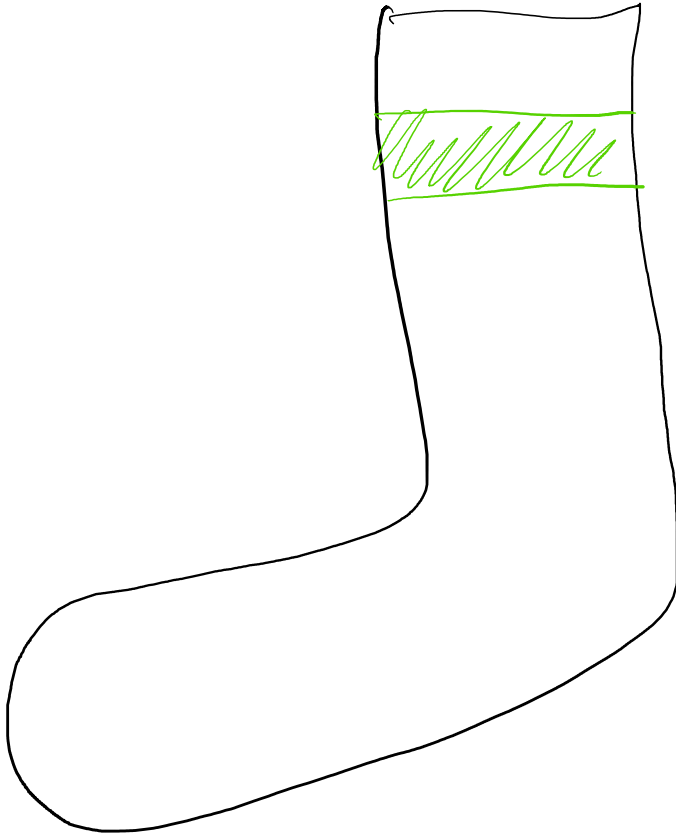
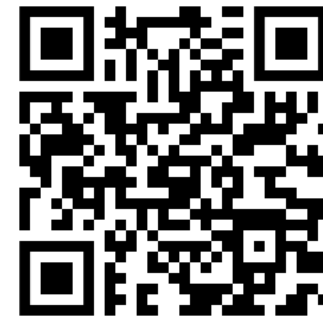


Warmup exercise



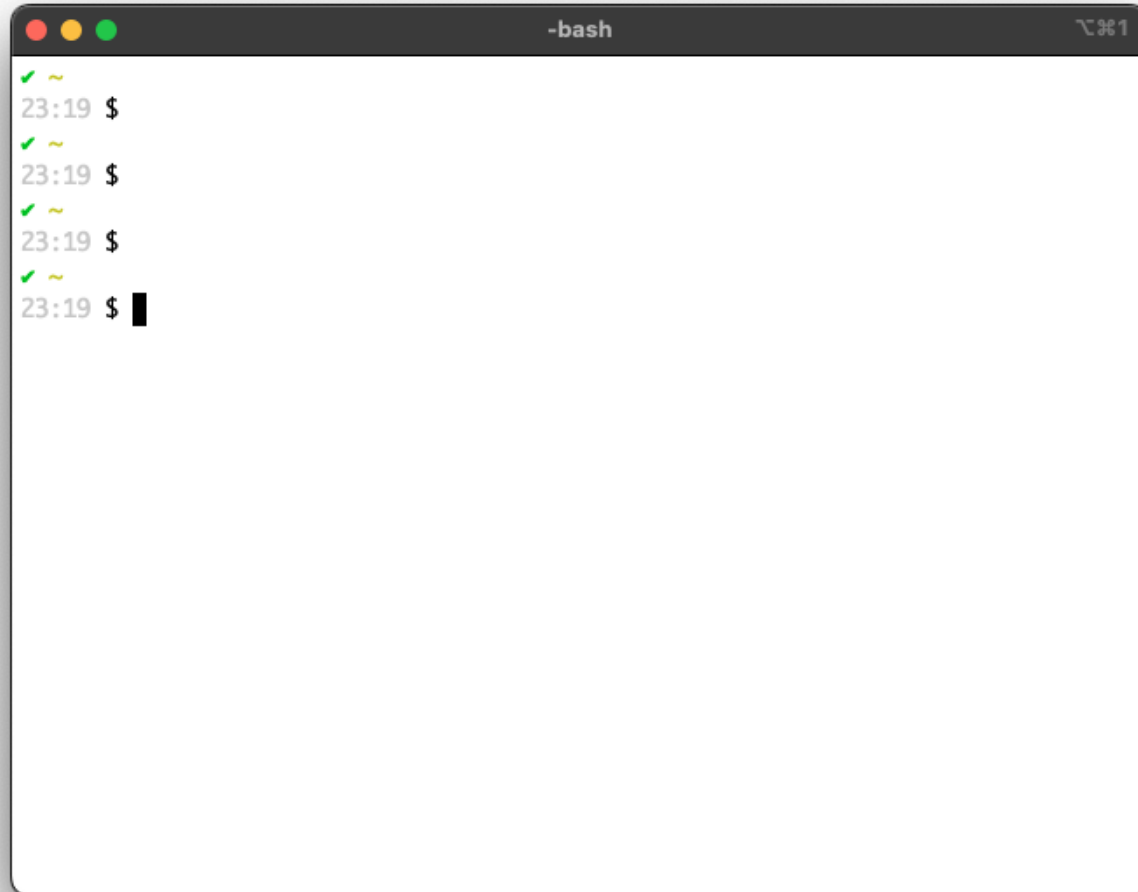
On the left we see:

1. A green sock
2. A white sock
3. A white sock with a green stripe



<https://github.com/ngs-lang/presentations>

Warmup exercise

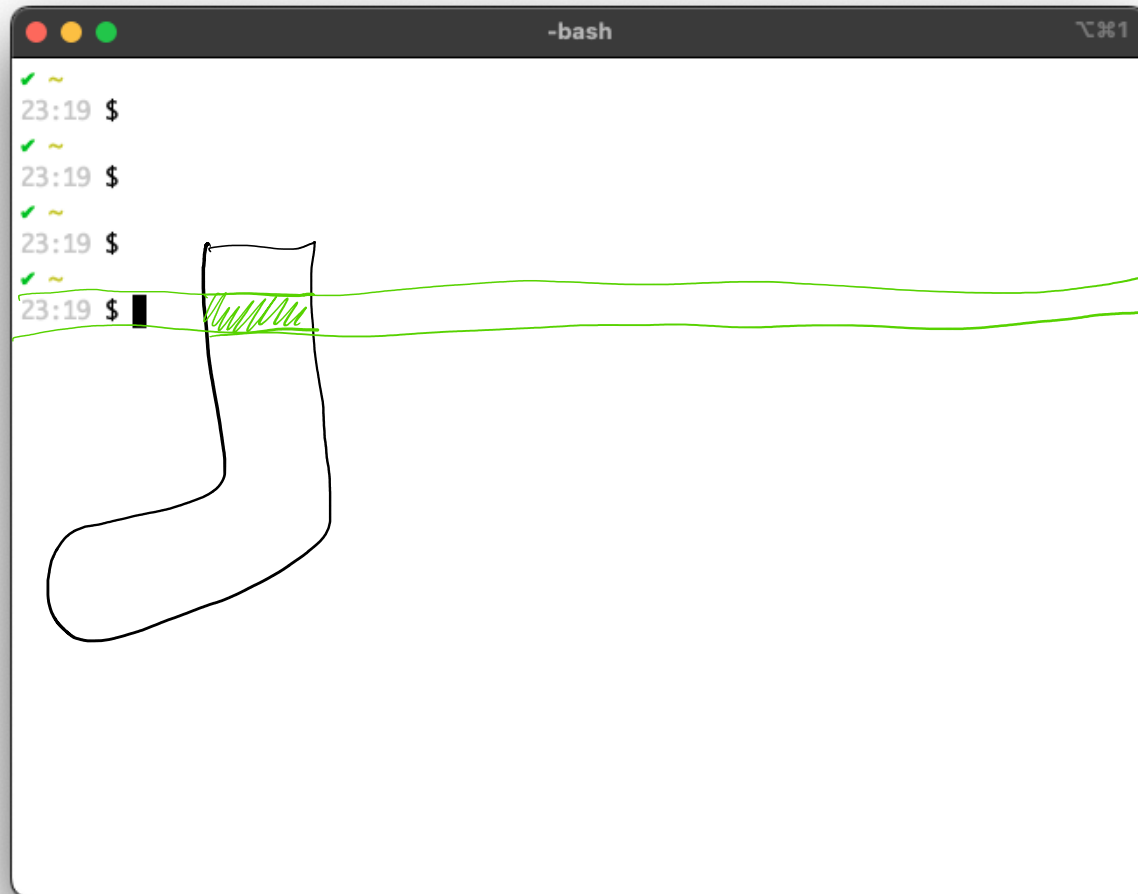
A terminal window with a dark title bar containing three colored window control buttons (red, yellow, green) on the left, the text "-bash" in the center, and a cursor icon on the right. The terminal content shows four lines, each starting with a green checkmark icon, a yellow tilde icon, and the text "23:19 \$". The fourth line has a black cursor block at the end.

```
✓ ~ 23:19 $  
✓ ~ 23:19 $  
✓ ~ 23:19 $  
✓ ~ 23:19 $ █
```

On the left we see:

1. Interactive shell
2. Non-interactive shell
3. Slightly interactive shell

Warmup exercise



On the left we see:

1. Interactive shell
2. Non-interactive shell
3. Slightly interactive shell

Hint: is that a green sock?

Hint: interactive shell vs
interactive website

Unix Shell

We can do better now

Command Line Interface is not the pinnacle of UI today

About Me

- [Ilya Sher](#)
- Bash user – since around 1996
- DevOps & Software – professionally since 2001
- Working on Next Generation Shell – since 2013



CLI – how we got here?

Very old communication paradigm:

send text – receive text

CLI – how we got here?

- Telegraph
- Teleprinter (aka teletype, aka TTY)
- Video Display Unit



CLI – how we got here?

- Telegraph
- Teleprinter (aka teletype, aka TTY)
- Video Display Unit



CLI – how we got here?

- Telegraph
- Teleprinter (aka teletype, aka TTY) - with computer
- Video Display Unit



CLI – how we got here?

- Telegraph
- Teleprinter (aka teletype, aka TTY)
- Video Display Unit



Turning Point

Technological breakthrough – VT52 with cursor movement support

Allowed full screen user interfaces

1. 1971 - 1975
2. 1976 - 1980
3. 1981 - 1985



[ClickRick](#), [CC BY-SA 3.0](#), via Wikimedia Commons

Turning Point

Technological breakthrough – VT52 with cursor movement support

Allowed full screen user interfaces

1. 1971 - 1975 (answer: 74/75)
2. 1976 - 1980
3. 1981 - 1985



[ClickRick](#), [CC BY-SA 3.0](#), via Wikimedia Commons

Turning Point – Text Editing



```
-bash
10:04 $ cat 1.txt
1
2
3
10:04 $ ed 1.txt
6
^,$s/2/2x/
w 2.txt
7
q
10:05 $ cat 2.txt
1
2x
3
10:05 $
```

A terminal window titled "-bash" showing a sequence of commands. The first command is "cat 1.txt", which outputs lines 1, 2, and 3. The second command is "ed 1.txt", which enters the ed editor. Inside ed, the user enters "6" (line number), "^,\$s/2/2x/" (substitution command), "w 2.txt" (write command), "7" (line number), and "q" (quit command). A red rectangle highlights the "ed 1.txt" command and its subsequent input. After exiting ed, the user runs "cat 2.txt", which outputs lines 1, 2x, and 3. The prompt returns to "10:05 \$".

Before: CLI text editing

Text editing goes full screen instead of CLI.

[Bill Joy](#) releases [vi](#) in 1976



```
vi
1
2
3
~
~
~
"1.txt" line 1 of 3 --33%-- col 1
```

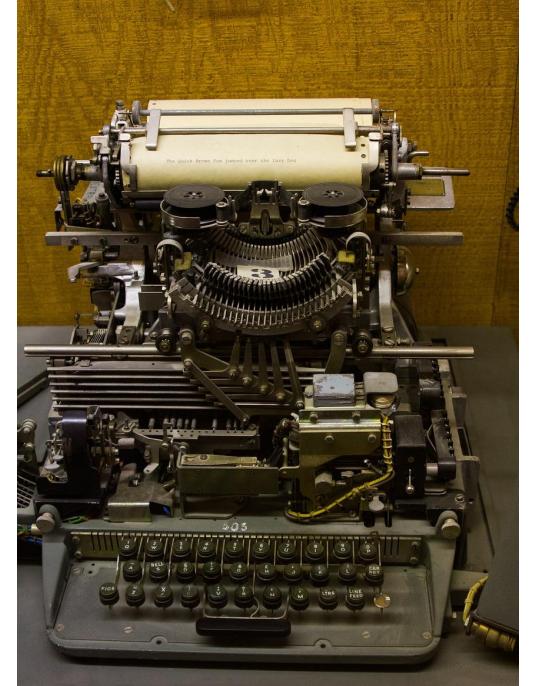
A terminal window titled "vi" showing the vi editor in full-screen mode. The first three lines are numbered 1, 2, and 3. Below them are three tilde (~) characters. At the bottom, a status line reads: "1.txt" line 1 of 3 --33%-- col 1.

After: full screen text editing

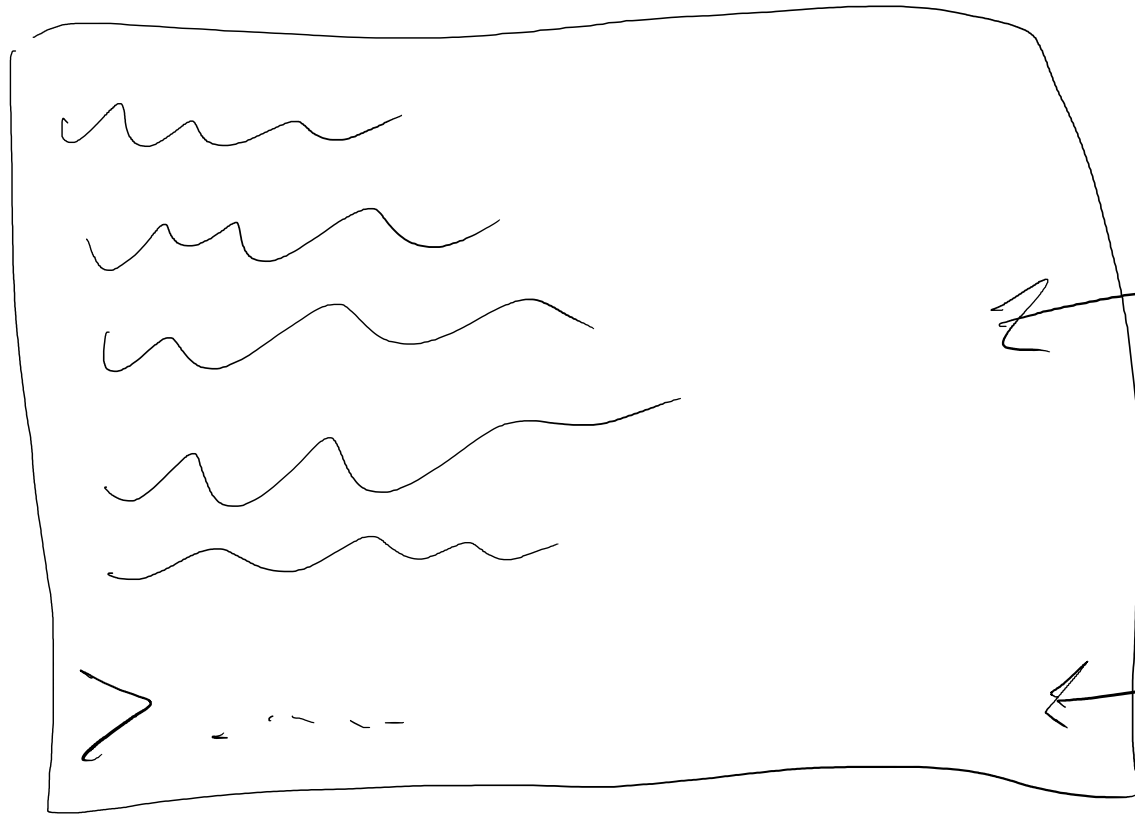
Turning Point - Shells

Not a turning point

Shells never responded (except for command line completion)

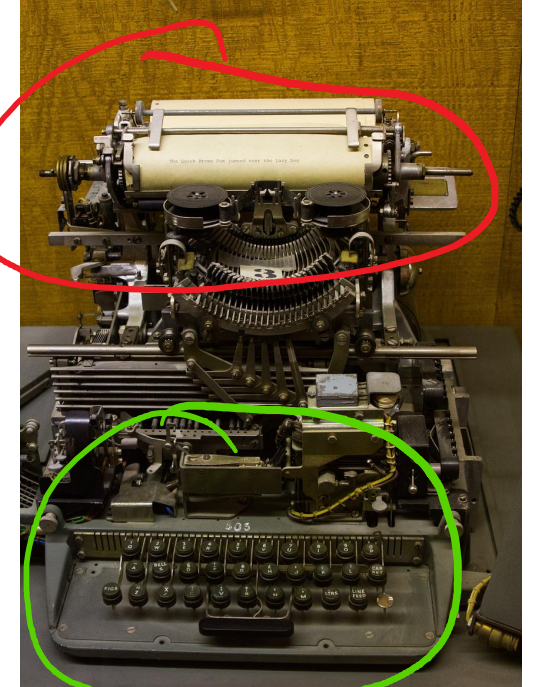


User Interface - “Interactive Shell”

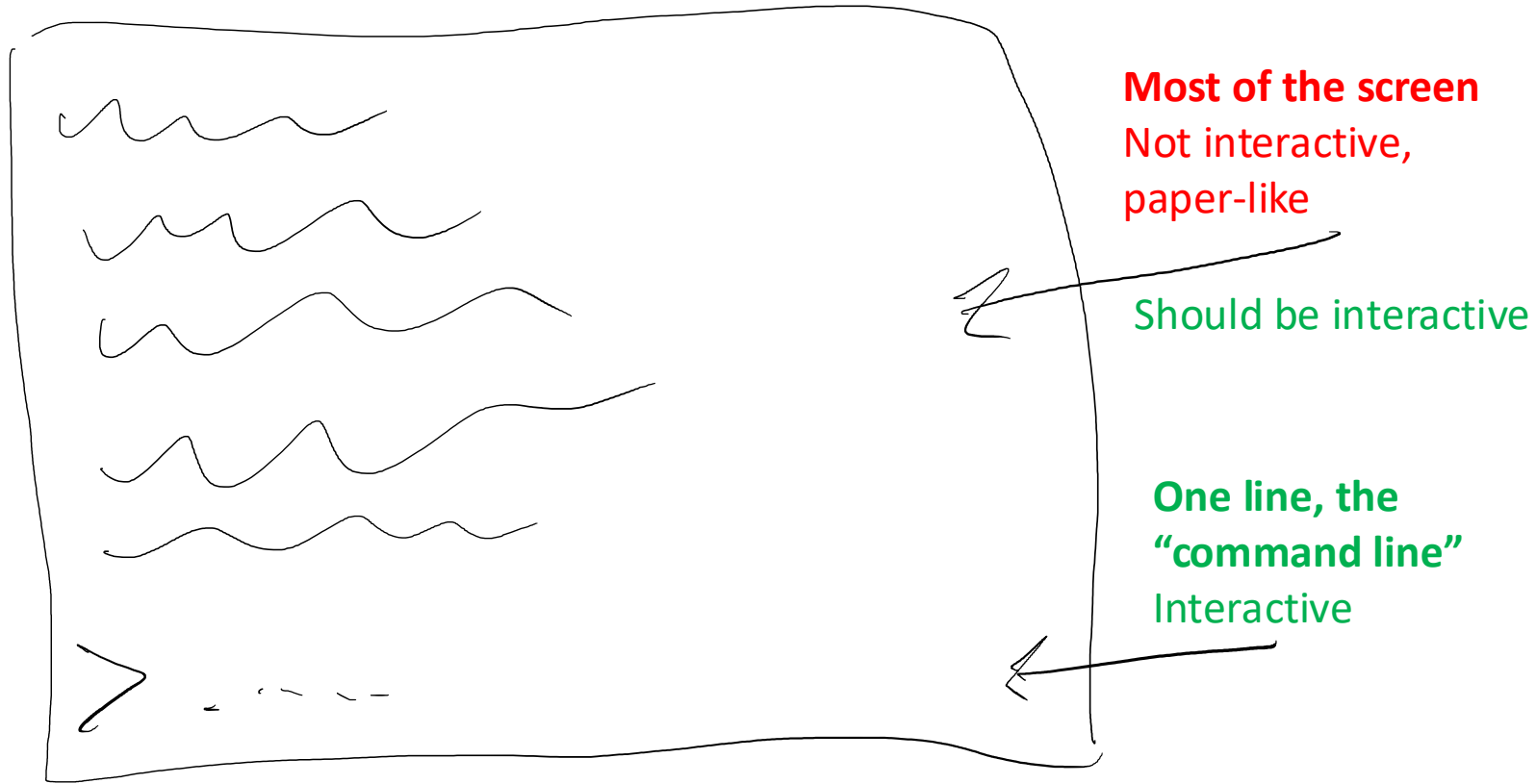


Most of the screen
Not interactive,
paper-like

One line, the
“command line”
Interactive



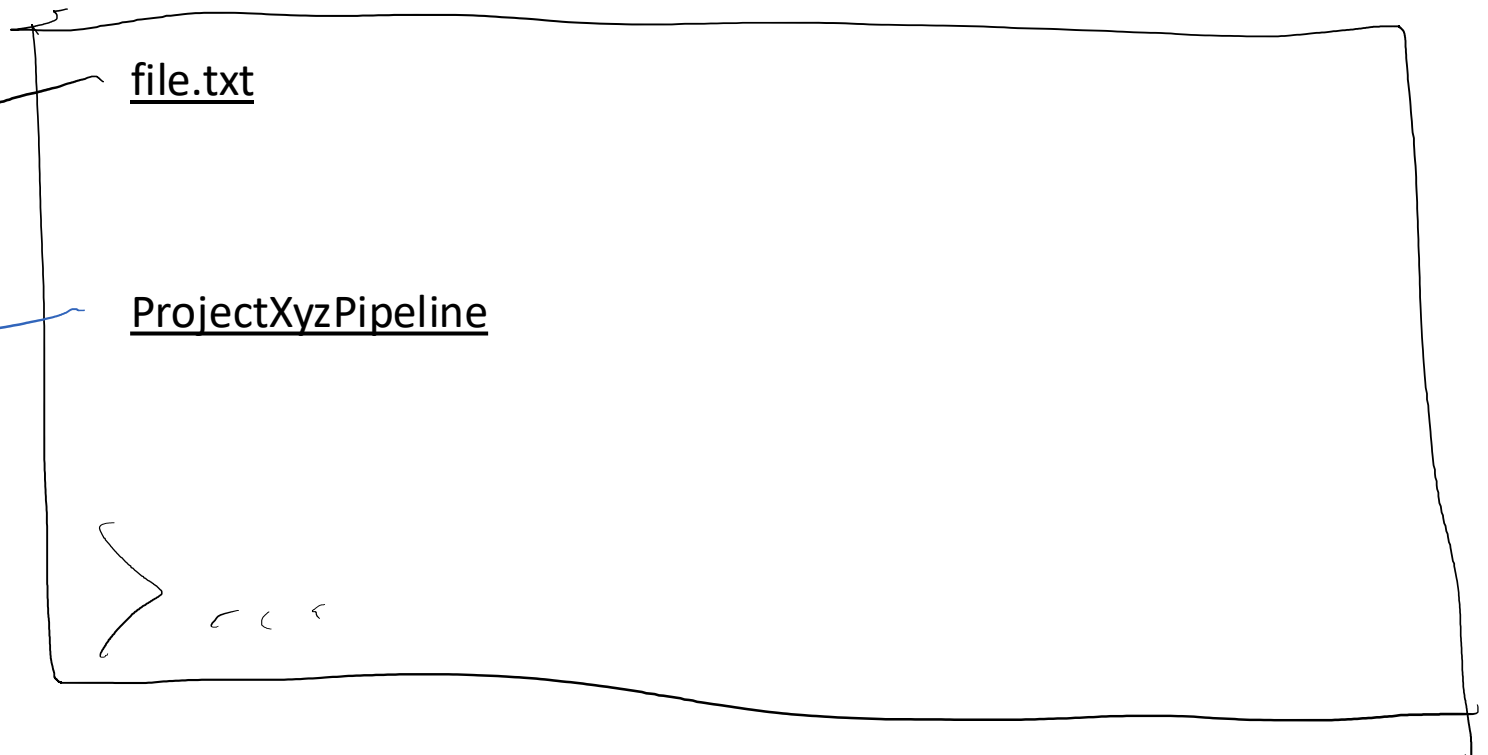
User Interface - “Interactive Shell”



[ClickRick](#), [CC BY-SA 3.0](#), via Wikimedia Commons

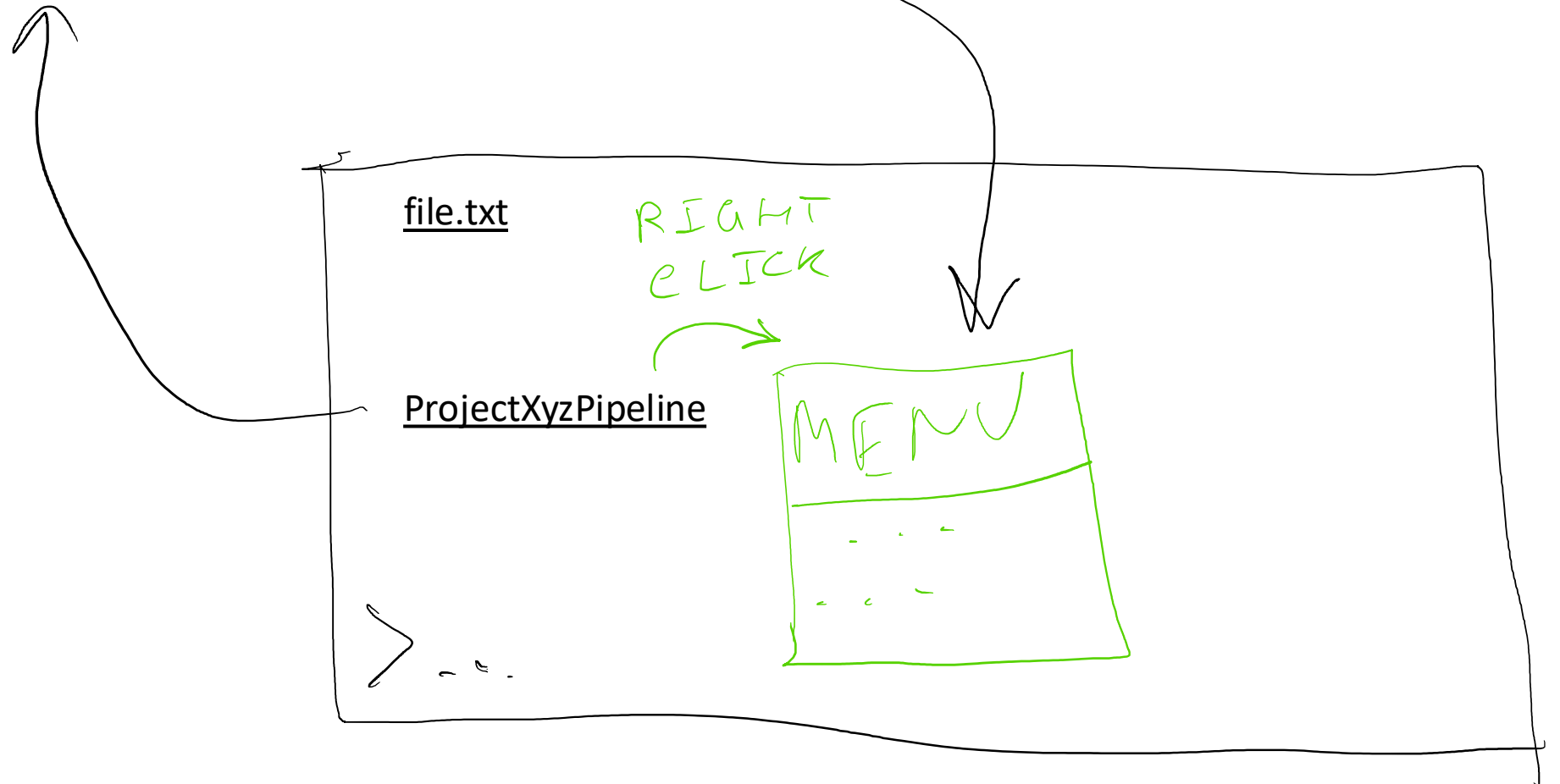
User Interface – Objects on the Screen

- type = file, id = /Users/blah/file.txt , name = file.txt
- type = AWS::CodePipeline, id = ..., name = ProjectXyzPipeline

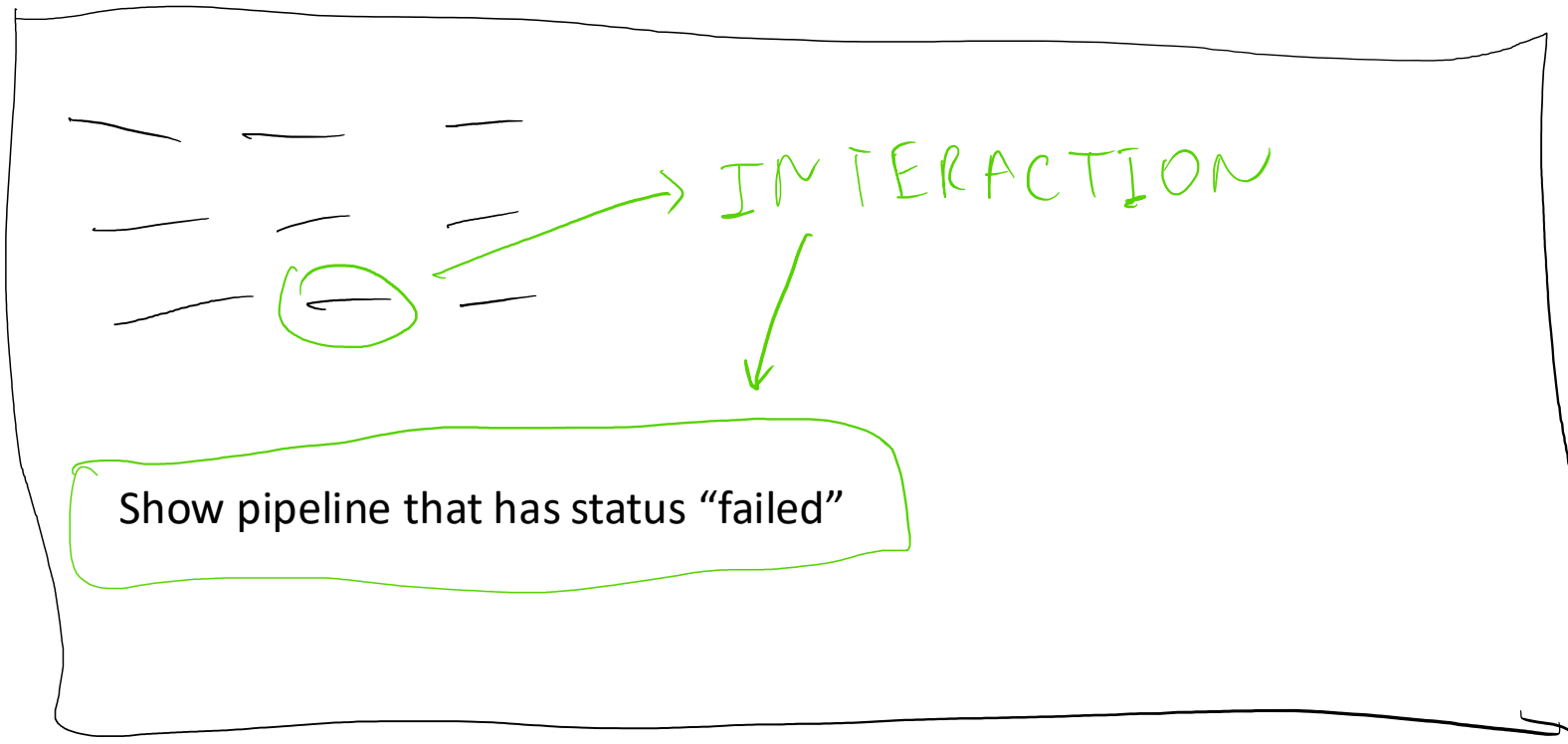


User Interface – Plugins

- Plugin Blah: handles AWS::CodePipeline



User Interface – Record / Replay



Capture the semantics
of the interaction

More powerful than just
history: i-123

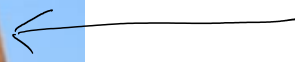
Display the recording

Let edit the recording



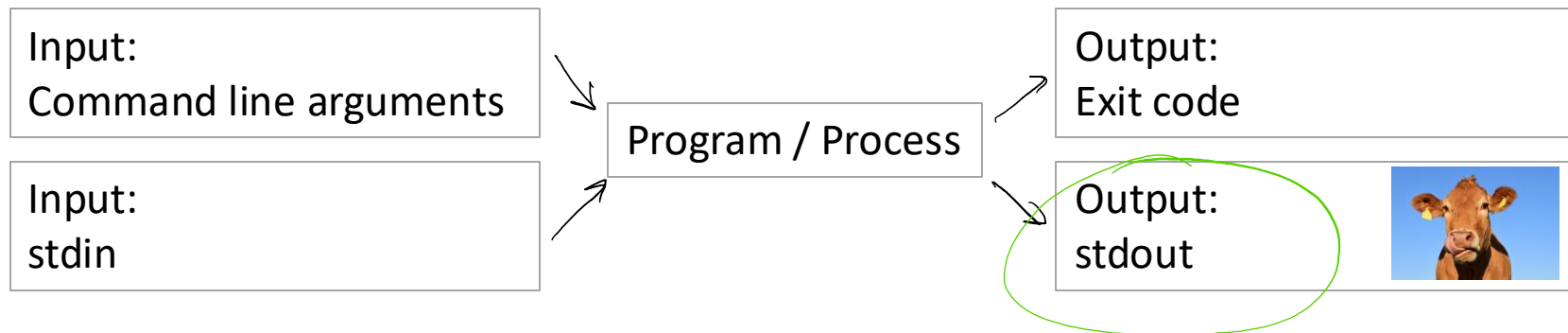
User Interface – How?

SACRED
COW



User Interface – How?

Understanding Output



"Shell is not supposed to get into semantics"

User Interface – How?

Understanding Output

Semantics – already implemented

Command line completion

Input: ✓
Command line arguments

Input:
stdin

Program / Process

Semantics – already implemented

Basic error handling

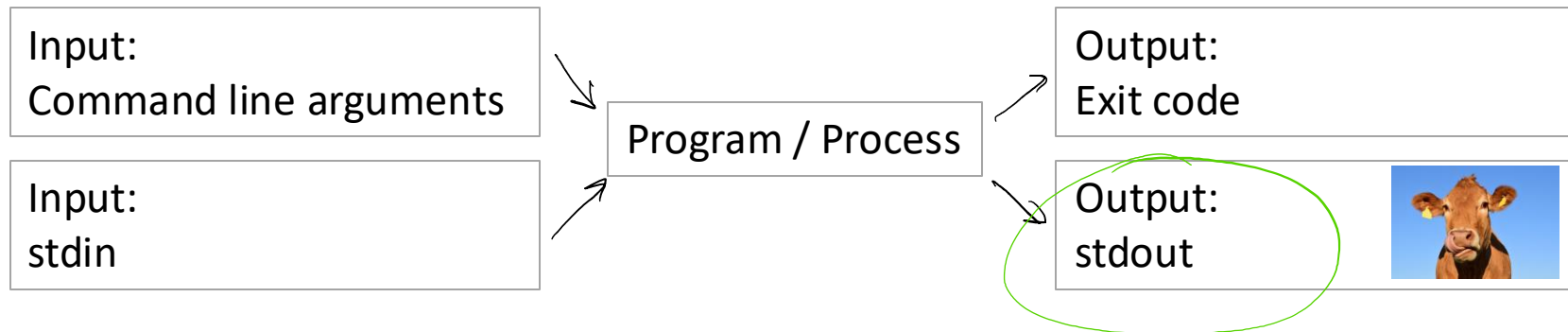
Output: ✓
Exit code

Output: 
stdout

“Shell is not supposed to get into semantics”

User Interface – How?

Understanding Output



"It's too much work"

User Interface – How?

SAME SCALE


Implemented in shell for many commands

Command line completion

Input:
Command line arguments

Input:
stdin

Program

A terminal window titled "-bash" with a cursor at the bottom. The prompt is "10:26 \$". The command entered is "jc ifconfig | jq . | head". The output is a JSON array containing one object: [{"name": "lo0", "flags": 8049, "state": ["UP", "LOOPBACK", "RUNNING", "MULTICAST"]}], followed by a comma and a closing bracket. The prompt is "11:04 \$".

```
10:26 $ jc ifconfig | jq . | head
[
  {
    "name": "lo0",
    "flags": 8049,
    "state": [
      "UP",
      "LOOPBACK",
      "RUNNING",
      "MULTICAST"
    ],
  },
],
11:04 $
```

stdout

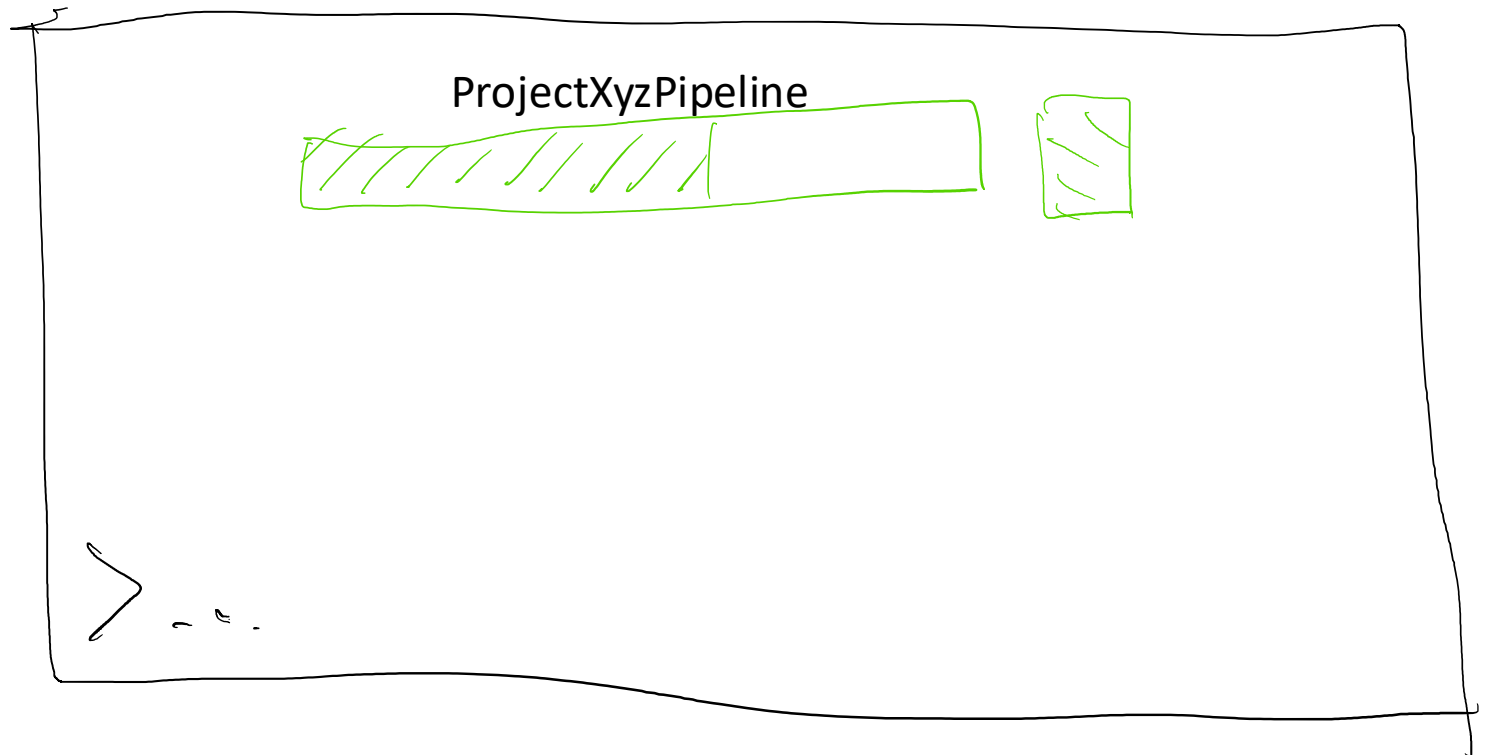
"It's too much work"

Understanding Output

User Interface – Objects of Interest

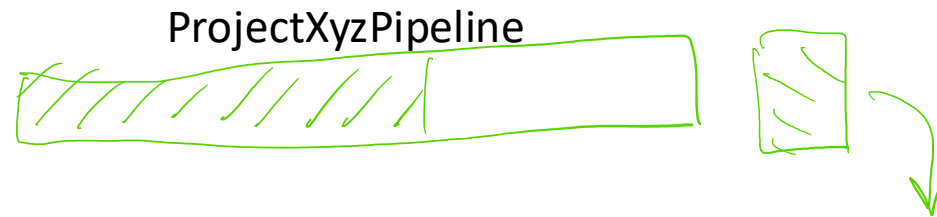
- Information displayed automatically – ex: CI/CD progress

Would be very awkward if implemented in other shells



User Interface – Objects of Interest

- Information displayed automatically – ex: CI/CD progress



Sensible default rules like:

- Show CI/CD runs of pipelines that I created
- Show CI/CD runs that were triggered by my commit

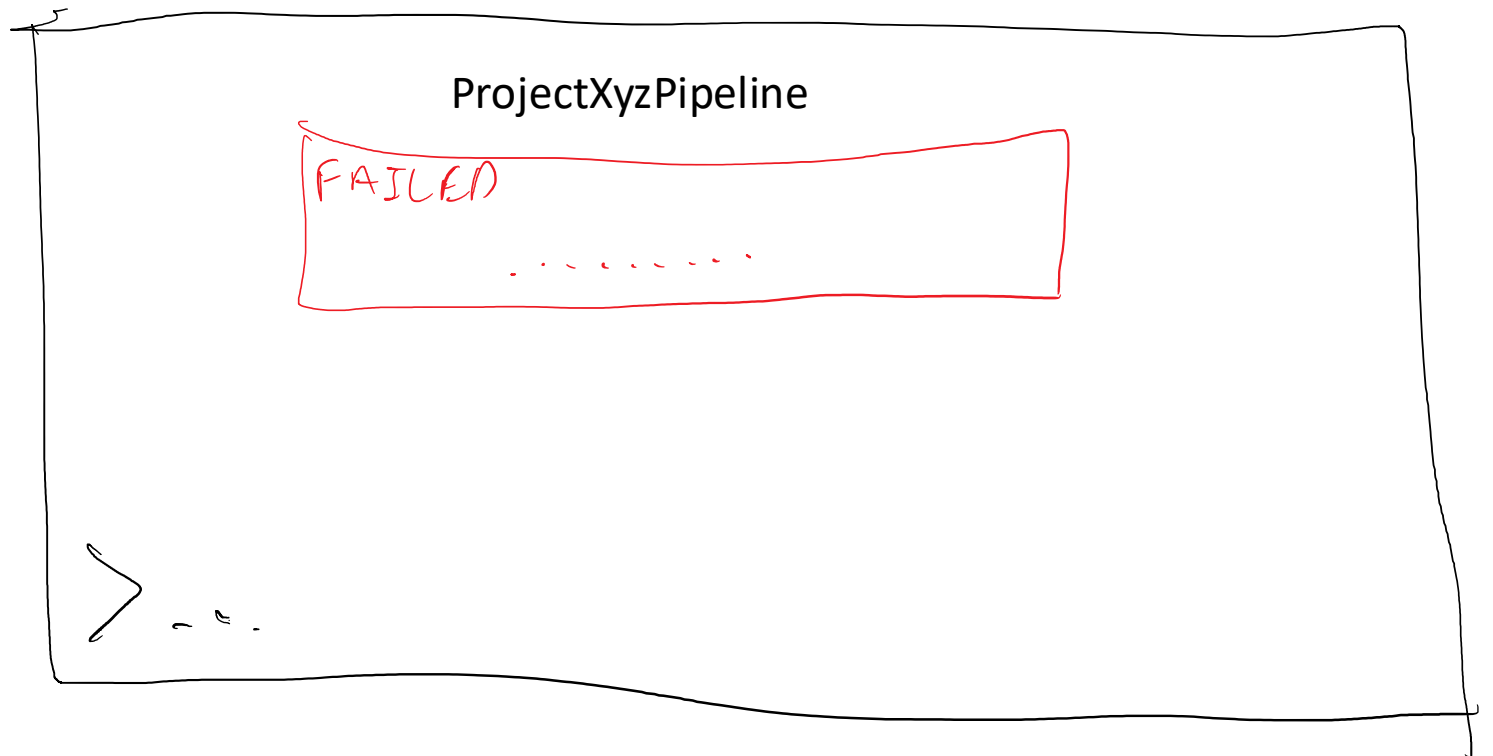
Same but for last errors instead of runs

This information is shown to you because it matches rule “blah”

- Edit rule
- Delete rule
- Exclude this object

User Interface – Objects of Interest

- Information displayed automatically – last error



UI Vision Summary

Semantics - the more a program "understands" the more powerful it can be.

- Interactive objects on the screen
- Capture as much of interaction semantics as you can.

Thanks!

- Next Generation Shell - <https://ngs-lang.org/>
- Ilya Sher - <https://ilya-sher.org/>

What's next?

- Talk to me / join Discord / give feedback
- Try NGS
- Spread the word
- Help designing the UI

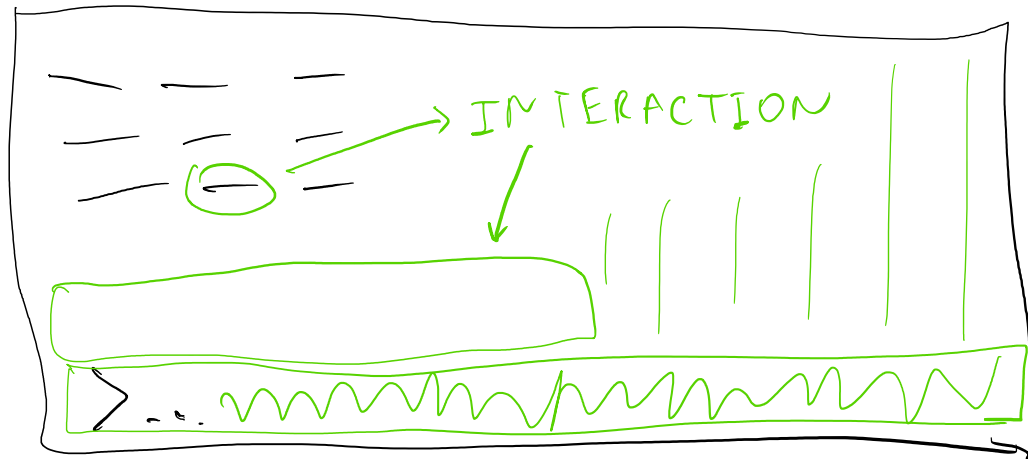


<https://github.com/ngs-lang/presentations>

Bonus Slides

Interfaces

	Web	CLI
Pros	Easy	Powerful
Cons	<ul style="list-style-type: none">• No history• Non-repeatable• Limited	<ul style="list-style-type: none">• Requires programming to achieve almost anything (or it won't be reproducible)• Can't interact with the output



Green sock
... with green stripe



Cocona vs NGS

«Micro-framework for .NET
Core **console application**»

```
using Cocona;  
CoconaApp.Run((string name) => {  
    Console.WriteLine($"Hello {name}");  
})
```

```
F main(name:Str) {  
    echo("Hello ${name}")  
}
```


Cocona vs NGS

```
using Cocona;  
CoconaApp.Run((string name) => {  
    Console.WriteLine($"Hello {name}");  
})
```

```
F main(name:Str) {  
    echo("Hello ${name}")  
}
```

General-purpose
programming language with
domain-specific facilities

Cocona vs NGS

```
var app = CoconaApp.Create();
```

```
app.AddCommand("list", () => { ... });
```

```
app.AddCommand("add", () => { ... });
```

```
app.AddCommand("delete", () => { ... });
```

```
app.Run();
```

Cocona vs NGS

```
ns {  
    F list(...) ...  
    F add(...) ...  
    F delete(...) ...  
}
```

also available as `require(...)::list(...)`

NGS Architecture

Component	Implemented in
UI frontend – thin plugins	JS / NGS
UI backend	NGS
NGS stdlib	NGS
NGS language core	C

jc with NGS

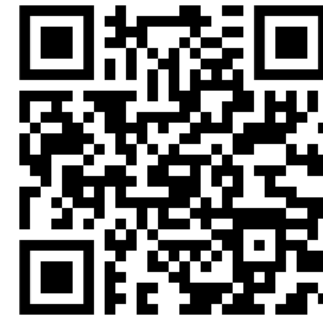
```
-bash
10:24 $ jc ifconfig | jq . | head
[
  {
    "name": "lo0",
    "flags": 8049,
    "state": [
      "UP",
      "LOOPBACK",
      "RUNNING",
      "MULTICAST"
    ],
  },
]
✓ ~
10:25 $ jc ifconfig | ngs -ppj '_.filter({"name": /^en/, "status": "active"}).name'
[
  "en8"
]
✓ ~
10:25 $ jc ifconfig | ngs -ppj '_.filter({"state": Present("LOOPBACK")}).name'
[
  "lo0"
]
✓ ~
10:26 $
```

Thanks!

- Next Generation Shell - <https://ngs-lang.org/>
- Ilya Sher - <https://ilya-sher.org/>

What's next?

- Talk to me / join Discord / give feedback
- Try NGS
- Spread the word
- Help designing the UI



<https://github.com/ngs-lang/presentations>