

INTRODUCTION

conda install hell

SOLUTION

pixi

- extremely fast
- clever isolation

BIT OF HISTORY

Scripting language support extensions in lower-level languages

For example for Python and R

Python has Cython to do that very easily

Great idea ->

Heavy lifting in typed performant language ->

Easily script around it

Numpy/Pandas/Scipy/Tensorflow do this

DATA SCIENCE BOOM

2010 big boom in data science

However, practically impossible to install numpy on Windows
(pip just arrived in 2008)

That was the hole Anaconda filled with conda packages (2012).

Conda packages provides a way to bring Windows binaries to the masses
(to install alongside pypi packages.)

WINDOWS BINARIES

Bringing Windows binaries to the masses is a big problem.

It still is. No big adopted package manager for Windows.

Still no way to bring binaries cross platform to the masses.*

No greatly adopted package manager
(except maybe apt).

So, conda packages is a great vehicle for shipping binaries.

(*The problem for this is someone has to pay the cost of hosting.)

CONDA GOT ADOPTED TO BIOINFORMATICS

So conda and the likes got also adopted by the *bioinformatics* community.

Because bioinformatics packages are often scripts,

But they rely on C++ standalone programs that do the heavy lifting.

Conda packages could just bring these binaries with you and *manage environments*

CONDA ENVIRONMENTS

Conda environments are great for development.

That is the intended use case.

...

```
conda create -n myenv
```

```
conda activate myenv
```

```
python ...
```

```
conda install ..
```

...

But bioinformatics people want to run the packages as tools!

Putting everything into one environment creates a big mess.

RUNNING BINARIES WITH CONDA

That was never (and still is not) the intended usecase.

``conda run`` works, but it is extremely slow
(conda is slow in so many regards).

Also the new client (mamba, completely written in C!)
just has a very slow ``run`` command.

Only micromamba could provide you with "fast" ``run`` command.
First below a 1s (0.2s)

CONDA RUN IS JUST USELESS

You now have to call ``conda run`` everywhere

Not great.

Any other tool just looking the PATH for a binary fails.

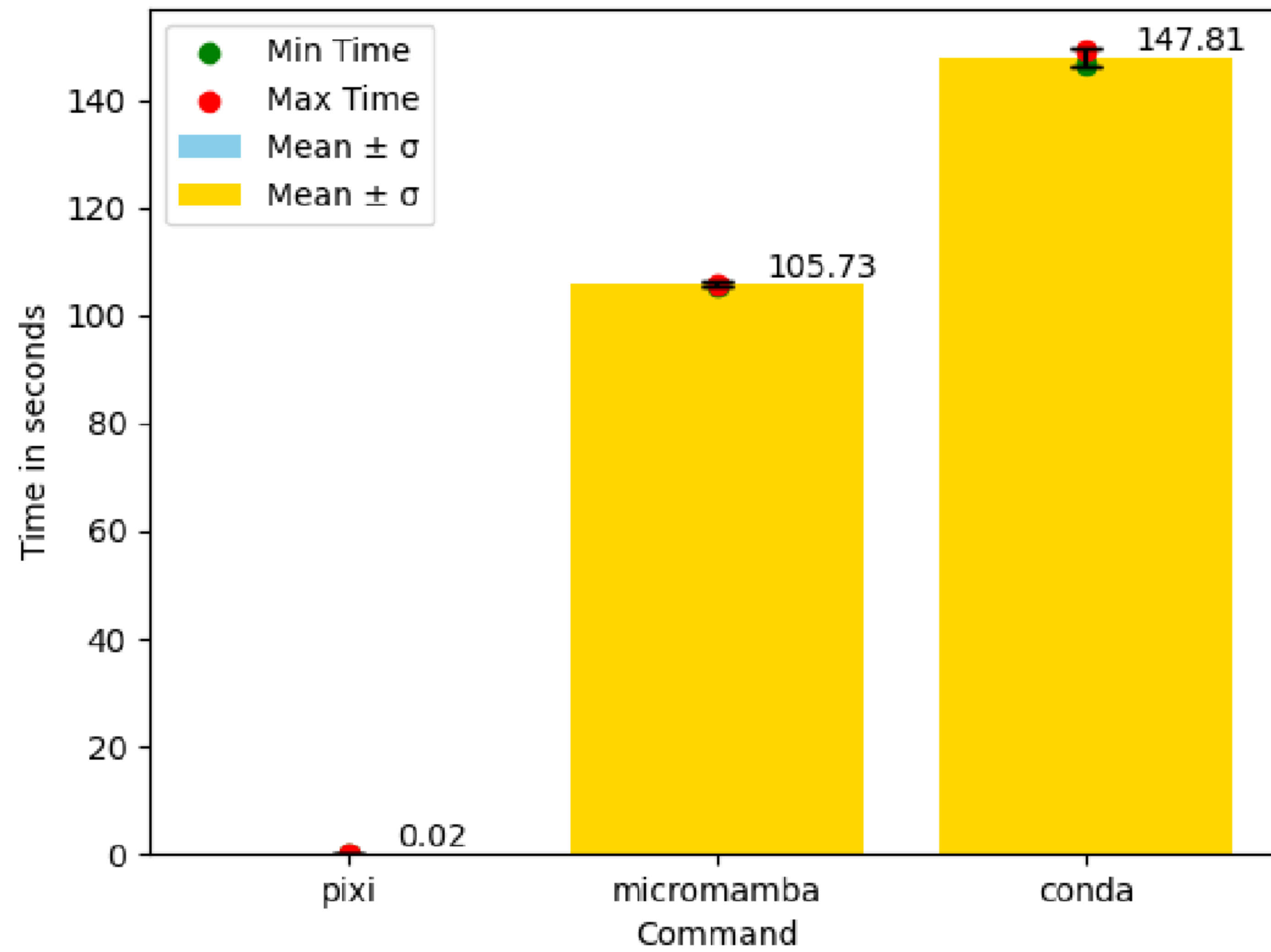
Real solution:

Just use conda environments with Snakemake

NOW WE BRING PIXI TO THE TABLE

- * Fast (written in Rust) (has written its own solvers as well)
- * Project-first, not environment-first
- * A super fast way to run (global) binaries with trampolines
- * Resolves both conda and pypi
- * Legal to use (if you use conda now you are a criminal)

Comparison of Environment Creation Times



PIXI INSTALLATION

```

```
curl -fsSL https://pixi.sh/install.sh | sh
```

```

That is it. A portable binary in your ` \$PATH `.

No base environment. No bootstrapping another conda tool with another conda tool.

TOOL INSTALLATION

```

```
pixi global install <package>
```

```

```

```
pixi global install -c conda-forge -c bioconda snakemake==7.25.4
```

```

THE UGLY

channel-priority is on strict by default and cannot be disabled (considered a security problem).

``pixi global install -c bioconda -c conda-forge`` will make you cry.

bioconda packages will be preferred here,
but are often way too old in comparison to conda-forge.

Just use:

``pixi global install -c conda-forge -c bioconda``

Or just

`pixi config set default-channels '["conda-forge", "bioconda"]'`

The plethora of channels of the conda ecosystem is also a headache sometimes.

Especially considering the fact that there then also exists equivalents which also exist on pypi.

SIMPLE SOLUTION TO TOOL INSTALLATION

Install the tool, and then

```
export PATH="$HOME/.pixi/bin:$PATH"
```

to the top of your script and you are done!

BONUS:

The complete setup is captured in `~/.pixi/manifests/pixi-global.toml`

Can be put on another machine and repeat the whole setup exactly.

ADVANCED USAGE

Adding more dependencies to the environment:

```
pixi global install jupyter --with polars
```

Or later

```
pixi global install -e jupyter polars
```

Misc: Using a fixed PATH (if you are really afraid of module PATH changes)

```
export PIXI_BASE_PATH=$PATH
```

VERSIONS - EXPOSING

```
pixi global install -c conda-forge -c bioconda -e snakemake-9 --expose snakemake-9=snakemake snakemake==9.*  
snakemake-9 --version
```

OR

```
pixi global expose remove snakemake  
pixi global expose add snakemake -e snakemake-9
```

NB. `expose` is pretty slow

THE BAD - SINGLE USER ORIENTED

Problem: Using `pixi global expose`,
we just changed the snakemake version for everybody using this pixi install!

No easy solution, but fear not, I will have a working workflow.

ONTO THE TRAMPOLINE

`~/pixi/bin/program` is where all the programs execution start.

`~/pixi/bin/trampoline_configuration/trampoline_bin` can be hardlinked to anything.

Which then loads the configuration found in `~/pixi/bin/trampoline_configuration/program.json`

This will load the environment configuration and run the main binary in it.

Provided the conda package does nothing weird, this can be considered a working binary from your PATH. No magic required.

This has to be in that place. The binary cannot move. It relies on the directory structure.

You can still make soft links to it.

Trampolines are still a new technology, but should be equivalent to

```
```bash
(
 conda activate myenv
 program ...
)
```
```

NOW: PROPOSAL FOR VERSION MANAGEMENT

Pixi is not replacing module load

First, module load and easybuild sometimes provide a perfectly fine solution

Second, because some packages are just not available on conda.

Third, module load provides the better multi user version managing.

Pixi is just a better way to "install something", better than conda!

A PROBLEMATIC MODULE FILE

multiqc

```

```
prepend_path("PATH", "/appdata/users/service-hgn/miniconda3/envs/multiqc-1.30/bin/")
```

```
prepend_path("PYTHONPATH", "/appdata/users/service-hgn/miniconda3/envs/multiqc-1.30/lib/python3.13/site-packages")
```

```

conda bin folder are filled with other programs

PYTHONPATH is just very problematic

Easily breaks with other modules

```

```
module load multiqc
```

```
module load snakemake
```

```
python -c 'import numpy' # breaks already!
```

```


| | | | | |
|--------------|-------------------|-------------------|-------------------|---------------------------|
| PATH | | | | |
| KaleidoApp | fc-list | libpng16-config | psicc | tiffcrop |
| bunzip2 | fc-match | linkicc | pydoc | tiffdither |
| bzcat | fc-pattern | makeconv | pydoc3 | tiffdump |
| bzcmp | fc-query | markdown-it | pydoc3.13 | tiffinfo |
| bzdiff | fc-scan | markdown_py | pygmentize | tiffmedian |
| bzegrep | fc-validate | mathjax-path | python | tiffset |
| bzfgrep | freetype-config | multiqc | python3 | tiffsplit |
| bzgrep | genbrk | natsort | python3-config | tificc |
| bzip2 | gencfu | ncurses6-config | python3.1 | tjbench |
| bzip2recover | gencnval | ncursesw6-config | python3.13 | toe |
| bzless | gendict | normalizer | python3.13-config | tput |
| bzmore | genrb | nspr-config | raw2tiff | tqdm |
| c_rehash | humanfriendly | nss-config | rdjpgcom | transicc |
| captoinfo | icu-config | numpy-config | reset | tset |
| certutil | icuexportdata | openssl | rich-click | unzstd |
| cjpeg | icuinfo | opj_compress | sqlite3 | wish |
| clear | idle3 | opj_decompress | sqlite3_analyzer | wish8.6 |
| coloredlogs | idle3.13 | opj_dump | tabs | wrjpgcom |
| derb | infocmp | pal2rgb | tclsh | x86_64-conda-linux-gnu-ld |
| djpeg | infotocap | pip | tclsh8.6 | xmlwf |
| dotenv | jpegtran | pip3 | tic | zstd |
| f2py | jpgicc | pk12util | tiff2bw | zstdcat |
| fax2ps | jsonschema | pkgdata | tiff2pdf | zstdgrep |
| fax2tiff | kaleido | plotly_get_chrome | tiff2ps | zstdless |
| fc-cache | libdeflate-gunzip | png-fix-itxt | tiff2rgba | zstdmt |
| fc-cat | libdeflate-gzip | pngfix | tiffcmp | |
| fc-conflist | libpng-config | ppm2tiff | tiffcp | |

| | | |
|---|--|------------------------------------|
| PYTHONPATH | | |
| MarkupSafe-3.0.2.dist-info | hyperframe-6.1.0.dist-info | pydantic |
| PIL | idna | pydantic-2.11.7.dist-info |
| PySocks-1.7.1.dist-info | idna-3.10.dist-info | pydantic_core |
| PyYAML-6.0.2.dist-info | importlib_metadata | pydantic_core-2.33.2.dist-info |
| README.txt | importlib_metadata-8.7.0.dist-info | pygments |
| __pycache__ | jinja2 | pygments-2.19.2.dist-info |
| _brotli.cpython-313-x86_64-linux-gnu.so | jinja2-3.1.6.dist-info | python_dotenv-1.1.1.dist-info |
| _cffi_backend.cpython-313-x86_64-linux-gnu.so | jsonschema | referencing |
| _plotly_utils | jsonschema-4.25.0.dist-info | referencing-0.36.2.dist-info |
| _yaml | jsonschema_specifications | regex |
| annotated_types | jsonschema_specifications-2025.4.1.dist-info | regex-2024.11.6.dist-info |
| annotated_types-0.7.0.dist-info | kaleido | requests |
| attr | kaleido-0.2.1.dist-info | requests-2.32.4.dist-info |
| attrs | markdown | rich |
| attrs-25.3.0.dist-info | markdown-3.8.2.dist-info | rich-14.1.0.dist-info |
| brotli-1.1.0.dist-info | markdown_it | rich_click |
| brotli.py | markdown_it_py-3.0.0.dist-info | rich_click-1.8.9.dist-info |
| certifi | markupsafe | rpds |
| certifi-2025.7.14.dist-info | mdurl | rpds_py-0.26.0.dist-info |
| cffi | mdurl-0.1.2.dist-info | socks.py |
| cffi-1.17.1.dist-info | multiqc | sockshandler.py |
| charset_normalizer | multiqc-1.30.dist-info | spectra |
| charset_normalizer-3.4.2.dist-info | narwhals | spectra-0.0.11.dist-info |
| click | narwhals-2.0.0.dist-info | tiktoken |
| click-8.2.1.dist-info | natsort | tiktoken-0.9.0.dist-info |
| colorama | natsort-8.4.0.dist-info | tiktoken_ext |
| colorama-0.4.6.dist-info | networkx | tqdm |
| coloredlogs | networkx-3.5.dist-info | tqdm-4.67.1.dist-info |
| coloredlogs-15.0.1.dist-info | numpy | typeguard |
| coloredlogs.pth | numpy-2.3.2.dist-info | typeguard-4.4.4.dist-info |
| colormath | packaging | typing_extensions-4.14.1.dist-info |
| colormath-3.0.0.dist-info | packaging-25.0.dist-info | typing_extensions.py |
| conda-site.pth | pillow-11.3.0.dist-info | typing_inspection |
| dotenv | pip | typing_inspection-0.4.1.dist-info |
| h2 | pip-25.1.1.dist-info | urllib3 |
| h2-4.2.0.dist-info | plotly | urllib3-2.5.0.dist-info |
| hpack | plotly-6.2.0.dist-info | yaml |
| hpack-4.1.0.dist-info | polars | zipp |
| humanfriendly | polars-1.31.0.dist-info | zipp-3.23.0.dist-info |
| humanfriendly-10.0.dist-info | pyaml_env | zstandard |
| humanize | pyaml_env-1.2.2.dist-info | zstandard-0.23.0-py3.13.egg-info |
| humanize-4.12.3.dist-info | pycparser | |
| hyperframe | pycparser-2.22.dist-info | |

A BETTER WAY TO DO THIS

```

```
pixi global install multiqc
```

```
ln -s ~/.pixi/bin/multiqc /appdata/bin/1.30/multiqc
```

```

OR, even better

```

```
pixi global install -e multiqc-1.30 --expose multiqc-1.30=multiqc multiqc
```

```
ln -s ~/.pixi/bin/multiqc-1.30 /appdata/apps/multiqc/1.30/multiqc
```

```

(exposing the versions directly already)

NOW THE MODULE FILE BECOMES:

```
```\nprepend_path("PATH", "/appdata/apps/multiqc/1.30/")\n```
```

It just works! `multiqc` (and only `multiqc`)  
with the correct version is now added to the path.

I believe that this is intended usage for modules also:

Make a single binary available in a versioned folder (just as easybuild).

NB. Keep in mind permissions of course!

Setting `\$\_PIXI\_HOME` to something outside of a home folder might help.



## MODULE LOAD IS HERE TO STAY

but Pixi is still a full replacement for:

`conda`, `miniconda`, `mamba`, `micromamba`,

`pip`, `poetry`, `uv`, `pdm`, `hatch`, `rey`

`venv`, `virtualenv`, `pipenv`, `pyenv`,

`conda-build`, `setuptools`, `flit`,

and also task running like `npm run` and what not.

## PIXI WITH SNAKEMAKE

You still have to use conda with snakemake to activate the environments

Supporting pixi in snakemake is a work in progress.

You can however still just install conda with pixi.

TLDR

Conda packages from conda-forge and bioconda are great!

The conda client is not great.

Pixi is the next-gen replacement for conda and is great!

Especially since, conda was never used for tool installation, pixi has a system

Module load still provides the better multi-user versioned system, and familiar

Pixi is just a quick and sane package manager when easybuild fails you

Next up: Sane next-gen dependency management and task running for projects with pixi



**BRAINSTORMING**

But I have some more slides as well!

PS. For those that are on the terminal a lot and like speed and ergonomics

Rewrite it in rust (RIIR):

grep -> ripgrep (rg)

find -> fd-find (fd)

cd -> zoxide (z)

uv -> pip (uv included in pixi)

sed -> sd

I practically use all of them

Best of all: Pixi can install most of them!

More RIIR (which I do not use)

du -> dust/dua

tmux -> zellij

ps -> procs

make -> just (task runner, but pixi comes also with a task runner)

Source: <https://github.com/j-m-hoffmann/awesome-rewrite-it-in-rust>

And some more modern tools I use

man -> tldr (tealdeer is the rust implementation)

du -> ncdu

? -> fzf (fuzzy finder, there is also skim in rust)

And of course let's not forget:

vim -> neovim