

# NuShell

Superglue for your OS

Willem Vanhulle



Wednesday, Dec 3, 2025

# **Introduction**

---

What does the following Bash code do?

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

[ Shell

What does the following Bash code do?

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

[Shell]

Nu:

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

[Shell]

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?

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

[ Shell]

Nu:

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

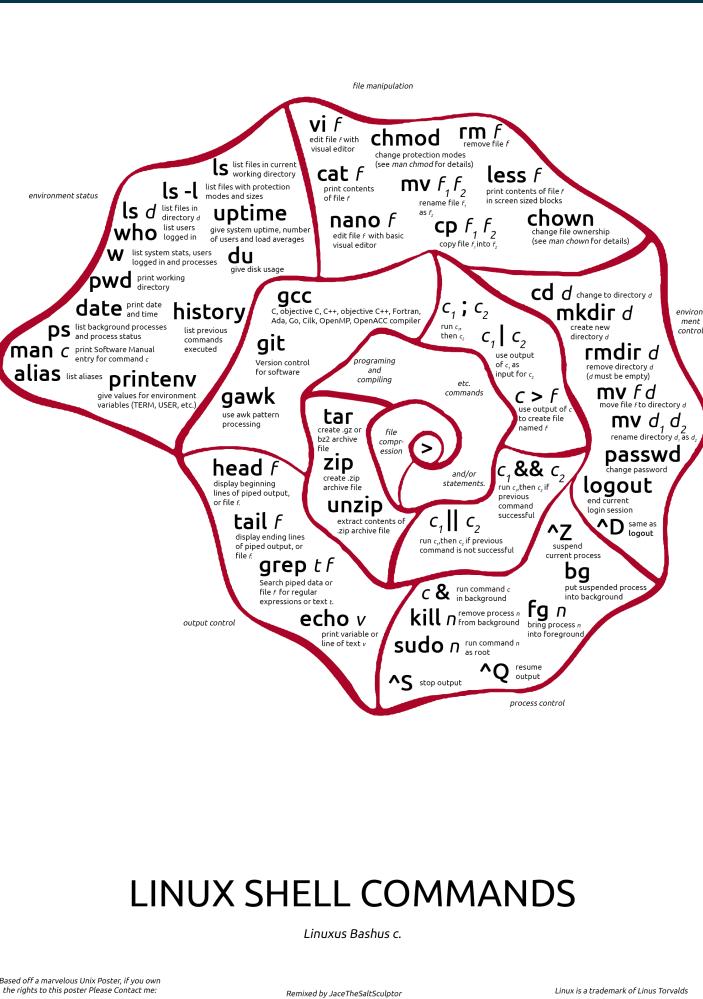
[ Shell]

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

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

- Download binary: [github.com/nushell/nushell/releases](https://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`

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

# Exercises

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

To run an exercise:

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

 Shell

To pipe in data from Bash:

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

 Shell

Piping within NuShell:

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

 Shell

## Basics

---

# Commands output tables

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

```
ls | where size > 10kb
```

 Shell

# =>	#	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

ps

Shell

# =>	#	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

```
ps | where status == Running
```

Shell

#	#	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
```

Shell

#	#	pid	ppid	name	status	cpu	mem	virtual
# =>								
# =>	0	6585	6584	nu	Running	0.00	31.9 MiB	1.2 GiB
# =>								

How does this work?

# Running processes

```
ps | where status == Running
```

 Shell

#	pid	ppid	name	status	cpu	mem	virtual
# =>							
# =>	0	6585	6584	nu	Running	0.00	31.9 MiB
# =>							
# =>							

How does this work?

```
ps | describe
```

 Shell

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

# Running processes

```
ps | where status == Running
```

 Shell

#	pid	ppid	name	status	cpu	mem	virtual
# =>							
# =>	0	6585	6584	nu	Running	0.00	31.9 MiB
# =>							
# =>							

How does this work?

```
ps | describe
```

 Shell

```
# => 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.

```
ps | where cpu > 0 | sort-by cpu | reverse
```

 Shell

# =>	#	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

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

 Shell

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

Why does cp need \$in?

# Example

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

 Shell

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:

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

 Shell

# Battle of the pipelines

Bash pipeline:

Bash command

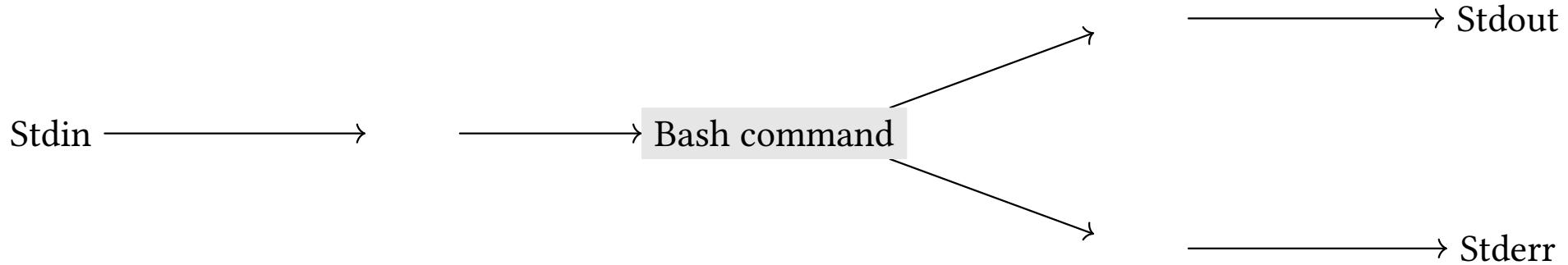
# Battle of the pipelines

**Bash** pipeline:

Stdin —————→      —————→ Bash command

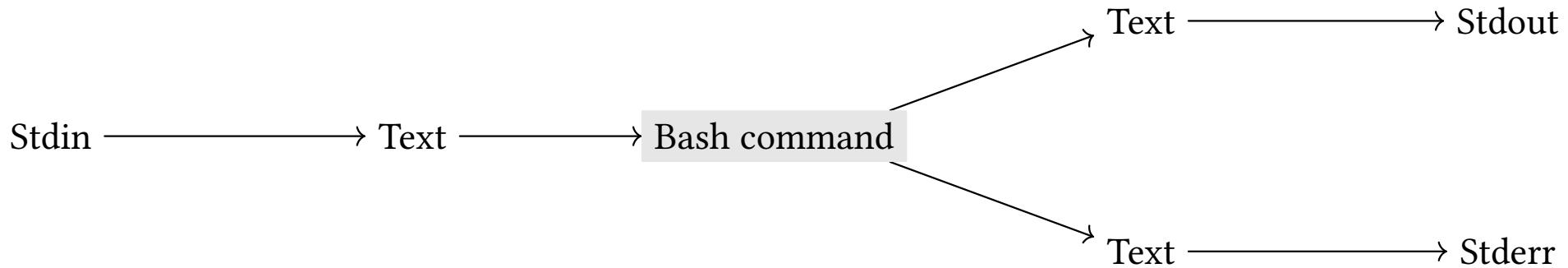
# Battle of the pipelines

Bash pipeline:



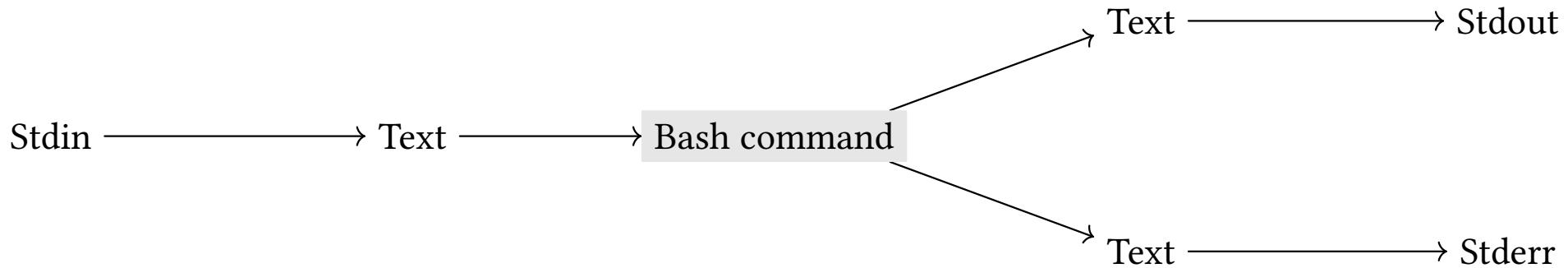
# Battle of the pipelines

Bash pipeline:



# Battle of the pipelines

Bash pipeline:

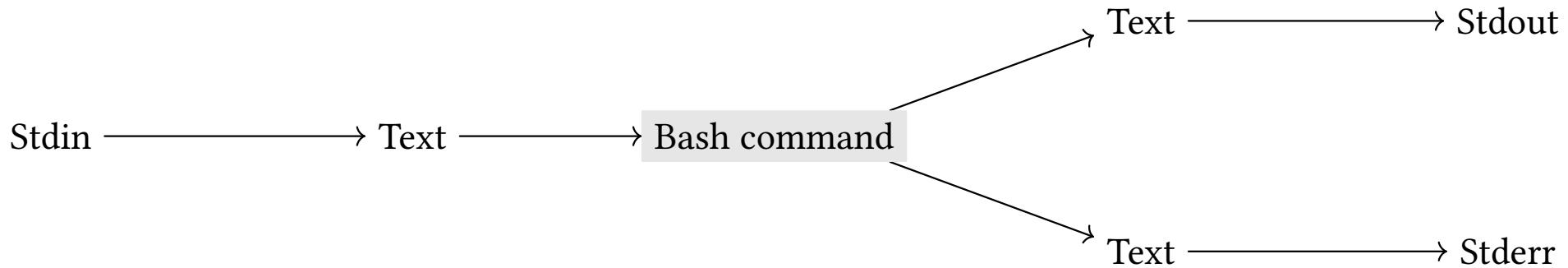


Nu pipeline:

Nu command

# Battle of the pipelines

Bash pipeline:

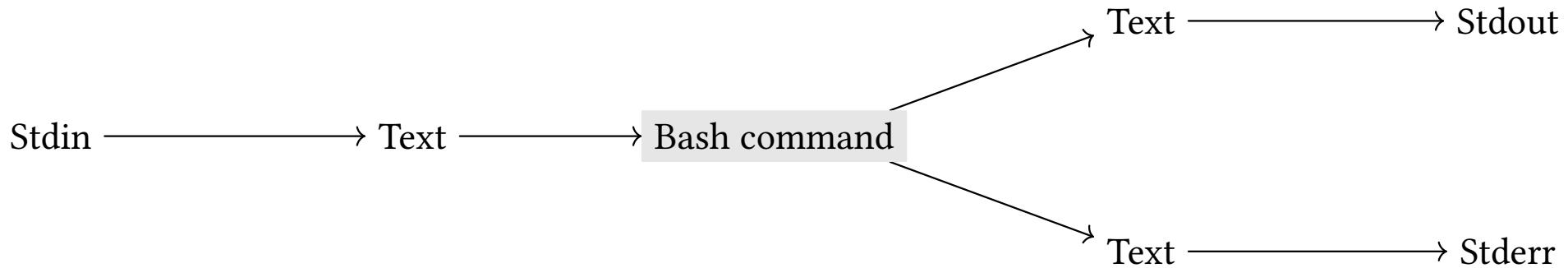


Nu pipeline:

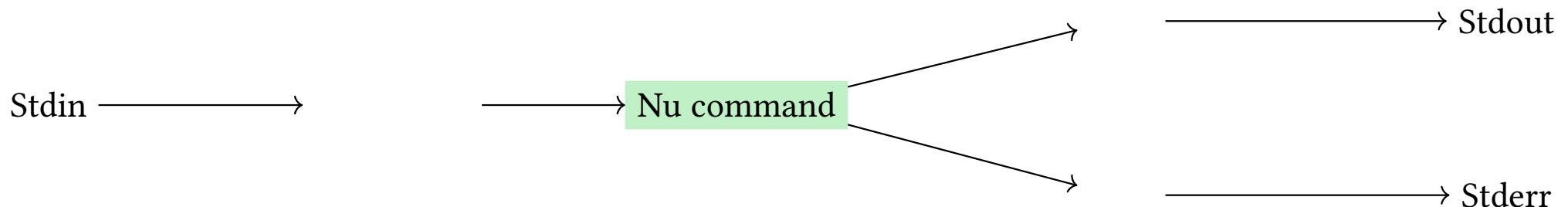


# Battle of the pipelines

Bash pipeline:

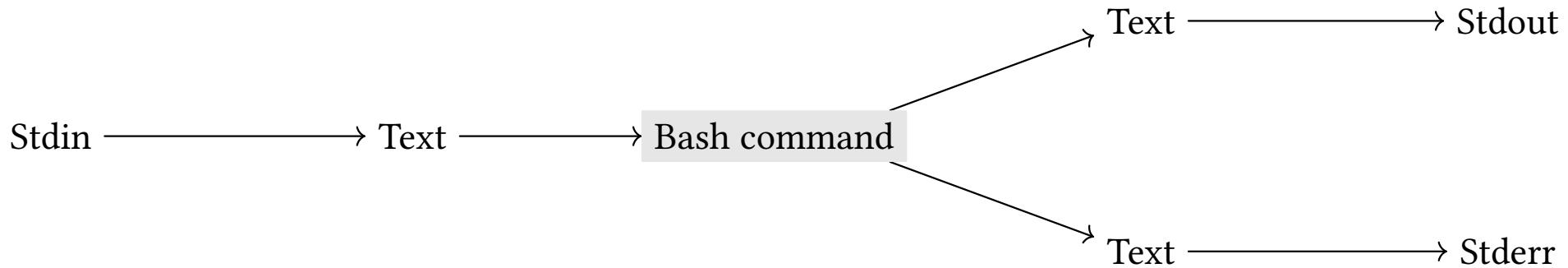


Nu pipeline:

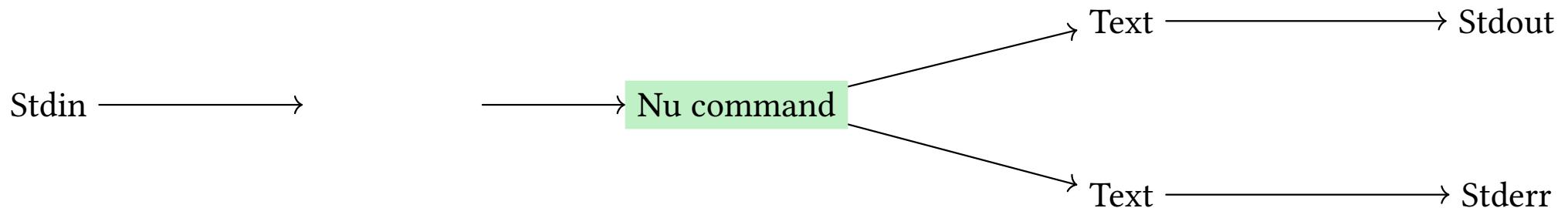


# Battle of the pipelines

Bash pipeline:

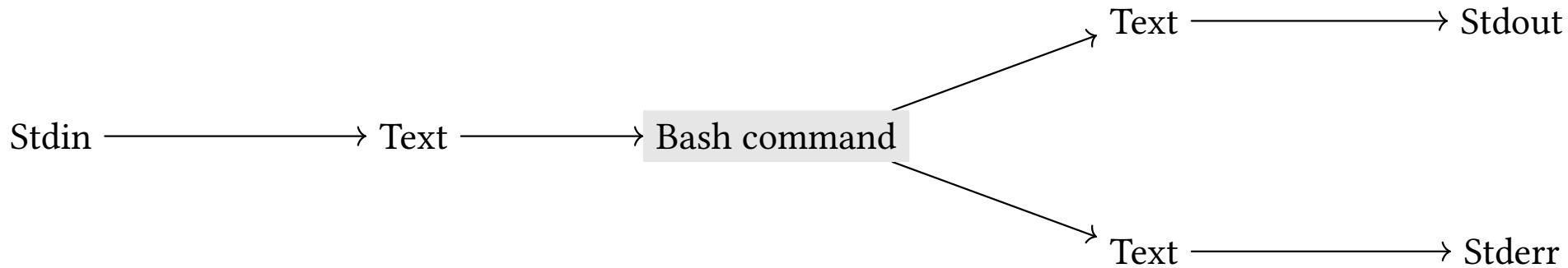


Nu pipeline:

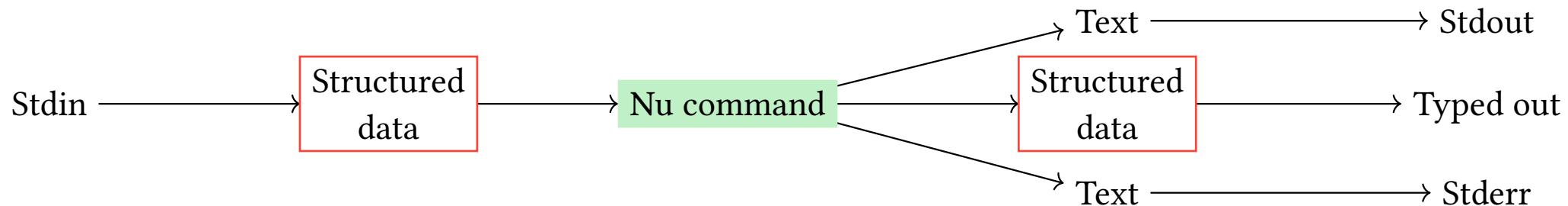


# Battle of the pipelines

Bash pipeline:

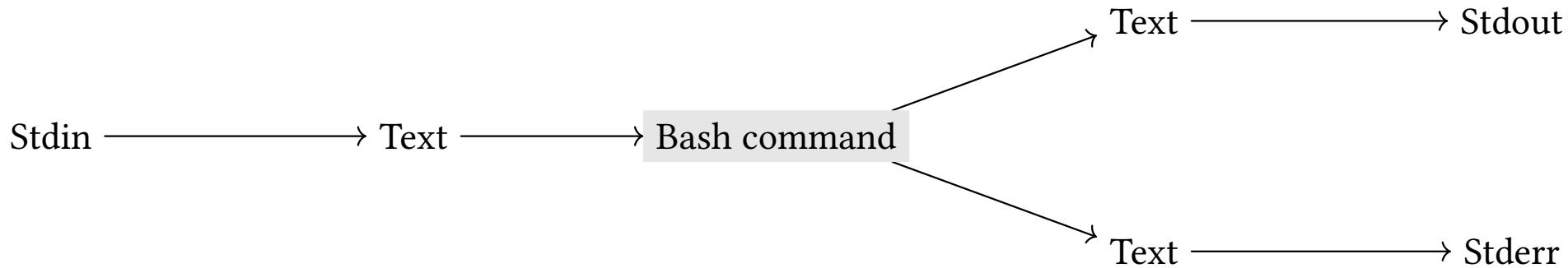


Nu pipeline:

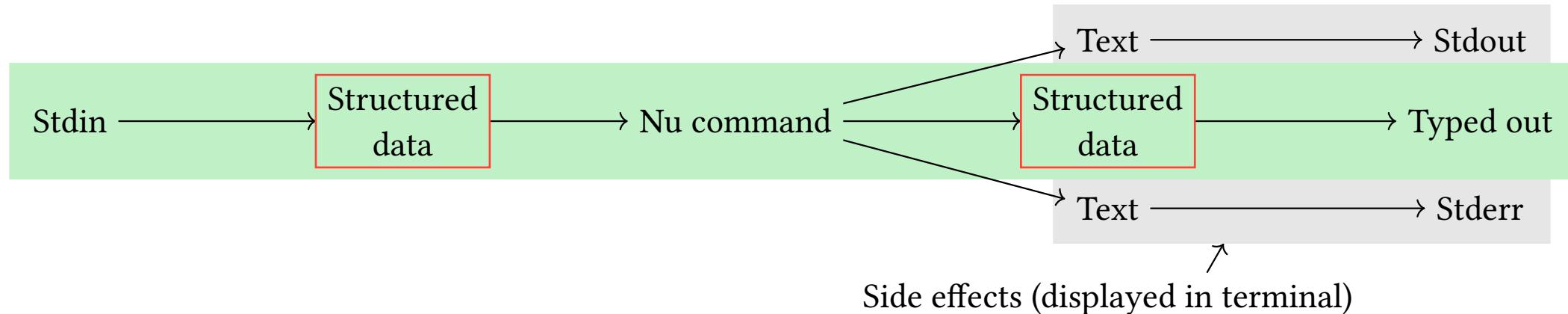


# Battle of the pipelines

Bash pipeline:



Nu pipeline:



# Records

Tables are built from rows of records:

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

 Shell } (1) Table  
} (2) Record  
} (3) Cell path  
} (4) String

Another way to find this out:

```
ls | sort-by size | reverse | first | describe  
# => 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

Shell

#	#	pid	name	cpu	mem	virtual
# =>						
# =>	0	9268	Notepad2.exe	0.00	32.0 MB	9.8 MB
# =>						

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

Solution:

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

0	SUCCESS: Sent termination signal to the process with PID 9268.
---	--

Or more concisely:

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

## Custom commands

---

# Creating simple CLI tools

Custom commands 

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

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

 Shell

# Creating simple CLI tools

Custom commands 

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

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

 Shell

Only positional arguments:

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

 Shell

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

# Optional arguments

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

 Shell

```
greet  
# => Hello, You
```

Default values:

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

 Shell

# Typing pipeline

Custom commands 

```
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  
# =>
```

 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)!"
}
```

 Shell

# Exercises

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

**Warning:** You need the shebang: `#!/usr/bin/env -S nu --stdin` to be able to pipe into it.

**Explore**

---

# Zooming-in

Explore 

Telescoping into structured data:

[help](#) [commands](#) | [explore](#)

 Shell

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

```
[bell book candle] | where ($it =~ 'b')
```

 Shell

```
# => 

|   |      |
|---|------|
| 0 | bell |
| 1 | book |


```

Commas are optional in list literals.

```
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]
```

 Shell

# Iterating over lists

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

 Shell

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

Filtering lists:

```
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].
```

 Shell

Boolean conditions:

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

 Shell

# 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:

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

 Shell

# Exercise

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

Solution:

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

 Shell

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:

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

 Shell

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

Solution:

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

 Shell

## Records

---

# Single-row tables

```
let my_record = {  
    name: "Sam"  
    age: 30  
}  
  
$my_record | update age { $in + 1 }  
  
# =>  
# => 

|      |     |
|------|-----|
| name | Sam |
| age  | 31  |

  
# =>
```

 Shell

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

# Parsing JSON

```
# 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

```
{ "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 ]  
# =>
```

 Shell

# Exercise

Exercise: Convert the following JQ command to Nushell:

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

 Shell

Solution:

# Exercise

Exercise: Convert the following JQ command to Nushell:

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

 Shell

Solution:

```
'[{"name": "Alice", "age": 30}, {"name": "Bob", "age": 25}]'  
| from json  
| where age > 28
```

 Shell

#	name	age
0	Alice	30

# Exercise

Exercise: COnvert following JQ command to Nushell:

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

 Shell

Solution:

```
'[1, 2, 3, 4, 5]'  
| from json  
| each { |x| $x * 2 }  
# => 

|   |    |
|---|----|
|   |    |
| 0 | 2  |
| 1 | 4  |
| 2 | 6  |
| 3 | 8  |
| 4 | 10 |
|   |    |

  
# => 0 2  
# => 1 4  
# => 2 6  
# => 3 8  
# => 4 10  
# =>
```

 Shell

## Tables

---

# Creating tables

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

 Shell

# 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?

# 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

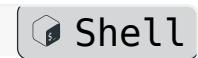
What is the difference between `get` and `select`?

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`?

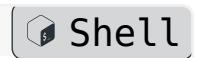
# Getting data out

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:

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

 Shell

## Data exploration

---

# Grouping

```
open data/covid_19_data.csv | group-by `Country/Region`
```

 Shell

# =>	Afghanistan	[table 75 rows]
# =>	Albania	[table 74 rows]
# =>	Algeria	[table 75 rows]
# =>	...	...
# =>		

Does the previous command return a record or a table?

# Grouping

```
open data/covid_19_data.csv | group-by `Country/Region`
```

Shell

# =>	Afghanistan	[table 75 rows]
# =>	Albania	[table 74 rows]
# =>	Algeria	[table 75 rows]
# =>	...	...
# =>		

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:

```
{a: 1, b: 2} | transpose
```

Shell

```
# =>
# =>      # | 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

```
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}
```

 Shell

What does \$group.rows contain?

# Total deaths per country

```
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:

```
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?

# 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

## **Personal project nu-lint**

---

```
●132 def check-membership [item list] {
133   | $item in $list
134 }
135
136 # Multiple violations in single function
●137 def complex-violations [data] {
    | Parameter 'data' is missing type annotation
    | Custom command 'complex-violations' produces output but lacks output type annotation
    | Use pipeline input instead of parameter
138   mut result = []
139   $data | each { item |
●140     if $item > 0 {
141       if $item < 100 {
142         let processed = $item * 2
●143         $result = $result ++ [$processed]
144       }
145     }
146   }
●147   let final = $result
148   $final
149 }
150
151 # External commands that should use builtins
●152 def external-heavy [] {
```

NOR benches/fixtures/with\_violations.nu

● 119 ● 3 1 sel 137:25

# Nu-lint is a Nu linter

Ready for testing. Compatible with earlier NuShell tools.

Personal project nu-lint 

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

List all rules with `nu-lint list-rules`.

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

List all rules with `nu-lint list-rules`.

```
bashisms (16)
prefer_builtin_awk
prefer_builtin_cat
prefer_builtin_cd
prefer_builtin_cut
prefer_builtin_date
prefer_builtin_echo
prefer_builtin_find
prefer_builtin_grep
prefer_builtin_head
prefer_builtin_ls
prefer_builtin_read
prefer_builtin_sed
prefer_builtin_sort
prefer_builtin_tail
prefer_builtin_uniq
prefer_builtin_wc
```

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

List all rules with `nu-lint list-rules`.

## **bashisms** (16)

`prefer_builtin_awk`  
`prefer_builtin_cat`  
`prefer_builtin_cd`  
`prefer_builtin_cut`  
`prefer_builtin_date`  
`prefer_builtin_echo`  
`prefer_builtin_find`  
`prefer_builtin_grep`  
`prefer_builtin_head`  
`prefer_builtin_ls`  
`prefer_builtin_read`  
`prefer_builtin_sed`  
`prefer_builtin_sort`  
`prefer_builtin_tail`  
`prefer_builtin_uniq`  
`prefer_builtin_wc`

## **external-tools** (9)

`prefer_builtin_curl`  
`prefer_builtin_eza`  
`prefer_builtin_fd`  
`prefer_builtin_hostname`  
`prefer_nushell_over_jq`  
`prefer_builtin.getenv`  
`prefer_builtin_rg`  
`prefer_builtin_wget`  
`prefer_builtin_which`

## **error-handling** (8)

`add_metadata_to_error`  
`check_complete_exit_code`  
`descriptive_error_messages`  
`escape_string_interpolation_operators`  
`prefer_complete_for_external_commands`  
`prefer_error_make_for_stderr`  
`prefer_try_for_error_handling`  
`print_exit_use_error_make`

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

List all rules with `nu-lint list-rules`.

## **bashisms** (16)

`prefer_builtin_awk`  
`prefer_builtin_cat`  
`prefer_builtin_cd`  
`prefer_builtin_cut`  
`prefer_builtin_date`  
`prefer_builtin_echo`  
`prefer_builtin_find`  
`prefer_builtin_grep`  
`prefer_builtin_head`  
`prefer_builtin_ls`  
`prefer_builtin_read`  
`prefer_builtin_sed`  
`prefer_builtin_sort`  
`prefer_builtin_tail`  
`prefer_builtin_uniq`  
`prefer_builtin_wc`

## **external-tools** (9)

`prefer_builtin_curl`  
`prefer_builtin_eza`  
`prefer_builtin_fd`  
`prefer_builtin_hostname`  
`prefer_nushell_over_jq`  
`prefer_builtin.getenv`  
`prefer_builtin_rg`  
`prefer_builtin_wget`  
`prefer_builtin_which`

## **error-handling** (8)

`add_metadata_to_error`  
`check_complete_exit_code`  
`descriptive_error_messages`  
`escape_string_interpolation_operators`  
`prefer_complete_for_external_commands`  
`prefer_error_make_for_stderr`  
`prefer_try_for_error_handling`  
`print_exit_use_error_make`

## **performance** (10)

`prefer_compound_assignment`  
`prefer_direct_use`  
`prefer_lines_over_split`  
`prefer_parse_over_split`  
`prefer_pipeline_input`  
`prefer_range_iteration`  
`prefer_where_over_each_if`  
`prefer_where_over_for_if`  
`remove_redundant_in`  
`unnecessary_variable_before_return`

## **formatting** (7)

`brace_spacing`  
`no_trailing_spaces`  
`omit_list_commas`  
`pipe_spacing`  
`prefer_multiline_functions`  
`prefer_multiline_lists`  
`prefer_multiline_records`

# Nu-lint is a Nu linter

Personal project nu-lint 

Ready for testing. Compatible with earlier NuShell tools.

Download at <https://www.github.com/wvhulle/nu-lint>.

List all rules with `nu-lint list-rules`.

## **bashisms** (16)

`prefer_builtin_awk`  
`prefer_builtin_cat`  
`prefer_builtin_cd`  
`prefer_builtin_cut`  
`prefer_builtin_date`  
`prefer_builtin_echo`  
`prefer_builtin_find`  
`prefer_builtin_grep`  
`prefer_builtin_head`  
`prefer_builtin_ls`  
`prefer_builtin_read`  
`prefer_builtin_sed`  
`prefer_builtin_sort`  
`prefer_builtin_tail`  
`prefer_builtin_uniq`  
`prefer_builtin_wc`

## **external-tools** (9)

`prefer_builtin_curl`  
`prefer_builtin_eza`  
`prefer_builtin_fd`  
`prefer_builtin_hostname`  
`prefer_nushell_over_jq`  
`prefer_builtin.getenv`  
`prefer_builtin_rg`  
`prefer_builtin_wget`  
`prefer_builtin_which`

## **error-handling** (8)

`add_metadata_to_error`  
`check_complete_exit_code`  
`descriptive_error_messages`  
`escape_string_interpolation_operators`  
`prefer_complete_for_external_commands`  
`prefer_error_make_for_stderr`  
`prefer_try_for_error_handling`  
`print_exit_use_error_make`

## **performance** (10)

`prefer_compound_assignment`  
`prefer_direct_use`  
`prefer_lines_over_split`  
`prefer_parse_over_split`  
`prefer_pipeline_input`  
`prefer_range_iteration`  
`prefer_where_over_each_if`  
`prefer_where_over_for_if`  
`remove_redundant_in`  
`unnecessary_variable_before_return`

## **formatting** (7)

`brace_spacing`  
`no_trailing_spaces`  
`omit_list_commas`  
`pipe_spacing`  
`prefer_multiline_functions`  
`prefer_multiline_lists`  
`prefer_multiline_records`

## **documentation** (4)

`exported_function_docs`  
`main_named_args_docs`  
`main_positional_args_docs`  
`descriptive_error_messages`

## **naming** (3)

`kebab_case_commands`  
`screaming_snake_constants`  
`snake_case_variables`

## **type-safety** (4)

`missing_type_annotation`  
`prefer_path_type`  
`typed_pipeline_io`  
`external_script_as_argument`

## **side-effects** (3)

`mixed_io_types`  
`print_and_return_data`  
`pure_before_side_effects`

## **systemd** (1)

`systemd_journal_prefix`

# Nu service with Nix

Personal project nu-lint 

Create a custom Nix module to **control brightness based on civil sunset and sunrise times (heliocron)**:

# Nu service with Nix

Personal project nu-lint 

Create a custom Nix module to **control brightness based on civil sunset and sunrise times (heliocron)**:

```
let
  cfg = config.programs.solar-brightness;

  brightnessScript = pkgs.writers.writeNuBin "solar-brightness" (
    builtins.readFile ./solar-brightness-manager.nu
);

in
{
  options.programs.solar-brightness = {
    enable = lib.mkEnableOption "Solar-based brightness control";

    interval-minutes = lib.mkOption {
      type = lib.types.str;
      default = "15min";
      description = "Check brightness interval (systemd/nushell duration format: sec, min, hr, day)";
      example = "30min";
    };
  };
}
```

Find the code at <https://github.com/wvhulle/nix-user-modules>, my public Nix user modules. Feel free to use any of them!

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {
    systemd.user.services.solar-brightness = {
        Unit = {
            Description = "Solar-based brightness adjustment";
            After = [ "graphical-session.target" ];
            PartOf = [ "graphical-session.target" ];
        };
        Service = {
            Type = "oneshot";
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}"';
        };
    };
};
```

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {
    systemd.user.services.solar-brightness = {
        Unit = {
            Description = "Solar-based brightness adjustment";
            After = [ "graphical-session.target" ];
            PartOf = [ "graphical-session.target" ];
        };
        Service = {
            Type = "oneshot";
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}"';
        };
    };
};
```

- Import the external Nu file in the Nix store

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {  
    systemd.user.services.solar-brightness = {  
        Unit = {  
            Description = "Solar-based brightness adjustment";  
            After = [ "graphical-session.target" ];  
            PartOf = [ "graphical-session.target" ];  
        };  
        Service = {  
            Type = "oneshot";  
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}''';  
        };  
    };  
};
```



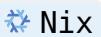
- Import the external Nu file in the Nix store (use an external .nu file for linter support!)

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {  
    systemd.user.services.solar-brightness = {  
        Unit = {  
            Description = "Solar-based brightness adjustment";  
            After = [ "graphical-session.target" ];  
            PartOf = [ "graphical-session.target" ];  
        };  
        Service = {  
            Type = "oneshot";  
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}''';  
        };  
    };  
};
```



- Import the external Nu file in the Nix store (use an external .nu file for linter support!)
- Call the imported Nu script in the Nix store in ExecStart

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {  
    systemd.user.services.solar-brightness = {  
        Unit = {  
            Description = "Solar-based brightness adjustment";  
            After = [ "graphical-session.target" ];  
            PartOf = [ "graphical-session.target" ];  
        };  
        Service = {  
            Type = "oneshot";  
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}''';  
        };  
    };  
};
```

- Import the external Nu file in the Nix store (use an external .nu file for linter support!)
- Call the imported Nu script in the Nix store in ExecStart
- Pass along command-line arguments with string interpolation

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {
    systemd.user.services.solar-brightness = {
        Unit = {
            Description = "Solar-based brightness adjustment";
            After = [ "graphical-session.target" ];
            PartOf = [ "graphical-session.target" ];
        };
        Service = {
            Type = "oneshot";
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}"''';
        };
    };
};
```

- Import the external Nu file in the Nix store (use an external .nu file for linter support!)
- Call the imported Nu script in the Nix store in ExecStart
- Pass along command-line arguments with string interpolation (use NuShells built-in command line argument parsing!)

# Calling the Nu script

Personal project nu-lint 

Declare a systemd service that uses the options from your custom solar Nix module:

```
config = lib.mkIf cfg.enable {
    systemd.user.services.solar-brightness = {
        Unit = {
            Description = "Solar-based brightness adjustment";
            After = [ "graphical-session.target" ];
            PartOf = [ "graphical-session.target" ];
        };
        Service = {
            Type = "oneshot";
            ExecStart = ''${brightnessScript}/bin/solar-brightness adjust --min-brightness ${toString cfg.min-brightness} --max-brightness ${toString cfg.max-brightness} --latitude ${toString cfg.location.latitude} --longitude ${toString cfg.location.longitude} --twilight-type ${cfg.twilight-type} --solar-offset "${cfg.solar-offset}" --transition-max-step ${toString cfg.transition.max-step} --transition-step-delay "${cfg.transition.step-delay}"';
        };
    };
};
```

- Import the external Nu file in the Nix store (use an external .nu file for linter support!)
- Call the imported Nu script in the Nix store in ExecStart
- Pass along command-line arguments with string interpolation (use NuShells built-in command line argument parsing!)
- Provide dependencies heliocron for the service and timer (omitted)