Crate gitoxide

This is the documentation of the binaries that come with `gitoxide`. These are called…

`gix`

A developer tool to allow using `gitoxide` algorithms and functionality outside of the test suite. It will be unstable as long as the `gix` crate is unstable and is explicitly not to be understood as `git` replacement.

`ein`

A program to eventually become the most convenient way to do typical operations on `git` repositories, with all tooling one typically needs built right into it. For now, it's most useful for its assorted set of `tools` which help to build automations or learn something about `git` repositories.

Feature Flags

Feature configuration can be complex and this document seeks to provide an overview.

Build Configuration

These combine common choices of building blocks to represent typical builds.

- **`max`** *(enabled by default)* — Everything, all at once.

  As fast as possible, with TUI progress, progress line rendering with auto-configuration, all transports based on their most mature implementation (HTTP), all `ein` tools, CLI colors and local-time support, JSON output, regex support for rev-specs.
- **`max-pure`** — Like `max`, but only Rust is allowed.

  This is the most compatible build as it won't need a C compiler or C toolchains to build. It's also not the fastest as or the most feature-rich in terms of available transports as it uses Rust's HTTP implementation.

  As fast as possible, with TUI progress, progress line rendering with auto-configuration, all transports available but less mature pure Rust HTTP implementation, all `ein` tools, CLI colors and local-time support, JSON output, regex support for rev-specs.
- **`max-control`** — Like `max`, but with more control for configuration. See the *Package Maintainers* headline for more information.
- **`lean`** — All of the good stuff, with less fanciness for smaller binaries.

  As fast as possible, progress line rendering, all transports based on their most mature implementation (HTTP), all `ein` tools, CLI colors and local-time support, JSON output.
- **`small`** — The smallest possible build, best suitable for small single-core machines.

  This build is essentially limited to local operations without any fanciness.

  Optimized for size, no parallelism thus much slower, progress line rendering.
- **`lean-async`** — Like lean, but uses Rusts async implementations for networking.

  This build is more of a demonstration showing how async can work with `gitoxide`, which generally is blocking. This also means that the selection of async transports is very limited to only HTTP (without typical `git` configuration) and git over TCP like provided by the `git daemon`.

  As fast as possible, progress line rendering, less feature-ful HTTP (pure Rust) and only `git-damon` support, all `ein` tools, CLI colors and local-time support, JSON output.

  Due to async client-networking not being implemented for most transports, this one supports only the 'git+tcp' and HTTP transport. It uses, however, a fully asynchronous networking implementation which can serve a real-world example on how to implement custom async transports.

Package Maintainers

These features are meant to mimic the normal build configurations, but leave it to you to configure C libraries, involving choices for `zlib`, hashing and HTTP implementation.

Additional features *can* be provided with `--features` and are handled by the `gix-features` crate. If nothing else is specified, the Rust implementation is used. Note that only one feature of each section can be enabled at a time.

- **zlib**
  - `gix-features/zlib-ng`
  - `gix-features/zlib-ng-compat`
  - `gix-features/zlib-stock`
  - `gix-features/zlib-rust-backend` (*default if no choice is made*)
- **sha1**
  - `gix-features/fast-sha1`
  - `gix-features/rustsha1` (*default if no choice is made*)
- **HTTP** - see the *Building Blocks for mutually exclusive networking* headline

Building Blocks

Typical combinations of features of our dependencies, some of which are referred to in the `gitoxide` crate's code for conditional compilation.

- **`fast`** — Makes the crate execute as fast as possible by supporting parallel computation of otherwise long-running functions as well as fast, hardware accelerated hashing, along with a faster zlib backend. If disabled, the binary will be visibly smaller.
- **`fast-safe`** — Makes the crate execute as fast as possible by supporting parallel computation of otherwise long-running functions as well as fast, hardware accelerated hashing, along with a faster zlib backend. If disabled, the binary will be visibly smaller.
- **`pretty-cli`** — Use `clap` 3.0 to build the prettiest, best documented and most user-friendly CLI at the expense of binary size. Provides a terminal user interface for detailed and exhaustive progress. Provides a line renderer for leaner progress display, without the need for a full-blown TUI.
- **`prodash-render-line-crossterm`** — The `--verbose` flag will be powered by an interactive progress mechanism that doubles as log as well as interactive progress that appears after a short duration.
- **`prodash-render-tui`** — Progress reporting with a TUI, can then be enabled with the `--progress` flag.
- **`prodash-render-line`** — Progress reporting by visually drawing lines into the terminal without switching to an alternate window.
- **`cache-efficiency-debug`** — Prints statistical information to inform about cache efficiency when those are dropped. Use this as a way to understand if bigger caches actually produce greater yiedls.
- **`gitoxide-core-tools`** — A way to enable most `gitoxide-core` tools found in `ein tools`, namely `organize` and `estimate hours`.
- **`gitoxide-core-tools-query`** — A program to perform analytics on a `git` repository, using an auto-maintained sqlite database

Building Blocks for mutually exclusive networking

Blocking and async features are mutually exclusive and cause a compile-time error. This also means that `cargo … --all-features` will fail. Within each section, features can be combined.

Blocking

The backends are mutually exclusive, e.g. choose either `curl` or `request`.

- **`gitoxide-core-blocking-client`** — Use blocking client networking.
- **`http-client-curl`** — Support synchronous 'http' and 'https' transports (e.g. for clone, fetch and push) using **curl**.
- **`http-client-reqwest`** — Support synchronous 'http' and 'https' transports (e.g. for clone, fetch and push) using **reqwest**.

Async

- **`gitoxide-core-async-client`** — Use async client networking.