NuShell

Superglue for your OS

Willem Vanhulle

SysGhent 

Wednesday, Dec 3, 2025

Introduction

What does the following Bash code do?

```shell
find . -type f -name "*.log" -mtime +30 -exec rm {} \;
```

What does the following Bash code do?

```shell
find . -type f -name "*.log" -mtime +30 -exec rm {} \;
```

Nu:

```shell
ls **/*.log | where modified < (date now) - 30day | rm
```

Improvements:

- Decomposes the problem with pipes
- Does not require `find` flags
- Built-in glob, duration and date type

What does Nu in NuShell stand for?

What does the following Bash code do?

```shell
find . -type f -name "*.log" -mtime +30 -exec rm {} \;
```

Nu:

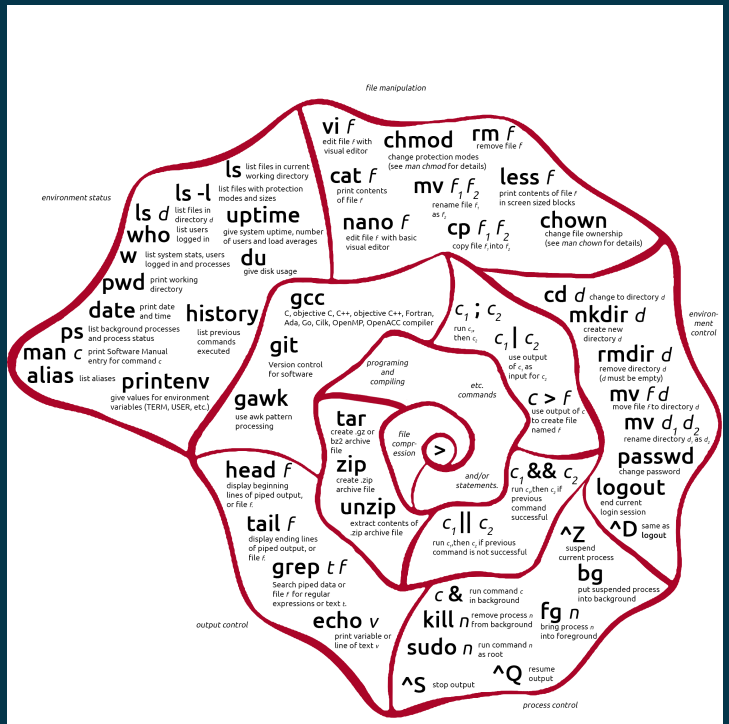
```shell
ls **/*.log | where modified < (date now) - 30day | rm
```

Improvements:

- Decomposes the problem with pipes
- Does not require `find` flags
- Built-in glob, duration and date type

What does Nu in NuShell stand for?

New shell.



LINUX SHELL COMMANDS

Linuxus Bashus c.

Based off a marvelous Unix Poster, if you own the rights to this poster Please Contact me:

Remixed by JaceTheSaltSculptor

Linux is a trademark of Linus Torvalds

Prerequisites

Try it out out yourself

No installation: https://www.nushell.sh/demo/

Install locally: https://www.nushell.sh/book/installation.html

- Download binary: github.com/nushell/nushell/releases
- Rust:
  - Install `rustup` from https://rustup.rs/
  - Add Cargo bin to your PATH if not done automatically
  - `cargo install nu`
- Mac: `brew install nushell`
- Windows (winget): `winget install nushell`
- Windows (chocolatey): `choco install nushell`

Linux:

- Debian: `apt install rustup`
- Nix: `nix-shell -p nushell`
- Snap: `sudo snap install nushell --classic`

Exercises

Have a look at the `*.nu` files in this repo.

To run an exercise:

```shell
workshop.nu # With shebang
nu workshop.nu
```

To pipe in data from Bash:

```shell
cat somefile.txt | exercise.nu # With shebang
cat somefile.txt | nu exercise.nu
```

Piping within NuShell:

```shell
open somefile.txt | exercise.nu
```

Basics

Commands output tables

Shell

```
ls
# => ╭─────┬──────────────────────┬──────┬──────────┬──────────────╮
# => │  #  │         name         │ type │   size   │   modified   │
# => ├─────┼──────────────────────┼──────┼──────────┼──────────────┤
# => │   0 │ CITATION.cff         │ file │    812 B │ 2 months ago │
# => │   1 │ CODE_OF_CONDUCT.md   │ file │  3.4 KiB │ 9 months ago │
# => │   2 │ CONTRIBUTING.md      │ file │ 11.0 KiB │ 5 months ago │
# => │   3 │ Cargo.lock           │ file │ 194.9 KiB│ 15 hours ago │
# => │   4 │ Cargo.toml           │ file │  9.2 KiB │ 15 hours ago │
# => │   5 │ Cross.toml           │ file │    666 B │ 6 months ago │
# => │   6 │ LICENSE              │ file │  1.1 KiB │ 9 months ago │
# => │   7 │ README.md            │ file │ 12.0 KiB │ 15 hours ago │
# => ...
```

Sort output by column


```
ls | sort-by size | reverse
```

Shell

```
# => ╭────┬──────────────┬──────┬──────────┬──────────────╮
# => │ #  │     name     │ type │   size   │   modified   │
# => ├────┼──────────────┼──────┼──────────┼──────────────┤
# => │ 0  │ Cargo.lock   │ file │ 194.9 KiB │ 15 hours ago │
# => │ 1  │ toolkit.nu   │ file │  20.0 KiB │ 15 hours ago │
# => │ 2  │ README.md    │ file │  12.0 KiB │ 15 hours ago │
# => │ 3  │ CONTRIBUTING.md │ file │  11.0 KiB │ 5 months ago │
# => │ 4  │ ...          │ ...  │ ...       │ ...          │
# => │ 5  │ LICENSE      │ file │   1.1 KiB │ 9 months ago │
# => │ 6  │ CITATION.cff │ file │     812 B │ 2 months ago │
# => │ 7  │ Cross.toml   │ file │     666 B │ 6 months ago │
# => │ 8  │ typos.toml   │ file │     513 B │ 2 months ago │
# => ╰────┴──────────────┴──────┴──────────┴──────────────╯
```

Filtering output

Shell

```
ls | where size > 10kb
# => ╭───┬──────────────────┬──────┬──────────┬──────────────╮
# => │ # │       name       │ type │   size   │   modified   │
# => ├───┼──────────────────┼──────┼──────────┼──────────────┤
# => │ 0 │ CONTRIBUTING.md  │ file │  11.0 KiB │ 5 months ago │
# => │ 1 │ Cargo.lock       │ file │ 194.6 KiB │ 2 minutes ago │
# => │ 2 │ README.md        │ file │  12.0 KiB │ 16 hours ago │
# => │ 3 │ toolkit.nu       │ file │  20.0 KiB │ 16 hours ago │
# => ╰───┴──────────────────┴──────┴──────────┴──────────────╯
```

Processes

Shell

```
ps
# => ╭───┬──────┬──────┬──────────────────┬──────────┬──────┬──────────┬─────────╮
# => │ # │ pid  │ ppid │       name       │  status  │ cpu  │   mem    │ virtual │
# => ├───┼──────┼──────┼──────────────────┼──────────┼──────┼──────────┼─────────┤
# => │ 0 │    1 │    0 │ init(void)       │ Sleeping │ 0.00 │  1.2 MiB │ 2.2 MiB │
# => │ 1 │    8 │    1 │ init             │ Sleeping │ 0.00 │ 124.0 KiB │ 2.3 MiB │
# => │ 2 │ 6565 │    1 │ SessionLeader    │ Sleeping │ 0.00 │ 108.0 KiB │ 2.2 MiB │
# => │ 3 │ 6566 │ 6565 │ Relay(6567)      │ Sleeping │ 0.00 │ 116.0 KiB │ 2.2 MiB │
# => │ 4 │ 6567 │ 6566 │ nu               │ Running  │ 0.00 │  28.4 MiB │ 1.1 GiB │
# => ╰───┴──────┴──────┴──────────────────┴──────────┴──────┴──────────┴─────────╯
```

Running processes

Shell

```
ps | where status == Running
# => ╭───┬──────┬──────┬──────┬─────────┬──────┬──────────┬─────────╮
# => │ # │ pid  │ ppid │ name │ status  │ cpu  │   mem    │ virtual │
# => ├───┼──────┼──────┼──────┼─────────┼──────┼──────────┼─────────┤
# => │ 0 │ 6585 │ 6584 │ nu   │ Running │ 0.00 │ 31.9 MiB │ 1.2 GiB │
# => ╰───┴──────┴──────┴──────┴─────────┴──────┴──────────┴─────────╯
```

Running processes


```
ps | where status == Running
# => ╭────┬──────┬──────┬──────┬─────────┬──────┬──────────┬─────────╮
# => │ #  │ pid  │ ppid │ name │ status  │ cpu  │   mem    │ virtual │
# => ├────┼──────┼──────┼──────┼─────────┼──────┼──────────┼─────────┤
# => │ 0  │ 6585 │ 6584 │ nu   │ Running │ 0.00 │ 31.9 MiB │ 1.2 GiB │
# => ╰────┴──────┴──────┴──────┴─────────┴──────┴──────────┴─────────╯
```

Shell

How does this work?

Running processes


```shell
ps | where status == Running
# => ╭────┬──────┬──────┬──────┬─────────┬──────┬──────────┬─────────╮
# => │ #  │ pid  │ ppid │ name │ status  │ cpu  │   mem    │ virtual │
# => ├────┼──────┼──────┼──────┼─────────┼──────┼──────────┼─────────┤
# => │ 0  │ 6585 │ 6584 │ nu   │ Running │ 0.00 │ 31.9 MiB │ 1.2 GiB │
# => ╰────┴──────┴──────┴──────┴─────────┴──────┴──────────┴─────────╯
```

How does this work?

```shell
ps | describe
# => table<pid: int, ppid: int, name: string, status: string, cpu: float, mem:
filesize, virtual: filesize> (stream)
```

Running processes


```shell
ps | where status == Running
# => ╭────┬──────┬──────┬──────┬─────────┬──────┬──────────┬─────────╮
# => │ #  │ pid  │ ppid │ name │ status  │ cpu  │   mem    │ virtual │
# => ├────┼──────┼──────┼──────┼─────────┼──────┼──────────┼─────────┤
# => │ 0  │ 6585 │ 6584 │ nu   │ Running │ 0.00 │ 31.9 MiB │ 1.2 GiB │
# => ╰────┴──────┴──────┴──────┴─────────┴──────┴──────────┴─────────╯
```

How does this work?

```shell
ps | describe
# => table<pid: int, ppid: int, name: string, status: string, cpu: float, mem:
filesize, virtual: filesize> (stream)
```

Exercise

Find processes sorted by greatest cpu utilization.

Exercise

Find processes sorted by greatest cpu utilization.

Shell

```
ps | where cpu > 0 | sort-by cpu | reverse
# => ──────┬─────────┬──────────────────────┬──────────┬──────────┬─────────
# =>  #    │   pid   │         name         │   cpu    │   mem    │ virtual
# => ──────┼─────────┼──────────────────────┼──────────┼──────────┼─────────
# =>  0    │ 11928   │ nu.exe               │ 32.12    │ 47.7 MB  │ 20.9 MB
# =>  1    │ 11728   │ Teams.exe            │ 10.71    │ 53.8 MB  │ 50.8 MB
# =>  2    │ 21460   │ msedgewebview2.exe   │  8.43    │ 54.0 MB  │ 36.8 MB
# => ──────┴─────────┴──────────────────────┴──────────┴──────────┴─────────
```

Pipelines

Example


```shell
ls
| sort-by size
| reverse
| first
| get name
| cp $in ~
```

Whenever possible, Nushell commands are designed to act on pipeline input.

Why does `cp` need `$in`?

Example


```Shell
ls
| sort-by size
| reverse
| first
| get name
| cp $in ~
```

Whenever possible, Nushell commands are designed to act on pipeline input.

Why does `cp` need `$in`?

Because `cp` has two positional arguments.

No `\` needed in multi-line pipelines.

Equivalent:

```Shell
ls | sort-by size | reverse | first | get name | cp $in ~
```

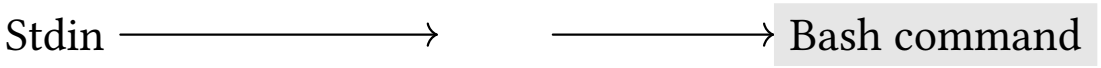
Battle of the pipelines

Bash pipeline:

Bash command

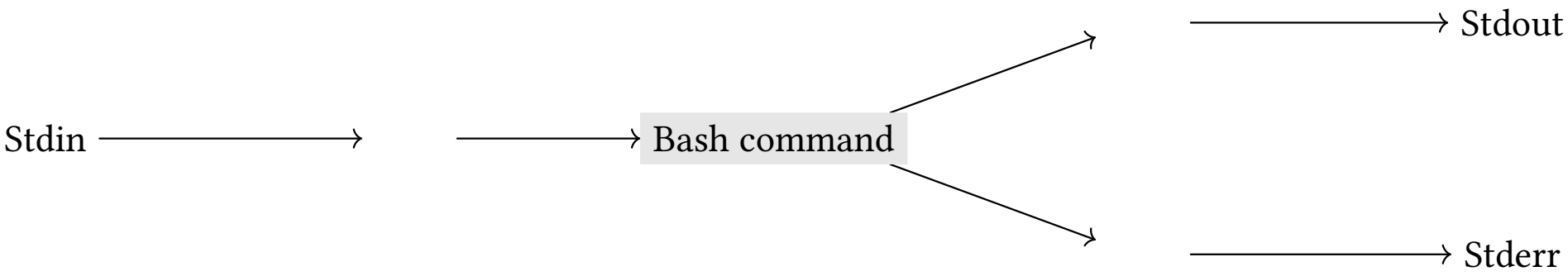
Battle of the pipelines

Bash pipeline:



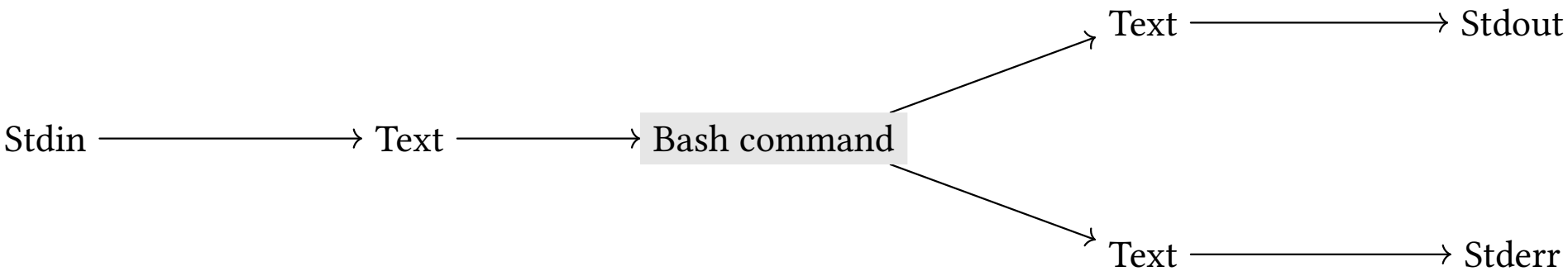
Battle of the pipelines

Bash pipeline:



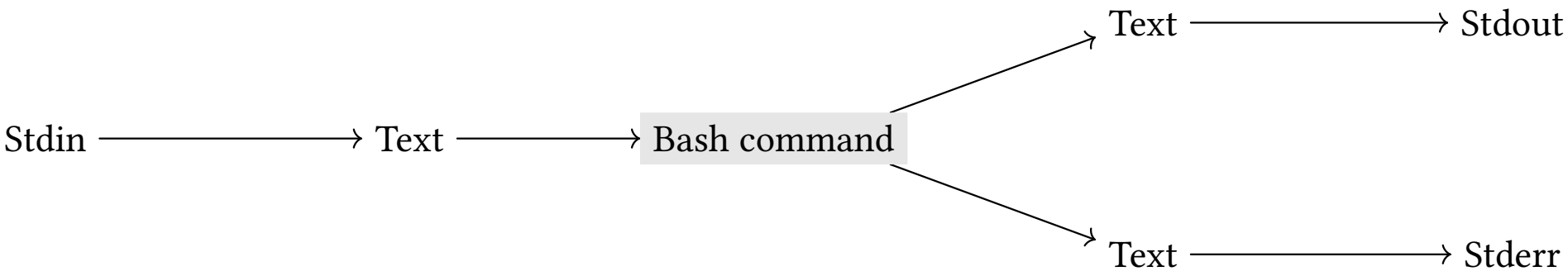
Battle of the pipelines

Bash pipeline:



Battle of the pipelines

Bash pipeline:

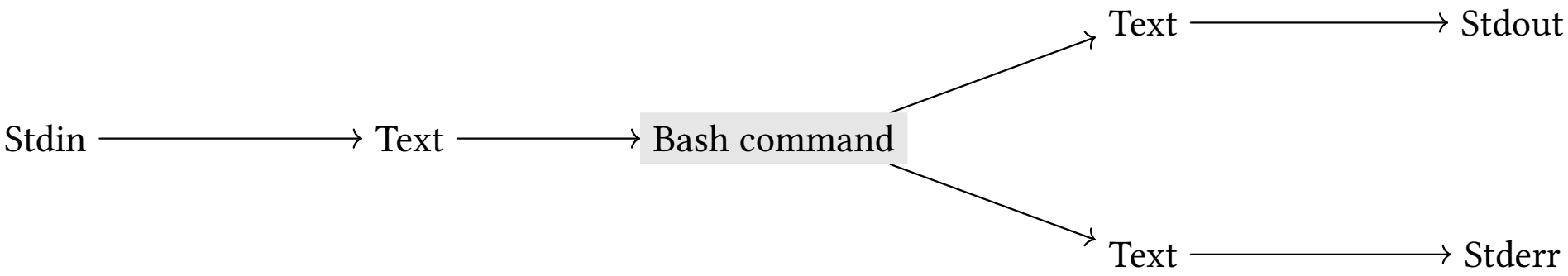


Nu pipeline:

Nu command

Battle of the pipelines

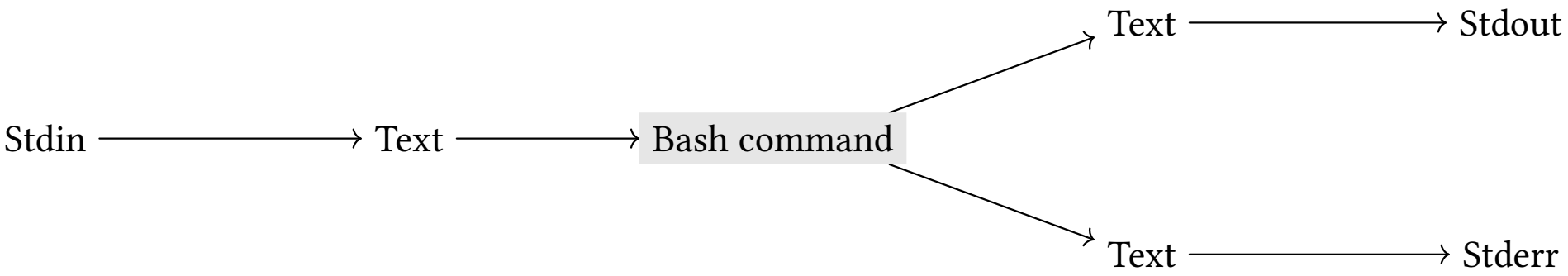
Bash pipeline:



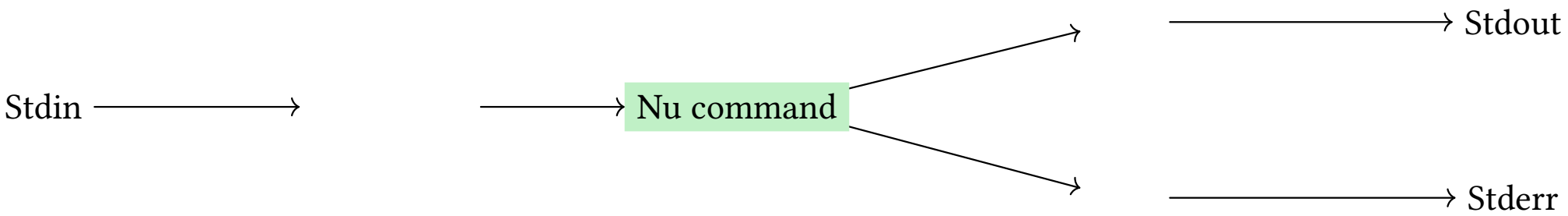
Nu pipeline:

Battle of the pipelines

Bash pipeline:

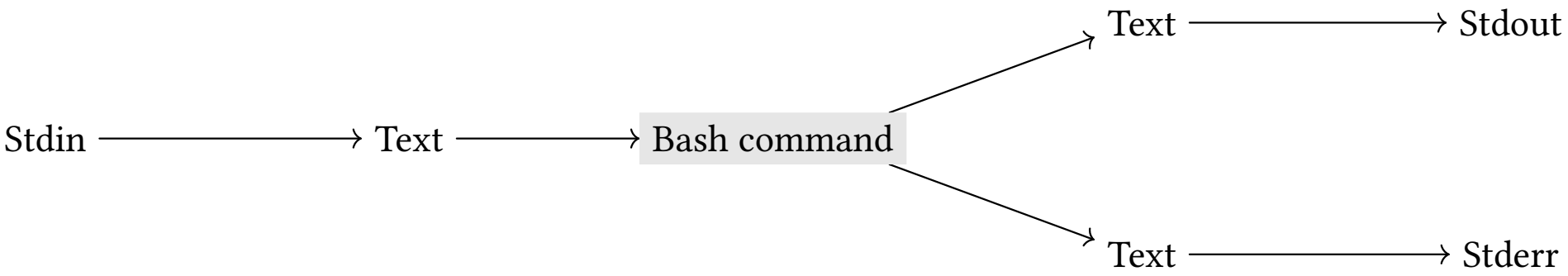


Nu pipeline:

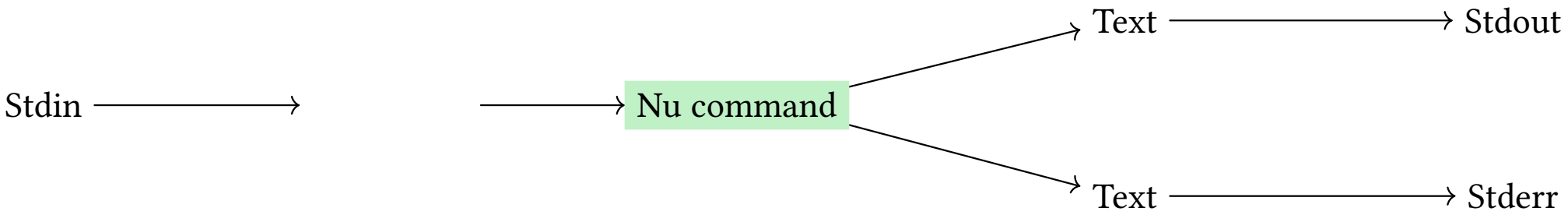


Battle of the pipelines

Bash pipeline:

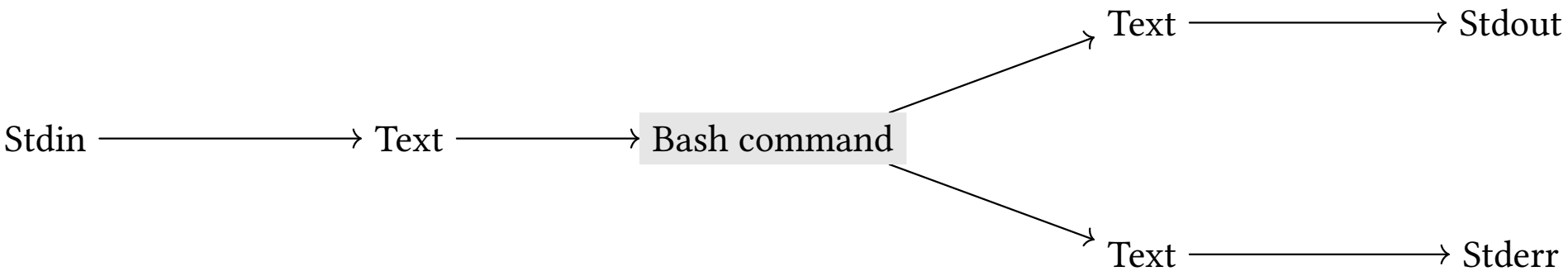


Nu pipeline:

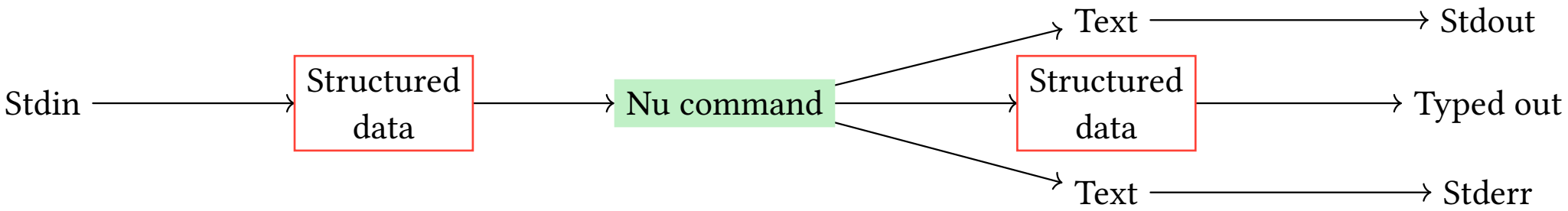


Battle of the pipelines

Bash pipeline:

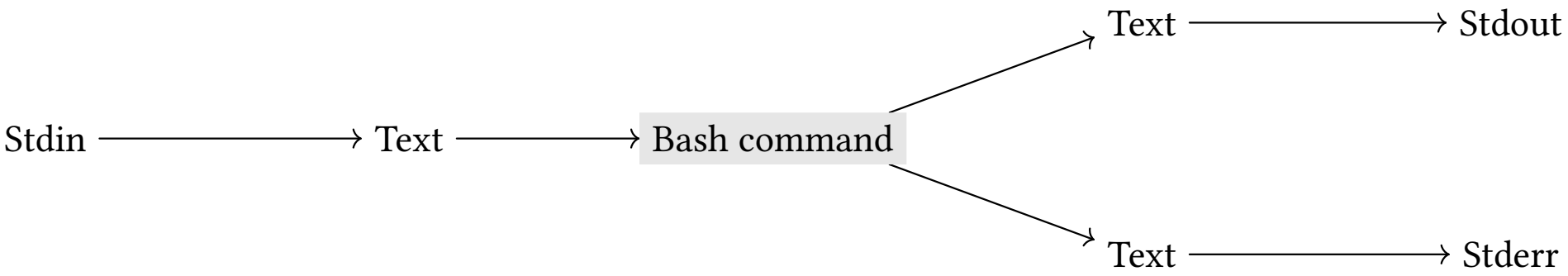


Nu pipeline:

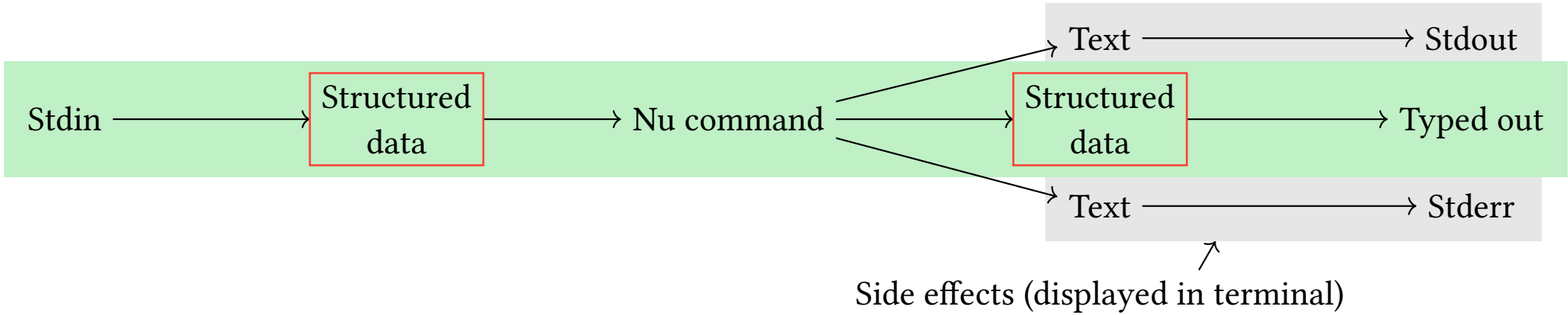


Battle of the pipelines

Bash pipeline:



Nu pipeline:



Records

Tables are built from rows of records:

```
ls                                                    Shell
| sort-by size                                        } (1) Table
| reverse                                             }
| first                                               } (2) Record
| get name                                            } (3) Cell path
| cp $in ~                                            } (4) String
```

Another way to find this out:

```
ls | sort-by size | reverse | first | describe                                    Shell
# => record<name: string, type: string, size: filesize, modified: datetime>
```

Exercise

Spawn a process and kill it based on its name.

Hint:

Exercise

Spawn a process and kill it based on its name.

Hint:

```
ps | where name == Notepad2.exe
# => ──────┬──────┬──────────────┬──────┬──────────┬─────────
# =>  #    │ pid  │     name     │ cpu  │   mem    │ virtual
# => ──────┼──────┼──────────────┼──────┼──────────┼─────────
# =>  0    │ 9268 │ Notepad2.exe │ 0.00 │ 32.0 MB  │  9.8 MB
# => ──────┴──────┴──────────────┴──────┴──────────┴─────────
```

Shell

Solution:

Exercise

Spawn a process and kill it based on its name.

Hint:

```
ps | where name == Notepad2.exe
# => ──────┬──────┬──────────────┬──────┬──────────┬─────────
# =>  #    │ pid  │     name     │ cpu  │   mem    │ virtual
# => ──────┼──────┼──────────────┼──────┼──────────┼─────────
# =>  0    │ 9268 │ Notepad2.exe │ 0.00 │ 32.0 MB  │  9.8 MB
# => ──────┴──────┴──────────────┴──────┴──────────┴─────────
```

Shell

Solution:

```
ps | where name == Notepad2.exe | get pid | get 0 | kill $in
# => ──────┬──────────────────────────────────────────────────────────────
# =>  0    │ SUCCESS: Sent termination signal to the process with PID 9268.
# => ──────┴──────────────────────────────────────────────────────────────
```

Shell

Or more concisely:

Exercise

Spawn a process and kill it based on its name.

Hint:

```
ps | where name == Notepad2.exe
# => ──────┬──────┬──────────────┬──────┬──────────┬─────────
# =>  #    │ pid  │     name     │ cpu  │   mem    │ virtual
# => ──────┼──────┼──────────────┼──────┼──────────┼─────────
# =>  0    │ 9268 │ Notepad2.exe │ 0.00 │ 32.0 MB  │  9.8 MB
# => ──────┴──────┴──────────────┴──────┴──────────┴─────────
```

Shell

Solution:

```
ps | where name == Notepad2.exe | get pid | get 0 | kill $in
# => ──────┬──────────────────────────────────────────────────────────────
# =>  0    │ SUCCESS: Sent termination signal to the process with PID 9268.
# => ──────┴──────────────────────────────────────────────────────────────
```

Shell

Or more concisely:

```
ps | where name == Notepad2.exe | get pid.0 | kill $in
```

Sh

Custom commands

Creating simple CLI tools

No positional arguments, no flags, just pipeline input and output.

Shell

```
def double [] {
  each { |num| 2 * $num }
}
```

Creating simple CLI tools

No positional arguments, no flags, just pipeline input and output.

```shell
def double [] {
  each { |num| 2 * $num }
}
```

Only positional arguments:

```shell
def greet [name1, name2] {
  $"Hello, ($name1) and ($name2)!"
}

greet Wei Mei
# => Hello, Wei and Mei!
```

Optional arguments


```nu
def greet [name?: string] {
  $"Hello, ($name | default 'You')"
}

greet
# => Hello, You
```

Default values:

```nu
def congratulate [age: int = 18] {
  $"Happy birthday! You are ($age) years old now!"
}
```

Typing pipeline

Shell

```
def inc []: int -> int {
  $in + 1
  print "Did it!"
}

# => Error: nu::parser::output_type_mismatch
# =>
# =>   × Command output doesn't match int.
# =>     ╭─[entry #1:1:24]
# =>   1 │ ╭─▶ def inc []: int -> int {
# =>   2 │ │       $in + 1
# =>   3 │ │       print "Did it!"
# =>   4 │ ├─▶ }
# =>     · ╰──── expected int, but command outputs nothing
# =>     ╰────
```

Documentation

Shell

```
# Greet guests along with a VIP
#
# Use for birthdays, graduation parties,
# retirements, and any other event which
# celebrates an event # for a particular
# person.
def vip-greet [
  vip: string        # The special guest
   ...names: string  # The other guests
] {
  for $name in $names {
    print $"Hello, ($name)!"
  }

  print $"And a special welcome to our VIP today, ($vip)!"
}
```

Exercises

Exercise: Create a command line program / script that takes a list of numbers as input and outputs their average.

Explore

Zooming-in

Telescoping into structured data:

```shell
help commands | explore
```

Key bindings:

- Go deeper: `Enter`
- Go back: `ESC` / `q`
- Navigate: Arrow keys or `j`/`k`

The `help` command is for built-in Nu commands. `man` is for external commands.

Data exploration

Open interactive data explorer with `:try` in `explore` mode.

Pipe current `explore` view into a pipeline with:

```
$in | select name description | where name == "ls"
```
Shell

(in older versions, maybe `$nu` instead of `$in`)

Exercise

Find the help page for the `cp` command and explore its output.

Use `help commands | explore` to find all commands in the `filters` category that contain "by" in their name.

Hint: In `:try` mode, use `where` and `=~` (or `str contains`).

Solution:

Exercise

Find the help page for the `cp` command and explore its output.

Use `help commands | explore` to find all commands in the `filters` category that contain "by" in their name.

Hint: In `:try` mode, use `where` and `=~` (or `str contains`).

Solution:

```
$in | where category == filters and name =~ by
```
Shell

Shorthand for:

Exercise

Find the help page for the `cp` command and explore its output.

Use `help commands | explore` to find all commands in the `filters` category that contain "by" in their name.

Hint: In `:try` mode, use `where` and `=~` (or `str contains`).

Solution:

```
$in | where category == filters and name =~ by
```
Shell

Shorthand for:

```
help commands |
where (
  ($it.category == "filters") and
  ($it.name | str contains "by")
)
```
Shell

} (1) Table output
} (2) Filter
} (3) Row condition 1
} (4) Row condition 2

List

Single column tables


```shell
[bell book candle] | where ($it =~ 'b')
# => ╭───┬──────╮
# => │ 0 │ bell │
# => │ 1 │ book │
# => ╰───┴──────╯
```

Commas are optional in list literals.

```shell
let colors = [yellow green]
let colors = ($colors | prepend red)
let colors = ($colors | append purple)
let colors = ($colors ++ ["blue"])
let colors = (["black"] ++ $colors)
$colors
# => [black red yellow green purple blue]
```

Iterating over lists


```nu
let names = [Mark Tami Amanda Jeremy]
$names | each { |elt| $"Hello, ($elt)!" }
# Outputs "Hello, Mark!" and three more similar lines.

$names | enumerate | each { |elt| $"($elt.index + 1) - ($elt.item)" }
# Outputs "1 - Mark", "2 - Tami", etc.
```

Filtering lists:

```nu
let colors = [red orange yellow green blue purple]
$colors | where ($it | str ends-with 'e')
# The block passed to `where` must evaluate to a boolean.
# This outputs the list [orange blue purple].
```

Boolean conditions:

```nu
let colors = [red green blue]
# Do any color names end with "e"?
$colors | any {|elt| $elt | str ends-with "e" } # true
```

Exercise

Exercise: Compute the sum of squares of the first 10 natural numbers.

Solution:

Exercise

Exercise: Compute the sum of squares of the first 10 natural numbers.

Solution:

```shell
1..10 | each { |n| $n * $n } | math sum
```

Exercise

Exercise: Compute the sum of squares of the first 10 natural numbers.

Solution:

```shell
1..10 | each { |n| $n * $n } | math sum
```

Exercise: Use the `lines` command to find out the first commit message in the Git history of this repository.

Solution:

Exercise

Exercise: Compute the sum of squares of the first 10 natural numbers.

Solution:

```shell
1..10 | each { |n| $n * $n } | math sum
```

Exercise: Use the `lines` command to find out the first commit message in the Git history of this repository.

Solution:

```shell
git log --oneline | lines | get 0 | str trim
```

Records

Single-row tables

Shell

```
let my_record = {
 name: "Sam"
 age: 30
 }
$my_record | update age { $in + 1 }
# => ╭──────┬─────╮
# => │ name │ Sam │
# => │ age  │ 31  │
# => ╰──────┴─────╯
```

Displayed like two columns, but still a single-row table (record).

Parsing JSON

Shell

```
# Nushell
{ "apples": 543, "bananas": 411, "oranges": 0 }
# => ╭─────────┬─────╮
# => │ apples  │ 543 │
# => │ bananas │ 411 │
# => │ oranges │ 0   │
# => ╰─────────┴─────╯
# JSON
'{ "apples": 543, "bananas": 411, "oranges": 0 }' | from json
# => ╭─────────┬─────╮
# => │ apples  │ 543 │
# => │ bananas │ 411 │
# => │ oranges │ 0   │
# => ╰─────────┴─────╯
```

Iterating

Shell

```
{ "apples": 543, "bananas": 411, "oranges": 0 } | items {|fruit, count| $"We have ($fruit) ($count)" }
# => ╭───┬──────────────────────────╮
# => │ 0 │ We have apples 543       │
# => │ 1 │ We have bananas 411      │
# => │ 2 │ We have oranges 0        │
# => ╰───┴──────────────────────────╯
```

Exercise

Exercise: Convert the following JQ command to Nushell:

```shell
echo '[{"name": "Alice", "age": 30}, {"name": "Bob", "age": 25}]' |
jq -r '.[] | select(.age > 28)'
```

Solution:

Exercise

Exercise: Convert the following JQ command to Nushell:

```shell
echo '[{"name": "Alice", "age": 30}, {"name": "Bob", "age": 25}]' |
jq -r '.[] | select(.age > 28)'
```

Solution:

```nu
'[{"name": "Alice", "age": 30}, {"name": "Bob", "age": 25}]'
| from json
| where age > 28
# => ╭───┬─────────┬─────╮
# => │ # │  name   │ age │
# => ├───┼─────────┼─────┤
# => │ 0 │ Alice   │  30 │
# => ╰───┴─────────┴─────╯
```

Exercise

Exercise: COnvert following JQ command to Nushell:

```shell
echo '[1, 2, 3, 4, 5]' |
jq -r 'map(. * 2)'
```

Solution:

```shell
'[1, 2, 3, 4, 5]'
| from json
| each { |x| $x * 2 }
# => ╭───┬────╮
# => │ 0 │  2 │
# => │ 1 │  4 │
# => │ 2 │  6 │
# => │ 3 │  8 │
# => │ 4 │ 10 │
# => ╰───┴────╯
```

Tables

Creating tables

Shell

```
let first = [[a b]; [1 2]]
let second = [[a b]; [3 4]]
$first | append $second
# => ──────────────
# =>  #  a  b
# => ──────────────
# =>  0  1  2
# =>  1  3  4
# => ──────────────
```

Getting data out


```
ls | get name
```
Shell

```
# => ──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

```
ls | select name
```
Shell

```
# => ──┬──────────────────
# =>  # │ name
# => ──┼──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

What is the difference between `get` and `select`?

Getting data out


```
ls | get name
```
Shell

```
# => ──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

```
ls | select name
```
Shell

```
# => ──┬──────────────────
# =>  # │ name
# => ──┼──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

What is the difference between `get` and `select`?

`get` returns a single column as a list, while `select` returns a table with one column.

What is the type of the arguments passed to `get` and `select`?

Getting data out


```
ls | get name
# => ──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

Shell

```
ls | select name
# => ──┬──────────────────
# =>  # │ name
# => ──┼──────────────────
# =>  0 │ files.rs
# =>  1 │ lib.rs
# =>  2 │ lite_parse.rs
# =>  3 │ parse.rs
# =>  4 │ path.rs
# =>  5 │ shapes.rs
# => ──┴──────────────────
```

Shell

What is the difference between `get` and `select`?

`get` returns a single column as a list, while `select` returns a table with one column.

What is the type of the arguments passed to `get` and `select`?

So-called “cell-paths”, relative paths into structured data.

Exercise

Download dataset with `fetch-data.nu`.

Explore the dataset: `open data/covid_19_data.csv | explore`.

Exercise: try sorting by Deaths descending

Solution:

Exercise

Download dataset with `fetch-data.nu`.

Explore the dataset: `open data/covid_19_data.csv | explore`.

Exercise: try sorting by Deaths descending

Solution:

```shell
open data/covid_19_data.csv | sort-by Deaths --reverse
```

Data exploration

Grouping

```
open data/covid_19_data.csv | group-by `Country/Region`
# => ╭─────────────────┬──────────────────────────╮
# => │ Afghanistan     │ [table 75 rows]          │
# => │ Albania         │ [table 74 rows]          │
# => │ Algeria         │ [table 75 rows]          │
# => │ ...             │ ...                      │
# => ╰─────────────────┴──────────────────────────╯
```

Shell

Does the previous command return a record or a table?

Grouping

```
open data/covid_19_data.csv | group-by `Country/Region`
# => ╭─────────────────┬──────────────────────────╮
# => │ Afghanistan     │ [table 75 rows]          │
# => │ Albania         │ [table 74 rows]          │
# => │ Algeria         │ [table 75 rows]          │
# => │ ...             │ ...                      │
# => ╰─────────────────┴──────────────────────────╯
```

Shell

Does the previous command return a record or a table?

A record: Keys = country names, Values = tables of rows for that country.

From record to table

Records cannot be iterated with `each` (which expects a table/list).

Use `transpose` to convert a record into a table:

Shell

```
{a: 1, b: 2} | transpose
# => ╭───┬──────────┬──────────╮
# => │ # │ column0  │ column1  │
# => ├───┼──────────┼──────────┤
# => │ 0 │ a        │        1 │
# => │ 1 │ b        │        2 │
# => ╰───┴──────────┴──────────╯

{a: 1, b: 2} | transpose key value
# => ╭───┬─────┬───────╮
# => │ # │ key │ value │
# => ├───┼─────┼───────┤
# => │ 0 │ a   │     1 │
# => │ 1 │ b   │     2 │
# => ╰───┴─────┴───────╯
```

Total deaths per country

Shell

```
open data/covid_19_data.csv
| group-by `Country/Region`
| transpose country rows
# => ╭────┬─────────────┬──────────────────╮
# => │ #  │   country   │       rows       │
# => ├────┼─────────────┼──────────────────┤
# => │ 0  │ Afghanistan │ [table 75 rows]  │
# => │ 1  │ Albania     │ [table 74 rows]  │
# => │ ...                                 │
# => ╰────┴─────────────┴──────────────────╯

# Now we can iterate with each
open data/covid_19_data.csv
| group-by `Country/Region`
| transpose country rows
| each {|group| $group.rows | get Deaths | math sum}
```

What does `$group.rows` contain?

Total deaths per country

Shell

```
open data/covid_19_data.csv
| group-by `Country/Region`
| transpose country rows
# => ╭─────┬─────────────┬──────────────────╮
# => │  #  │   country   │       rows       │
# => ├─────┼─────────────┼──────────────────┤
# => │  0  │ Afghanistan │ [table 75 rows]  │
# => │  1  │ Albania     │ [table 74 rows]  │
# => │ ... │             │                  │
# => ╰─────┴─────────────┴──────────────────╯

# Now we can iterate with each
open data/covid_19_data.csv
| group-by `Country/Region`
| transpose country rows
| each {|group| $group.rows | get Deaths | math sum}
```

What does `$group.rows` contain?

The table of all COVID rows for that country in the original table.

Exercises

Exercise: Add an additional column with country name and sort by deaths. Solution:

Exercises

Exercise: Add an additional column with country name and sort by deaths. Solution:

Shell

```
open data/covid_19_data.csv
| group-by `Country/Region`
| transpose country rows
| each {|group|
    {country: $group.country, total_deaths: ($group.rows | get Deaths | math sum)}
  }
| sort-by total_deaths --reverse
```

What would be an even simpler way to achieve this without using `transpose`?

Exercises

Exercise: Add an additional column with country name and sort by deaths. Solution:

```
  open data/covid_19_data.csv
  | group-by `Country/Region`
  | transpose country rows
  | each {|group|
      {country: $group.country, total_deaths: ($group.rows | get Deaths | math sum)}
    }
  | sort-by total_deaths --reverse
```
Shell

What would be an even simpler way to achieve this without using `transpose`?

Using `items` directly on the grouped record.

```
open data/covid_19_data.csv
| group-by `Country/Region`
| items {|country, rows|
    {country: $country, total_deaths: ($rows | get Deaths | math sum)}
  }
| sort-by total_deaths --reverse
```
Shell