Referensi API libvctrl@2.2.1
Untuk Pengembang Integrasi dan Backend

Muhammad Riduwan Khafidi
Program Studi Teknik Rekayasa Perangkat Lunak
Politeknik Negeri Batam
4342611034@students.polibatam.ac.id

November 5, 2026

Abstrak

libvctrl adalah workspace Rust untuk membangun sistem kendali versi yang dapat dikustomisasi. Dokumen ini menyajikan spesifikasi API publik workspace untuk pengembang yang akan memakai, memperluas, atau mengimplementasikan ulang kontraknya. Cakupan meliputi lapisan kontrak, implementasi referensi, primitif kriptografi, fasad, plumbing, dan backend penyimpanan. Setiap API dijelaskan melalui tanda tangan fungsi, batasan tipe, kondisi error, dan kontrak thread safety. Dokumen memuat tujuh visualisasi TikZ, dua belas tabel spesifikasi, dan blok kode yang dapat dikompilasi terhadap crate versi yang tercantum. Pengembang memperoleh referensi presisi untuk integrasi dan ekstensi tanpa menebak perilaku runtime.

1 Pendahuluan

1.1 Motivasi Pengembang

Seorang pengembang yang membangun sistem kendali versi kustom menghadapi empat keputusan yang tidak boleh salah: bagaimana objek direpresentasikan, bagaimana objek diserialisasi, bagaimana objek di-address secara kriptografis, dan bagaimana objek disimpan. libvctrl memisahkan keempat keputusan itu ke dalam empat lapisan yang dapat dipilih secara independen. Lapisan kontrak mendefinisikan representasi. Lapisan implementasi referensi mendefinisikan serialisasi dan addressing. Lapisan kriptografi menyediakan primitif. Lapisan backend mendefinisikan penyimpanan.

Pemisahan ini hanya bernilai jika pengembang dapat melihat batas setiap lapisan tanpa membaca kode sumber. Dokumen ini menyajikan batas itu secara eksplisit: tanda tangan fungsi lengkap dengan bound generik, kondisi error yang mungkin, dan invariant yang dijamin oleh konstruktor.

1.2 Audiens dan Asumsi

Dokumen ini ditujukan untuk pengembang Rust dengan pemahaman tentang trait, generik, lifetime, dan operator `?`. Dokumen mengasumsikan pembaca pernah memakai `cargo`, mema-

hami perbedaan antara `std::io::Read` dan `std::io::BufRead`, serta mengenal pola builder. Dokumen tidak mengajarkan dasar Rust. Dokumen tidak mengajarkan dasar sistem kendali versi.

1.3 Peta Pembahasan

Bagian kedua memetakan arsitektur workspace dan kontrak thread safety. Bagian ketiga membahas tipe dan trait kontrak. Bagian keempat membahas format biner byte-level. Bagian kelima membahas implementasi referensi. Bagian keenam membahas primitif kriptografi. Bagian ketujuh membahas fasad dan plumbing. Bagian kedelapan membahas template implementasi backend kustom. Bagian kesembilan menyajikan studi kasus end-to-end. Bagian kesepuluh merangkum kesalahan umum, dan bagian kesebelas memberikan lembar rujukan cepat.

2 Arsitektur Workspace

2.1 Graf Ketergantungan

Workspace libvctrl disusun satu arah. `libvctrl_handler` tidak bergantung pada crate lain di workspace. `libvctrl_core` bergantung pada `libvctrl_handler` dan `libvctrl_sha512`. Fasad `libvctrl` mengekspor ulang ketiganya. `libvctrl_plumbing` dibangun di atas `libvctrl_core` dan `libvctrl`. Backend seperti `libvctrl_monumentum` mengimplementasikan trait dari `libvctrl_handler` dan bergantung pada engine penyimpanan eksternal.

Gambar 1 menunjukkan arah ketergantungan. Panah berarti “bergantung pada”.

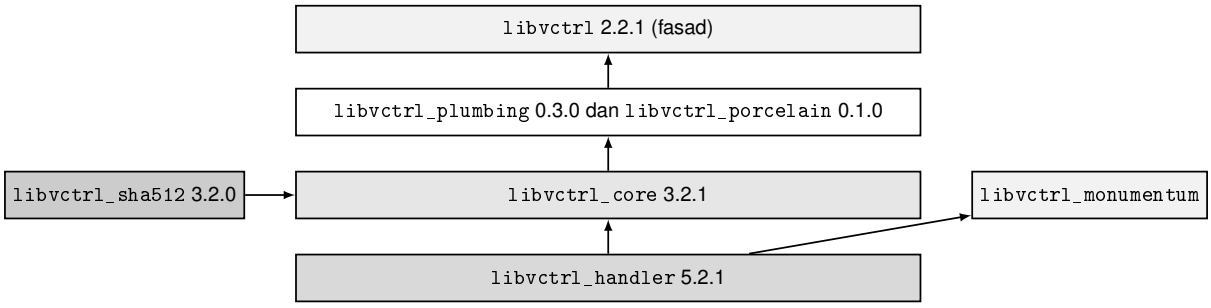


Figure 1. Arah ketergantungan antarcrate dalam workspace libvctrl.

2.2 Tabel Crate

Tabel 1 merangkum versi, peran, MSRV, dan lisensi setiap crate.

Table 1. Metadata crate pada workspace libvctrl.

Crate	Versi	Peran	MSRV	Lisensi
`libvctrl`	2.2.1	Fasad re-export	1.96.0	MIT
`libvctrl_handler`	5.2.1	Kontrak, tipe, validasi	1.96.0	MIT
`libvctrl_core`	3.2.1	Codec, builder, store	1.96.0	MIT
`libvctrl_sha512`	3.2.0	SHA-512, HMAC, HKDF	1.96.0	ISC
`libvctrl_plumbing`	0.3.0	Operasi perintah	1.96.0	MIT
`libvctrl_porcelain`	0.1.0	API tingkat tinggi	1.96.0	MIT
`libvctrl_monumentum`	0.1.0	Backend Monumentum	1.96.0	MIT

2.3 Kontrak Thread Safety

Seluruh trait operasi pada `libvctrl_handler` mendeklarasikan supertrait `Send + Sync`. Konsekuensinya: implementasi backend dapat dipakai lintas thread tanpa pembungkus tambahan. Operasi baca menerima `&self`; operasi tulis menerima `&mut self`. Tipe data immutable seperti `Hash`, `Blob`, dan `Tree` secara otomatis `Send + Sync` karena hanya memuat `Vec`, `String`, dan array.

Gambar 2 menunjukkan aliran bound dari trait ke tipe turunan.

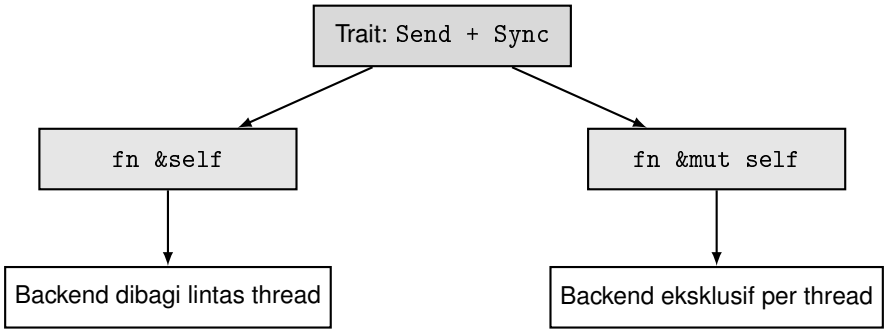


Figure 2. Bound `Send + Sync` pada trait menentukan pola pemakaian backend.

3 Kontrak: libvctrl_handler

3.1 Tipe Data dan Tanda Tangan Konstruktor

Semua tipe data bersifat immutable setelah konstruksi. Konstruktor bersifat fallible: input divalidasi sebelum objek dibuat. Tabel 2 mencantumkan tanda tangan konstruktor dan accessor kunci.

Table 2. Tanda tangan konstruktor dan accessor tipe kontrak.

Tipe	Konstruktor	Accessor kunci
`Hash`	`from_bytes(&[u8]) -> Result<Self, _>`	`as_bytes() -> &[u8; 64]`
`Blob`	`new(Vec<u8>) -> Result<Self, _>`	`data() -> &[u8]`, `size() -> usize`
`TreeEntry`	`new(String, EntryKind, Hash)`	`name()`, `kind()`, `hash()`
`Tree`	`new(Vec<TreeEntry>)`	`entries() -> &[TreeEntry]`, `get(&str)`
`CommitMeta`	`new(i64, i16, Option<String>)`	`timestamp()`, `timezone_offset()`
`Commit`	`new(Hash, Vec<Hash>, UserID, UserID, String)`	`tree()`, `parents()`, `author()`
`Tag`	`new(String, Hash, Option<UserID>, String)`	`name()`, `target()`, `tagger()`
`UserID`	`new(String, String) -> Result<Self, _>`	`name() -> &str`, `email() -> &str`
`ReflogEntry`	`new(Option<Hash>, Option<Hash>, String, i64, i16)`	`old_id()`, `new_id()`, `reason()`

Konstruktor `Commit::with_meta` dan `Tag::with_meta` menerima `CommitMeta` eksplisit. Keduanya dipakai ketika encoding atau timestamp perlu dikendalikan. Konstruktor `Commit::new` dan `Tag::new` memakai `CommitMeta::default()`.

3.2 Trait Operasi

Tabel 3 mencantumkan tanda tangan trait operasi yang harus diimplementasikan backend atau dipakai oleh alat.

Table 3. Tanda tangan trait operasi.

Trait	Tanda tangan metode utama
`Encoder`	`fn encode_blob<W: Write+Send>(&self, &Blob, &mut W) -> Result<(), VctrlError>`
`Decoder`	`fn decode_blob<R: Read+Send>(&self, R) -> Result<Blob, VctrlError>`
`Hasher`	`fn hash<R: Read+Send>(&self, R) -> Result<Hash, VctrlError>`
`ObjectStore`	`put(&mut self, &Hash, &[u8])`, `get(&self, &Hash) -> Result<Box<dyn Read+Send+'_>, _>`
`RefStore`	`set_ref(&mut self, &str, &Hash)`, `get_ref(&self, &str) -> Result<Hash, _>`
`Index`	`add(&mut self, Self::Entry)`, `write_tree(&self) -> Result<Self::TreeId, _>`
`Blame`	`blame_file(&self, &str) -> Result<Vec<BlameEntry>, _>`
`RevWalk`	`walk(&self, &Self::CommitId) -> Result<RevWalkIterator<Self::CommitId>, _>`
`TreeDiffer`	`diff_trees(&self, &Self::TreeId, &Self::TreeId) -> Result<TreeDelta, _>`
`Remote`	`fetch(&mut self, &[Self::RefSpec])`, `push(&mut self, &[Self::RefSpec])`
`Transport`	`fetch_object(&self, &Hash)`, `push_object(&mut self, &Hash, &[u8])`
`PackReader`	`read_object(&self, &Self::ObjectId) -> Result<Box<dyn Read+Send+'_>, _>`
`PackWriter`	`write_object(&mut self, &Self::ObjectId, &[u8])`
`Signer`	`sign(&mut self, &str, &[u8]) -> Result<Vec<u8>, _>`
`Verifier`	`verify(&self, &str, &[u8], &[u8]) -> Result<bool, _>`
`ConfigStore`	`get_string(&self, &str, &str) -> Result<Option<String>, _>`
`ReflogStore`	`append(&mut self, &Self::RefName, Option<Hash>, Option<Hash>, &str, i64, i16)`

Setiap trait memiliki bound `Send + Sync`. Implementasi backend wajib memenuhi bound tersebut. Untuk backend yang memakai `Arc<Mutex<E>>`, bound terpenuhi otomatis selama `E`:

`Send`.

3.3 Konstanta Sistem

Tabel 4 mencantumkan konstanta dan nilainya. Konstanta ini dipakai oleh konstruktor untuk menolak input yang melebihi batas.

Table 4. Konstanta sistem dan nilai default.

Konstanta	Nilai	Dipakai oleh
`HASH_LENGTH`	64	`Hash::from_bytes`, `Hash::as_bytes`
`MAX_NAME_LENGTH`	255	`UserID`, `validate_name`
`MAX_BLOB_SIZE`	100×1024^2	`Blob::new`
`MAX_TREE_ENTRIES`	100 000	`Tree::new`
`MAX_MESSAGE_LENGTH`	1024×1024	`Commit`, `Tag`
`MAX_PARENT_COUNT`	65 535	`Commit::new`

3.4 Varian Error

Semua operasi fallible mengembalikan `VctrlError`. Varian `IoError` membungkus `Arc<std::io::Error>` agar tetap `Clone`. Tabel 5 mencantumkan pemicu setiap varian.

Table 5. Varian `VctrlError` dan pemicunya.

Varian	Pemicu
`CorruptedData(String)`	Byte stream tidak sesuai format, UTF-8 invalid, atau trailing bytes
`DuplicateParent`	Vektor parent memuat hash yang sama lebih dari sekali
`ExceededMaxSize(String)`	Panjang melebihi `MAX_*`
`InvalidBlameRange`	`start_line == 0` atau `line_count == 0`
`InvalidEmail(String)`	Email kosong, tanpa `@`, terlalu panjang, atau memuat karakter kontrol
`InvalidHashLength(usize)`	Panjang byte bukan 64
`InvalidName(String)`	Nama kosong, terlalu panjang, atau memuat karakter kontrol
`InvalidTimezoneOffset(i16)`	Nilai di luar rentang `-1440..=1440`
`InvalidTreeStructure(String)`	Entri tree duplikat
`IoError(Arc<io::Error>)`	Kegagalan `Read` atau `Write`
`ObjectNotFound(Hash)`	Hash tidak ada di `ObjectStore`
`Other(String)`	Kesalahan yang tidak tercakup varian lain
`RefNotFound(String)`	Nama ref tidak ada di `RefStore`
`SerializationError(String)`	Panjang melebihi representasi prefix integer

3.5 Validasi Input

Fungsi validasi murni memeriksa input sebelum konstruktor dipanggil. Tabel 6 mencantumkan fungsi dan aturannya.

Table 6. Fungsi validasi dan aturannya.

Fungsi	Aturan
`validate_hash_bytes`	Panjang harus tepat `HASH_LENGTH`
`validate_name`	Tidak kosong, $\leq$ `MAX_NAME_LENGTH`, tanpa karakter kontrol
`validate_ref_name`	`validate_name` plus tanpa `/`, `..`, `@{`, dan karakter khusus Git
`validate_tree_entry_name`	`validate_name` plus tanpa `/`, `\`, `.`, `..`

Gambar 3 menunjukkan alur dari input mentah sampai objek immutable.

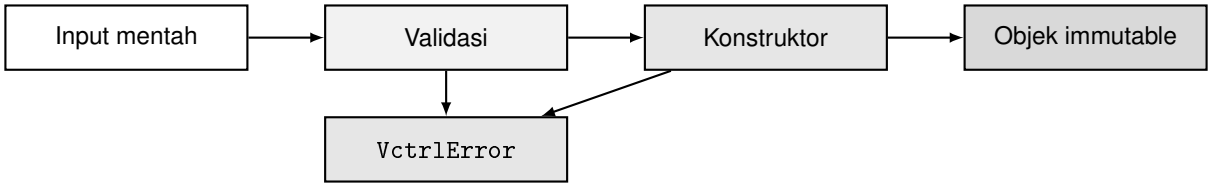


Figure 3. Alur validasi dan konstruksi objek kontrak.

4 Format Biner Byte-Level

4.1 Spesifikasi Format

`BinaryEncoder` menulis byte stream dengan layout deterministik. Semua integer little-endian. Panjang string diawali `u8`. Version byte saat ini bernilai `3`. Tabel 7 mencantumkan layout per tipe.

Table 7. Layout byte-level format biner versi 3.

Tipe	Layout
`Blob`	`u8 version | u64 LE data_len | data[data_len]`
`Tree`	`u8 version | u32 LE count | entry[count]`
`entry`	`u8 name_len | name[name_len] | u8 kind | u8 hash[64]`
`Commit`	`u8 version | u8 tree[64] | u16 LE parent_count`
	`| parent[parent_count][64] | u8 author_name_len`
	`| author_name | u8 author_email_len | author_email`
	`| u8 committer_name_len | committer_name`
	`| u8 committer_email_len | committer_email`
	`| u32 LE msg_len | msg | i64 LE timestamp`
	`| i16 LE tz_offset | u8 enc_len | enc`
`Tag`	`u8 version | u8 name_len | name`
	`| u8 target[64] | u8 has_tagger | [tagger]`
	`| u32 LE msg_len | msg | i64 LE timestamp`
	`| i16 LE tz_offset | u8 enc_len | enc`

Nilai `kind` pada entri tree: `0` untuk `Blob`, `1` untuk `Executable`, `2` untuk `Symlink`, `3` untuk `Tree`, `4` untuk `Submodule`. Nilai `has_tagger` pada tag: `0` untuk absen, `1` untuk hadir.

Gambar 4 menunjukkan layout blob dan tree secara visual.

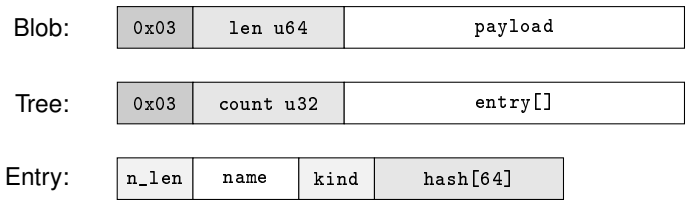


Figure 4. Layout byte-level untuk `Blob`, `Tree`, dan `TreeEntry`.

4.2 Batasan Decoder

`BinaryDecoder` menerapkan pertahanan berlapis. Input dibaca dengan `read_bounded` yang memakai buffer 4 KiB dan menolak alokasi melebihi batas per tipe. Setiap offset diperiksa sebelum slicing. Setiap string divalidasi UTF-8. Setiap nilai numerik dikonversi dengan `try_from` sehingga overflow terdeteksi sebelum alokasi. Batas `MAX_*` diperiksa ulang setelah konversi.

Decoder menolak: version byte yang bukan `3`, panjang yang tidak cocok, trailing bytes setelah payload, kind byte yang tidak dikenal, dan tagger presence byte yang bukan `0` atau `1`.

5 Implementasi Referensi: libvctrl_core

5.1 BinaryEncoder dan BinaryDecoder

Keduanya tipe zero-sized. `BinaryEncoder` mengimplementasikan `Encoder`. `BinaryDecoder` mengimplementasikan `Decoder`. Pemakaian ditunjukkan pada blok kode berikut.

```
use std::io::Cursor;
use libvctrl_core::codec::{BinaryDecoder, BinaryEncoder};
use libvctrl_handler::{Blob, Decoder, Encoder, VctrlError};

fn roundtrip() -> Result<(), VctrlError> {
    let blob = Blob::new(b"halo".to_vec())?;
    let mut buf = Vec::new();
    BinaryEncoder.encode_blob(&blob, &mut buf)?;
    let out = BinaryDecoder.decode_blob(Cursor::new(buf))?;
    assert_eq!(out.data(), blob.data());
    Ok(())
}
```

5.2 Sha512Hasher

`Sha512Hasher` adalah adapter zero-sized dari `libvctrl_sha512::Hash` ke `Hasher`. Tipe ini membaca input dalam chunk 4 KiB dan mengembalikan `Hash` 64 byte. Pemakaian ditunjukkan pada blok kode berikut.

```
use libvctrl_core::hash::Sha512Hasher;
use libvctrl_handler::{Hasher, VctrlError};

fn hash() -> Result<(), VctrlError> {
    let hasher = Sha512Hasher;
    let digest = hasher.hash(b"konten".as_ref())?;
    assert_eq!(digest.as_bytes().len(), 64);
    Ok(())
}
```

5.3 Builder

Lima builder tersedia. Tabel 8 mencantumkan tanda tangan metode.

Table 8. Metode builder pada `libvctrl_core::object`.

Builder	Metode
`BlobBuilder`	`new(), with_data(Vec<u8>), build() -> Result<Blob, _>`
`TreeEntryBuilder`	`new(String, EntryKind, Hash), build() -> Result<TreeEntry, _>`
`TreeBuilder`	`new(), entry(TreeEntry), add_entry(String, EntryKind, Hash), build()`
`CommitBuilder`	`new(), tree(Hash), parent(Hash), author(UserID), committer(UserID), message(impl Into<String>), meta(CommitMeta), build()`
`TagBuilder`	`new(), name(impl Into<String>), target(Hash), tagger(UserID), message(impl Into<String>), meta(CommitMeta), build()`

Builder memvalidasi field wajib pada `build()`. Jika field wajib absen, `build()` mengembalikan `VctrlError::Other`. Contoh ditunjukkan pada blok kode berikut.

```
use libvctrl_core::object::{CommitBuilder, TreeBuilder, TreeEntryBuilder};
use libvctrl_handler::{EntryKind, Hash, UserID, VctrlError};

fn build() -> Result<(), VctrlError> {
    let hash = Hash::from_bytes(&[1u8; 64])?;
    let entry = TreeEntryBuilder::new(
        "a.txt".to_string(), EntryKind::Blob, hash,
    ).build()?;
    let tree = TreeBuilder::new().entry(entry).build()?;
    let user = UserID::new(
        "Alice".to_string(), "alice@example.com".to_string(),
    )?;
    let commit = CommitBuilder::new()
        .tree(hash)
        .author(user.clone())
        .committer(user)
        .message("pesan")
        .build()?;
    assert_eq!(commit.message(), "pesan");
    let _ = tree;
    Ok(())
}
```

5.4 In-Memory Store

`MemoryStore` mengimplementasikan `ObjectStore` dengan `HashMap<Hash, Vec<u8>>`. `MemoryRefStore` mengimplementasikan `RefStore` dengan `HashMap<String, Hash>`. Keduanya cocok untuk pengujian dan prototipe. Contoh ditunjukkan pada blok kode berikut.

```
use std::io::Read;
use libvctrl_core::store::{MemoryRefStore, MemoryStore};
use libvctrl_handler::{Hash, ObjectStore, RefStore, VctrlError};

fn store() -> Result<(), VctrlError> {
    let mut objects = MemoryStore::new();
    let mut refs = MemoryRefStore::new();
    let hash = Hash::from_bytes(&[2u8; 64])?;
    objects.put(&hash, b"data")?;
    refs.set_ref("refs/heads/main", &hash)?;
    let mut buf = Vec::new();
    objects.get(&hash)?.read_to_end(&mut buf)?;
    assert_eq!(buf, b"data");
    Ok(())
}
```

Gambar 5 menunjukkan siklus encode, hash, simpan, dan bangun ulang.

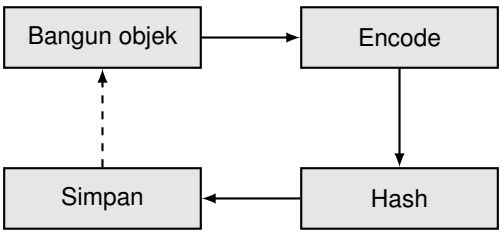


Figure 5. Siklus encode, hash, simpan, dan bangun ulang objek.

6 Kriptografi: libvctrl_sha512

6.1 Primitif dan Feature Flags

`libvctrl_sha512` menyediakan SHA-512, HMAC-SHA512, dan HKDF-SHA512. Modul `sha384` tersedia jika fitur `sha384` aktif (default). Fitur `opt_size` mengubah strategi inlining round function untuk mengurangi ukuran binary. Crate ini `no_std` ketika dibangun di luar target test. Satu-satunya dependensi eksternal adalah `zeroize` dengan `default-features = false`.

Tabel 9 mencantumkan tanda tangan metode utama.

Table 9. Tanda tangan API kriptografi.

Tipe	Metode	Keluaran
`Hash`	`new(), update(impl AsRef<[u8]>), finalize(), hash(impl AsRef<[u8]>)`	`[u8; 64]`
`Hash`	`verify(self, &[u8; 64]) -> bool`	`bool`
`HMAC`	`mac(T, U) -> [u8; 64], new(impl AsRef<[u8]>)`	`[u8; 64]`
`HMAC`	`finalize_verify(self, &[u8; 64]) -> bool`	`bool`
`HKDF`	`extract(salt, ikm) -> [u8; 64]`	`[u8; 64]`
`HKDF`	`expand(&mut [u8], prk, info)`	`()`
`sha384::Hash`	`hash, update, finalize`	`[u8; 48]`

6.2 Contoh HMAC dan HKDF

Blok kode berikut menunjukkan autentikasi pesan dan derivasi kunci.

```
use libvctrl_sha512::{HKDF, HMAC};

fn kripto() {
    let tag = HMAC::mac(b"pesan", b"kunci");
    assert!(HMAC::verify(b"pesan", b"kunci", &tag));

    let prk = HKDF::extract(b"salt", b"input");
    let mut out = [0u8; 32];
    HKDF::expand(&mut out, prk, b"konteks");
    assert_eq!(out.len(), 32);
}
```

7 Fasad dan Plumbing

7.1 Re-export Fasad

`libvctrl` mengekspor ulang `handler`, `reference`, dan `crypto` sebagai namespace. Namespace `crypto` memisahkan `libvctrl_sha512::Hash` (hasher) dari `libvctrl_handler::Hash` (content address). Pemakai yang mengimpor keduanya harus memakai alias.

7.2 Repository

`Repository` menyediakan API tingkat tinggi untuk commit, snapshot, baca objek, dan kelola cabang. Tabel 10 mencantumkan tanda tangan metode.

Table 10. Tanda tangan metode `Repository`.

Metode	Tanda tangan
`new`	`new(Box<dyn ObjectStore>, Box<dyn RefStore>) -> Self`
`commit`	`commit(&mut self, &str, impl IntoIterator<Item=FileToCommit>, impl Into<String>, UserID) -> Result<Hash, _>`
`snapshot`	`snapshot(&mut self, impl IntoIterator<Item=FileToCommit>) -> Result<Hash, _>`
`read_blob`	`read_blob(&self, &Hash) -> Result<Blob, _>`
`read_tree`	`read_tree(&self, &Hash) -> Result<Tree, _>`
`read_commit`	`read_commit(&self, &Hash) -> Result<Commit, _>`
`head`	`head(&self, &str) -> Result<Hash, _>`
`branches`	`branches(&self) -> Result<Vec<String>, _>`
`log`	`log(&self, &str) -> Result<Vec<(Hash, Commit)>, _>`
`file_at`	`file_at(&self, &Hash, &str) -> Result<Option<Vec<u8», _>`
`create_branch`	`create_branch(&mut self, &str, &Hash) -> Result<(), _>`
`delete_branch`	`delete_branch(&mut self, &str) -> Result<(), _>`

`FileToCommit` menyediakan tiga konstruktor: `blob(path, data)`, `executable(path, data)`, dan `symlink(path, target)`. Tipe ini memuat `path: String`, `kind: EntryKind`, dan `data: Vec<u8>`.

7.3 cat_file dan cat_file_batch

`libvctrl_plumbing` menyediakan dua fungsi tingkat perintah. Tabel 11 mencantumkan mode dan keluaran.

Table 11. Mode `CatFileMode` dan keluarannya.

Mode	Keluaran
`Exists`	Tidak ada output, sukses jika objek ada
`ObjectType`	Nama tipe diikuti newline
`ObjectSize`	Ukuran payload dalam byte
`PrettyPrint`	Representasi teks objek
`Raw(ObjectType)`	Byte payload mentah, gagal jika tipe tidak cocok

`BatchOptions` memuat field `format: Option<String>`, `nul_terminated: bool`, `follow_symlinks: bool`, `buffer: bool`, dan `print_contents: bool`. Format string mendukung placeholder `%(objectname)`, `%(objecttype)`, dan `%(objectsize)`.

Contoh pemakaian ditunjukkan pada blok kode berikut.

```
use libvctrl_plumbing::cat_file::{cat_file, CatFileMode};
use libvctrl::{BinaryDecoder, Hash, MemoryStore, VctrlError};

fn cat(store: &MemoryStore, hash: &Hash)
    -> Result<(), VctrlError>
{
    let decoder = BinaryDecoder;
    let mut out = Vec::new();
```

```
    cat_file(
        store, &decoder, &hash.to_string(),
        CatFileMode::PrettyPrint, &mut out,
    )?;
    assert!(!out.is_empty());
    Ok(())
}
```

Keluaran `PrettyPrint` untuk objek blob mengikuti format berikut.

```
halo
```

8 Template Implementasi Backend Kustom

8.1 Kontrak yang Harus Dipenuhi

Backend kustom yang ingin dipakai bersama `Repository` harus mengimplementasikan `ObjectStore` dan `RefStore`. Keduanya memiliki bound `Send + Sync`. Backend boleh mengimplementasikan trait lain seperti `Index` atau `RevWalk` jika diperlukan. Tabel 12 mencantumkan checklist implementasi.

Table 12. Checklist implementasi backend kustom.

Item	Kewajiban
`put`	Menyimpan byte mentah dengan kunci `Hash`, idempoten
`get`	Mengembalikan `Box<dyn Read + Send + '_>` yang membaca byte
`delete`	Menghapus entri jika ada, idempoten
`exists`	Mengembalikan `false` tanpa error jika tidak ada
`set_ref`	Memvalidasi nama via `validate_ref_name`
`get_ref`	Mengembalikan `RefNotFound` jika nama tidak ada
`delete_ref`	Menghapus ref jika ada, idempoten
`list_refs`	Mengembalikan vektor nama yang terurut
Bound	`impl ObjectStore for MyStore` dengan `MyStore: Send + Sync`

8.2 Template Implementasi

Template berikut menunjukkan pola implementasi dengan `Mutex` internal. Pola ini dipakai `MonumentumStore`.

```
use std::io::{Cursor, Read};
use std::sync::Mutex;
use libvctrl_handler::{
    Hash, ObjectStore, RefStore, VctrlError, validate_ref_name,
};

pub struct MyStore {
    objects: Mutex<std::collections::HashMap<Hash, Vec<u8>>>,
```

```
    refs: Mutex<std::collections::HashMap<String, Hash>>,
}

impl ObjectStore for MyStore {
    fn put(&mut self, hash: &Hash, data: &[u8])
        -> Result<(), VctrlError>
    {
        self.objects.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .insert(*hash, data.to_vec());
        Ok(())
    }

    fn get(&self, hash: &Hash)
        -> Result<Box<dyn Read + Send + '_>, VctrlError>
    {
        let guard = self.objects.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?;
        let bytes = guard.get(hash)
            .ok_or(VctrlError::ObjectNotFound(*hash))?
            .clone();
        Ok(Box::new(Cursor::new(bytes)))
    }

    fn delete(&mut self, hash: &Hash) -> Result<(), VctrlError> {
        self.objects.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .remove(hash);
        Ok(())
    }

    fn exists(&self, hash: &Hash) -> Result<bool, VctrlError> {
        Ok(self.objects.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .contains_key(hash))
    }
}

impl RefStore for MyStore {
    fn set_ref(&mut self, name: &str, hash: &Hash)
        -> Result<(), VctrlError>
    {
        validate_ref_name(name)?;
        self.refs.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .insert(name.to_string(), *hash);
        Ok(())
    }

    fn get_ref(&self, name: &str) -> Result<Hash, VctrlError> {
        self.refs.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .get(name).copied()
            .ok_or_else(|| VctrlError::RefNotFound(name.into()))
    }
```

```
    fn delete_ref(&mut self, name: &str)
        -> Result<(), VctrlError>
    {
        self.refs.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .remove(name);
        Ok(())
    }

    fn list_refs(&self) -> Result<Vec<String>, VctrlError> {
        let mut names: Vec<String> = self.refs.lock()
            .map_err(|_| VctrlError::Other("lock".into()))?
            .keys().cloned().collect();
        names.sort();
        Ok(names)
    }
}
```

Gambar 6 menunjukkan posisi backend sebagai implementasi trait.

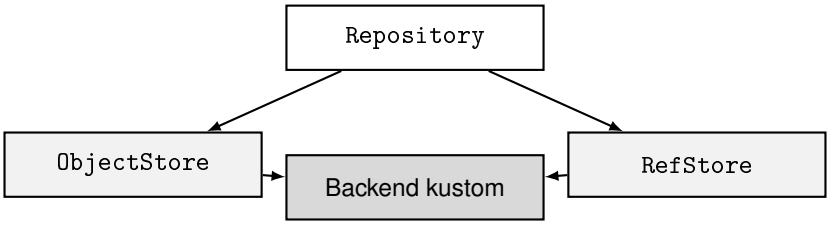


Figure 6. Integrasi backend kustom melalui trait penyimpanan.

9 Studi Kasus: Commit End-to-End

9.1 Langkah 1: Siapkan Store

Gunakan `MemoryStore` dan `MemoryRefStore` sebagai backend minimal.

```
use libvctrl::{MemoryRefStore, MemoryStore, Repository};

let mut repo = Repository::new(
    Box::new(MemoryStore::new()),
    Box::new(MemoryRefStore::new()),
);
```

9.2 Langkah 2: Buat UserID

Buat identitas penulis. Email wajib memuat `@`.

```
use libvctrl::UserID;
```

```
let user = UserID::new(
    "Alice".to_string(), "alice@example.com".to_string(),
)?;
```

9.3 Langkah 3: Siapkan Berkas

Bungkus isi berkas menjadi `FileToCommit`.

```
use libvctrl::FileToCommit;

let files = vec![
    FileToCommit::blob("catatan.md", b"halo".to_vec()),
];
```

9.4 Langkah 4: Commit

Panggil `commit` pada cabang utama.

```
let commit = repo.commit(
    "refs/heads/main", files, "commit pertama", user,
)?;
```

9.5 Langkah 5: Baca HEAD

Periksa hash ujung cabang.

```
let head = repo.head("refs/heads/main")?;
assert_eq!(head, commit);
```

9.6 Langkah 6: Baca Berkas pada Commit

Baca isi berkas pada commit tertentu.

```
let isi = repo.file_at(&commit, "catatan.md")?;
assert!(isi.is_some());
```

9.7 Langkah 7: Lihat Log

Telusuri riwayat commit.

```
let log = repo.log("refs/heads/main")?;
assert_eq!(log.len(), 1);
```

9.8 Langkah 8: Buat Cabang

Buat cabang baru dari commit yang sudah ada.

```
repo.create_branch("refs/heads/dev", &commit)?;
assert_eq!(repo.head("refs/heads/dev")?, commit);
```

9.9 Langkah 9: Snapshot Tanpa Commit

Buat tree tanpa membuat commit, berguna untuk preview.

```
let tree_hash = repo.snapshot(vec![
    FileToCommit::blob("catatan.md", b"halo".to_vec()),
])?;
assert_eq!(tree_hash.as_bytes().len(), 64);
```

9.10 Langkah 10: Commit Kedua

Commit kedua memakai HEAD cabang sebagai parent otomatis.

```
let commit2 = repo.commit(
    "refs/heads/main",
    vec![FileToCommit::blob("catatan.md", b"halo 2".to_vec())],
    "commit kedua",
    UserID::new(
        "Alice".to_string(), "alice@example.com".to_string(),
    )?,
)?;
let log = repo.log("refs/heads/main")?;
assert_eq!(log.len(), 2);
assert_eq!(log.first().map(|(h, _)| *h), Some(commit2));
```

Gambar 7 menunjukkan lima langkah utama di dalam `Repository::commit`.

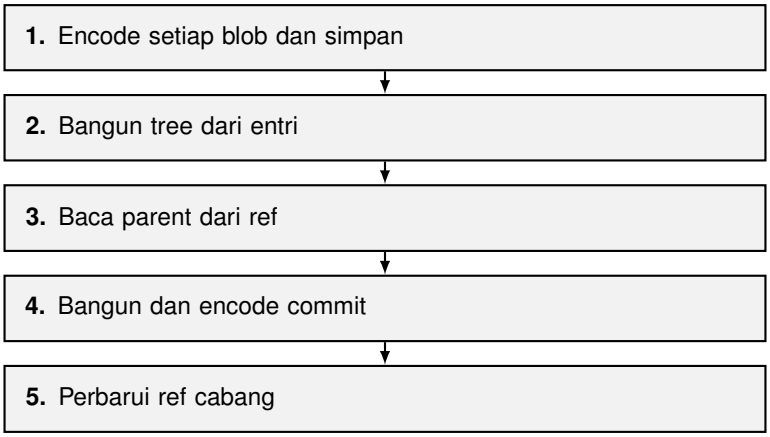


Figure 7. Lima langkah utama `Repository::commit`.

10 Kesalahan Umum dan Cara Menghindarinya

Tabel 13 merangkum pemicu error dan perbaikannya.

Table 13. Pemicu error umum dan perbaikannya.

Pemicu	Varian	Perbaikan
Hash kurang dari 64 byte	`InvalidHashLength`	Pakai array `[u8; 64]`
Nama entri memuat `/`	`InvalidName`	Pecah jadi tree bertingkat
Ref diawali `.` atau `/`	`InvalidName`	Pakai `refs/heads/...`
Parent duplikat	`DuplicateParent`	Deduplikasi sebelum `Commit::new`
Pesan melebihi 1 MiB	`ExceededMaxSize`	Perpendek pesan
Offset zona waktu > 1440	`InvalidTimezoneOffset`	Pakai menit, rentang ± 1440
Email tanpa `@`	`InvalidEmail`	Validasi sebelum `UserID::new`
`get` sebelum `put`	`ObjectNotFound`	Simpan objek terlebih dahulu
Version byte bukan 3	`CorruptedData`	Re-encode dengan encoder versi 3
Trailing byte setelah payload	`CorruptedData`	Pastikan panjang prefix akurat
Kind byte > 4	`CorruptedData`	Pakai `EntryKind` yang valid
Tagger presence byte bukan $0/1$	`CorruptedData`	Normalisasi ke `0` atau `1`

Kesalahan paling sering pada integrasi baru adalah panjang hash yang salah dan ref name yang tidak lolos `validate_ref_name`. Keduanya bisa dicegah dengan memanggil fungsi validasi sebelum konstruksi dan sebelum `set_ref`.

11 Lembar Rujukan Cepat

Tabel 14 menjadi lembar rujukan cepat untuk operasi yang paling sering dibutuhkan pengembang.

Table 14. Lembar rujukan cepat API libvctrl.

Kebutuhan	API atau tipe
Hash dari byte	`Hash::from_bytes(&[u8])`
Hash ke byte	`hash.as_bytes()`
Blob dari `Vec`	`Blob::new(Vec<u8>)`
Entri tree	`TreeEntry::new(String, EntryKind, Hash)`
Tree dari entri	`Tree::new(Vec<TreeEntry>)`
Commit tanpa meta	`Commit::new(tree, parents, a, c, msg)`
Commit dengan meta	`Commit::with_meta(..., meta)`
Tag anotasi	`Tag::new(name, target, tagger, msg)`
User valid	`UserID::new(name, email)`
Encode blob	`BinaryEncoder.encode_blob(&b, &mut w)`
Decode blob	`BinaryDecoder.decode_blob(r)`
Hash stream	`Sha512Hasher.hash(r)`
Simpan objek	`ObjectStore::put(&mut s, &h, &d)`
Ambil objek	`ObjectStore::get(&s, &h)`
Simpan ref	`RefStore::set_ref(&mut s, name, &h)`
Ambil ref	`RefStore::get_ref(&s, name)`
Commit berkas	`Repository::commit(...)`
Baca berkas	`Repository::file_at(&h, path)`
Log cabang	`Repository::log(name)`
Tampilkan objek	`cat_file(...)`
Mode batch	`cat_file_batch(...)`
Backend Monumentum	`MonumentumStore::new(engine)`

12 Kesimpulan

Dokumen ini menyajikan tiga perspektif API libvctrl: kontrak sebagai batas, format biner sebagai spesifikasi, dan backend sebagai titik ekstensi. Tiga prinsip penutup:

1. Kontrak adalah batas yang tidak dapat dilanggar. `libvctrl_handler` menetapkan tipe, trait, konstanta, dan kondisi error yang harus dipatuhi oleh setiap implementasi. Backend kustom yang melanggar kontrak akan gagal pada `Repository` atau pada decoder, bukan pada pemanggil.
2. Format adalah spesifikasi, bukan detail internal. Layout byte-level pada Bagian 4 adalah kontrak publik. Perubahan layout memerlukan versi baru pada version byte dan decoder yang kompatibel.
3. Ekstensi adalah titik masuk yang terdefinisi. Backend kustom mengimplementasikan `ObjectStore` dan `RefStore`, bukan memodifikasi `libvctrl_core`. Ekstensi tidak boleh mengubah tipe kontrak atau menambah varian error di luar `VctrlError`.

Pengembang yang memahami ketiga prinsip ini dapat memakai libvctrl sebagai SDK lengkap, sebagai kumpulan crate terpisah, atau sebagai fondasi untuk backend kustom tanpa risiko melanggar invariant yang dijamin oleh lapisan kontrak.

Referensi

References

[1] libvctrl Contributors. (2026). *libvctrl API Documentation*. docs.rs. https://docs.rs/libvctrl

[2] libvctrl Contributors. (2026). *libvctrl_handler: Contracts for Version Control*. docs.rs. https://docs.rs/libvctrl_handler

[3] libvctrl Contributors. (2026). *libvctrl_core: Reference Implementations*. docs.rs. https://docs.rs/libvctrl_core

[4] libvctrl Contributors. (2026). *libvctrl_sha512: SHA-512, HMAC, HKDF*. docs.rs. https://docs.rs/libvctrl_sha512

[5] libvctrl Contributors. (2026). *libvctrl_plumbing: Command-Level Operations*. docs.rs. https://docs.rs/libvctrl_plumbing

[6] Rust Team. (2026). *The Rust Programming Language*. https://doc.rust-lang.org/book/

[7] Rust Team. (2026). *Rust API Guidelines*. https://rust-lang.github.io/api-guidelines/

[8] Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress. https://git-scm.com/book/

[9] National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)*. FIPS PUB 180-4. https://doi.org/10.6028/NIST.FIPS.180-4

[10] Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. IETF. https://www.rfc-editor.org/rfc/rfc2104

[11] Krawczyk, H., & Eronen, P. (2010). *HKDF: HMAC-based Extract-and-Expand Key Derivation Function*. RFC 5869. IETF. https://www.rfc-editor.org/rfc/rfc5869