uutils coreutils

& the quest for compatibility

whoami

- Terts Diepraam
- Recent graduate of TU Delft
- A maintainer on uutils since 2021
- I love programming languages! (And Rust of course)

uutils coreutils

Cross-platform reimplementation of the coreutils in Rust

GNU Coreutils

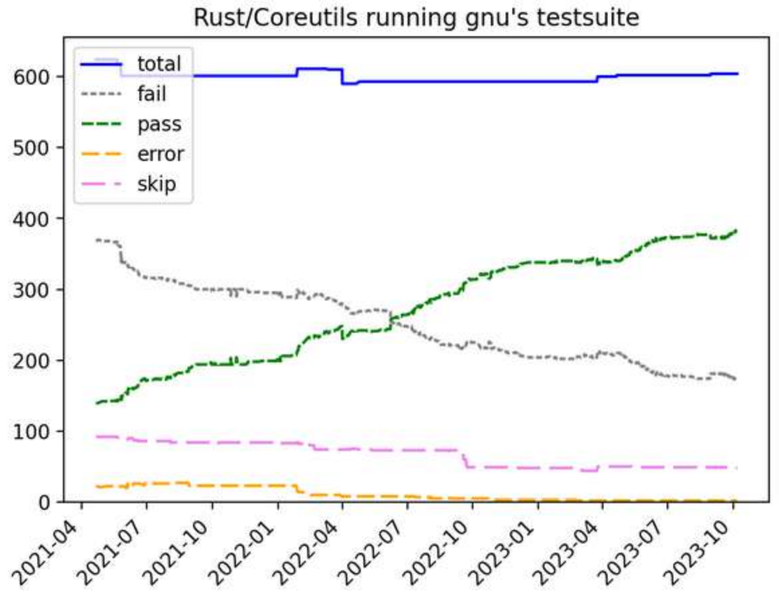
`arch`	`b2sum`	`base`	`basename`	`basenc`	`cat`	`chgrp`	`chmod`	`chown`
`chroot`	`cksum`	`comm`	`cp`	`csplit`	`cut`	`date`	`dd`	`df`
`dir`	`dircolors`	`dirname`	`du`	`echo`	`env`	`expand`	`expr`	`factor`
`false`	`fmt`	`fold`	`groups`	`hashsum`	`head`	`hostid`	`hostname`	`id`
`install`	`join`	`kill`	`link`	`ln`	`logname`	`ls`	`md5sum`	`mkdir`
`mkfifo`	`mknod`	`mktemp`	`more`	`mv`	`nice`	`nl`	`nohup`	`nproc`
`numfmt`	`od`	`paste`	`pathchk`	`pinky`	`pr`	`printenv`	`printf`	`ptx`
`pwd`	`readlink`	`realpath`	`relpath`	`rm`	`rmdir`	`seq`	`sha1sum`	`sha224sum`
`sha256sum`	`sha384sum`	`sha512sum`	`shred`	`shuf`	`sleep`	`sort`	`split`	`stat`
`stdbuf`	`stty`	`sum`	`sync`	`tac`	`tail`	`tee`	`test`	`timeout`
`touch`	`tr`	`true`	`truncate`	`tsort`	`tty`	`uname`	`unexpand`	`uniq`
`unlink`	`uptime`	`users`	`vdir`	`wc`	`who`	`whoami`	`yes`	

- Set of utilities you probably have on your system
- "Standard library of Bash"
- Written in C
- Maintained by the FSF

A brief history of uutils

- Started in August of 2013 by Jordi Boggiano
- 2 years before Rust 1.0!
- Several maintainers over the years
- 2020: Sylvestre Ledru put focus on GNU compatibility
- Active maintainers: Sylvestre Ledru, Daniel Hofstetter and me

- No compatibility? No chance of adoption
- GNU is most familiar
- Every quirk is used by *someone*



The problem:

```
C != Rust
```

The language determines the implementation

Case Study 1

Error Handling

GNU coreutils deal with errors in two ways:

1. Exit immediately
2. Store exit code for later

Error are usually printed immediately

- `Result`
- It's great to know which functions can fail!
- `?` is great too!
- Exit: bubble up the `Err`
- Store exit code: ???

Requirements for our solution

- In the style of Rust
- Control of exit code
- With convenient API for the GNU-style error handling
- Must work well with IO errors

The uutils error trait: exit code

```rust
trait UError: Error + Send {
    fn code(&self) -> i32 { ... }
}

type UResult<T> = Result<T, Box<dyn UError>>;
```

```
std::fs::copy(a, b).map_err_context(|| {
    format!("cannot copy {} to {}", a.quote(), b.quote())
});
```

(Inspired by `anyhow` and `eyre`, of course)

Storing exit code

```
// Prints the error and stores the exit code
show!(some_uerror);
```

It's not very idiomatic but it works really well for our use case

- `Result` is great!
- Error handling in Rust requires more "design"
- Don't be afraid to make custom error handling
- Practicality over purity

Case Study 2

Argument Parsing

```
while ((c = getopt(argc, argv, "ab:")) != -1) {
  switch (c) {
    case 'a':
      aflag = 1;
      break;
    case 'b':
      bvalue = optarg;
      break;
    case '?':
    default:
      abort();
    }
}
```

There's a lot not to like there

- `-1` as error value
- Specifying arguments as a string
- Modifying globals
- No exhaustiveness checking
- Error messages not automatic
- etc.

- `clap` is great!
- Automatic `--help`, error messages, completions, etc.
- `clap` "collects" all results into a final data structure
- Builder API: Into a `HashMap`
- Derive API: Into custom `struct`

- We use `clap` at the moment
- Though the derive API is not flexible enough
- Sometimes it is hard

The `-o` flag does two things:

- Use the `-l` format,
- Hide the group.

Setting `-o -C -l` is the same as `-o`

It is *partially overridden* by other arguments!

This is possible but very hard in `clap`

Other problems

```
head -10 some_file.txt  # clap doesn't have the -N
seq -s= 10              # clap uses = as separator
# etc.
```

Small incompatibilities sneak in

And workarounds lead to overcomplicated code!

Note: *none* of this is `clap`'s fault! It's a great library, it's just not made for coreutils.

A new argument parser (WIP)

- Solution: custom argument parser!
- Derive not on a `struct` but an `enum`
- Mirrors the *structure* of `getopt`
- So the default behaviour is the same

Small preview (sample from ls)

```rust
#[derive(Arguments)]
enum Arg {
    /// Do not ignore entries starting with .
    #[option("-a")]
    All,

    /// Sort by WORD
    #[option("--sort=WORD")]
    #[option("-t", default = Sort::Time, help = "Sort by time")]
    Sort(Sort),
}

struct Settings {
    which_files: Files,
    sort: Sort,
}
```

- Defaults matter!
- We can take the best ideas from other libraries
- And apply them in our own way

- We care about compatibility!
- But compatibility is hard!

Links

Come talk to me!

- During the break
- @tertsdiepraam on GitHub
- @terts@mastodon.online

Links to uutils

- uutils.github.io
- uutils/coreutils on GitHub
- uutils Discord