The Workspace Abstraction: Deep Dive

> *Last updated: 2026-02-27. Based on full codebase audit of all workspace-related source files.*

Table of Contents

---

1. What the Abstraction Provides

The Workspace abstraction is the **organizing principle** of fuel-code. Every session, event, git activity row, and device link groups under a workspace. A workspace corresponds to one git repository — identified not by a local filesystem path (which varies across machines) but by a **canonical ID** derived deterministically from the git remote URL.

This means the same repo checked out at `/Users/john/project` on a MacBook and `/home/deploy/project` on a remote server resolves to the **same workspace** — their sessions, commits, and activity merge into a single view.

Core value proposition: Answer "what work happened in this repo?" regardless of which machine, which checkout path, or which URL format was used to clone it.

The workspace is **one of the original five CORE.md abstractions** (Workspace, Session, Event, Device, Blueprint). It's described in the abstractions audit as "the most disciplined abstraction" — with type definitions, canonical ID derivation, resolver/upsert, junction table, routes, and CLI all present and aligned.

---

2. The Dual Identity System

Every workspace has **two identifiers**:

Identifier	Format	Example	Purpose
`canonical_id`	Normalized string	`"github.com/user/repo"`	Stable, human-meaningful, UNIQUE. Used for deduplication.
`id`	ULID (26-char)	`"01HXYZ..."`	Internal primary key. Used as FK in all downstream tables.

Why two? Events arrive from the CLI with workspace IDs as **canonical strings** (e.g., `"github.com/user/repo"`). But Postgres foreign keys should be opaque, stable identifiers — not human-meaningful strings that might need format changes. The resolver translates canonical → ULID at event ingestion time. All downstream references (sessions, events, git_activity) use the ULID.

The canonical_id has a UNIQUE constraint, so two events from different URL formats of the same repo (SSH vs HTTPS) resolve to the same workspace row.

---

3. Canonical ID Derivation

Defined in `packages/shared/src/canonical.ts` — three pure functions, no side effects, no I/O.

3.1 `normalizeGitRemote(remoteUrl: string): string | null`

Converts any git URL format to `host/path`:

Input Format	Example Input	Output
SSH (SCP-style)	`git@github.com:user/repo.git`	`github.com/user/repo`
HTTPS	`https://github.com/user/repo.git`	`github.com/user/repo`
SSH protocol	`ssh://git@github.com/user/repo`	`github.com/user/repo`
Git protocol	`git://github.com/repo.git`	`github.com/repo`
With port	`ssh://git@host:2222/repo`	`host/repo`
Empty/whitespace	"" or " "	`null`

Normalization rules (applied in order):

1. **Detect SCP-style**: Match `[user@]host:path` where `:` is NOT followed by `//`
2. **Strip protocol**: Remove `https://`, `http://`, `ssh://`, `git://`
3. **Strip auth**: Remove everything before `@`
4. **Lowercase host only**: Host is lowercased, path case preserved
5. **Strip port**: Remove `:digits` from host
6. **Strip `.git` suffix**: Remove from path
7. **Strip trailing slashes**: Clean path

Key design decision: Host is lowercased but path is NOT. This means `github.com/User/Repo` and `github.com/user/repo` are **different canonical IDs**. GitHub treats paths case-insensitively but GitLab/Gitea don't, so the conservative choice is to preserve case.

3.2 `deriveWorkspaceCanonicalId(remoteUrl, firstCommitHash): string`

Priority-based derivation:

```
1. Remote URL available?  → normalizeGitRemote(url)
2. No remote, has commits? → "local:<sha256(firstCommitHash)>"
3. Neither?               → "_unassociated"
```

The `local:` prefix uses SHA-256 of the first commit hash (not the hash itself), making it deterministic per repo history. Two clones of the same local repo with the same initial commit resolve to the same workspace.

3.3 `deriveDisplayName(canonicalId: string): string`

Human-readable name extraction:

Canonical ID	Display Name
`"_unassociated"`	`"_unassociated"`
`"local:abc123def..."`	`"local-abc12345"` (first 8 hex chars)
`"github.com/user/repo"`	`"repo"` (last path segment)
`"gitlab.com/org/group/project"`	`"project"`

Collision risk: Two repos named `repo` under different orgs (e.g., `github.com/alice/repo` and `github.com/bob/repo`) get the same `display_name: "repo"` . The API handles this via the ambiguity check (see Section 8).

---

4. The Database Schema

4.1 `workspaces` Table

`packages/server/src/db/migrations/001_initial.sql` :

```sql
CREATE TABLE IF NOT EXISTS workspaces (
    id              TEXT PRIMARY KEY,         -- ULID (26-char Crockford Base32)
    canonical_id    TEXT NOT NULL UNIQUE,     -- Normalized remote URL or sentinel
    display_name    TEXT NOT NULL,            -- Human-readable repo name
    default_branch  TEXT,                     -- Branch captured on first sight
    metadata        JSONB NOT NULL DEFAULT '{}', -- all_remotes, etc.
    first_seen_at   TIMESTAMPTZ NOT NULL DEFAULT now(),
    updated_at      TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Column details:

Column	Type	Nullable	Purpose
`id`	TEXT PK	NO	ULID, generated once on first event for this repo
`canonical_id`	TEXT UNIQUE	NO	Deterministic identity. UNIQUE constraint prevents duplicates.
`display_name`	TEXT	NO	Last path segment of canonical_id. Non-unique.
`default_branch`	TEXT	YES	Git branch from the first session.start event. Never overwritten.
`metadata`	JSONB	NO	Contains `all_remotes` array if multiple remotes detected.
`first_seen_at`	TIMESTAMPTZ	NO	Immutable. When this workspace was first created.
`updated_at`	TIMESTAMPTZ	NO	Touched on every event from this workspace.

No explicit indexes on `workspaces` — the table is expected to be small (one row per git repo). The UNIQUE constraint on `canonical_id` creates an implicit unique index.

4.2 TypeScript Type

`packages/shared/src/types/workspace.ts` :

```typescript
export const UNASSOCIATED_WORKSPACE = "_unassociated" as const;

export interface Workspace {
  id: string;                          // ULID primary key
  canonical_id: string;                // Normalized identifier
  display_name: string;                // Human-readable name
  default_branch: string | null;       // Default branch (nullable)
  metadata: Record<string, unknown>;   // Arbitrary metadata
  first_seen_at: string;               // ISO-8601 timestamp
  updated_at: string;                  // ISO-8601 timestamp
}
```

Type/DB alignment: The Workspace type accurately matches the database schema. All 7 columns are present with correct types. This is the **only abstraction** in fuel-code where the TypeScript type and database schema are fully aligned.

4.3 Foreign Key References

Four tables reference `workspaces.id` :

Table	FK Column	Relationship
`workspace_devices`	`workspace_id` (composite PK)	Many-to-many with devices
`sessions`	`workspace_id NOT NULL`	Every session belongs to one workspace
`events`	`workspace_id NOT NULL`	Every event scoped to one workspace
`git_activity`	`workspace_id NOT NULL`	Every git operation tied to one workspace

---

5. The Workspace-Device Junction

`packages/server/src/db/migrations/001_initial.sql` + `004_add_git_hooks_prompt_columns.sql` :

```sql
CREATE TABLE IF NOT EXISTS workspace_devices (
    workspace_id              TEXT NOT NULL REFERENCES workspaces(id),
    device_id                 TEXT NOT NULL REFERENCES devices(id),
    local_path                TEXT NOT NULL,
    hooks_installed           BOOLEAN NOT NULL DEFAULT false,
    git_hooks_installed       BOOLEAN NOT NULL DEFAULT false,
    pending_git_hooks_prompt  BOOLEAN NOT NULL DEFAULT false,
    git_hooks_prompted        BOOLEAN NOT NULL DEFAULT false,
    last_active_at            TIMESTAMPTZ NOT NULL DEFAULT now(),
    PRIMARY KEY (workspace_id, device_id)
);
```

This junction table tracks **where each workspace is checked out on each device**:

Column	Purpose
`local_path`	Filesystem path (e.g., `/Users/john/Desktop/fuel-code`). Updated on every event — handles repo moves.
`hooks_installed`	Whether CC session hooks are active for this workspace-device pair.
`git_hooks_installed`	Whether git hooks are active (via `core.hooksPath`). Separate from CC hooks.
`pending_git_hooks_prompt`	Flag: should next interactive session prompt user to install git hooks?
`git_hooks_prompted`	Once-only flag: has the user been prompted? Prevents repeated nagging.
`last_active_at`	Timestamp of last event from this workspace on this device.

Design note: `hooks_installed` (CC hooks) and `git_hooks_installed` (git hooks) are tracked separately because they're installed by different commands and can exist independently. The `hooks_installed` column is a **ghost column** — it exists in the schema but is never read or written by any code. Only `git_hooks_installed` is functional.

5.1 Link Management

`packages/core/src/workspace-device-link.ts` :

```
export async function ensureWorkspaceDeviceLink(
  sql: Sql,
  workspaceId: string,
  deviceId: string,
  localPath: string,
): Promise<void> {
  await sql`
    INSERT INTO workspace_devices (workspace_id, device_id, local_path)
    VALUES (${workspaceId}, ${deviceId}, ${localPath})
    ON CONFLICT (workspace_id, device_id) DO UPDATE SET
      last_active_at = now(),
      local_path = EXCLUDED.local_path
  `;
}
```

Called by the event processor on **every event** after workspace and device are resolved. The upsert:

- Creates the link if it doesn't exist
- Updates `last_active_at` if it does
- Updates `local_path` in case the checkout moved

5.2 Hook Prompt Flow

The git hooks prompt system uses the workspace-device junction to implement a one-time prompt per workspace:

```
1. Session starts in workspace W on device D
2. Handler checks workspace_devices: is git_hooks_installed? is git_hooks_prompted?
3. If neither: sets pending_git_hooks_prompt = true
4. CLI pre-action hook hits GET /api/prompts/pending
5. If pending: shows prompt to user
6. User accepts/declines: POST /api/prompts/dismiss
7. Sets git_hooks_prompted = true (never asks again)
```

---

6. The Resolution Pipeline

Every event that enters fuel-code goes through workspace resolution in the event processor (`packages/core/src/event-processor.ts`):

```
Event arrives
  event.workspace_id = "github.com/user/repo"  (canonical string, NOT a ULID)
      │
      ▼
resolveOrCreateWorkspace(sql, "github.com/user/repo", hints?)
      │
      ├─ New workspace? INSERT with fresh ULID, RETURNING id
      │
      └─ Existing? ON CONFLICT (canonical_id) DO UPDATE SET updated_at = now(),
RETURNING id
      │
      ▼
Returns ULID: "01HXYZ..."
      │
      ▼
ensureWorkspaceDeviceLink(sql, "01HXYZ...", deviceId, localPath)
      │
      ▼
INSERT event row with workspace_id = "01HXYZ..."  (ULID, not canonical)
```

6.1 `resolveOrCreateWorkspace(sql, canonicalId, hints?)`

`packages/core/src/workspace-resolver.ts`:

```
export async function resolveOrCreateWorkspace(
  sql: Sql,
  canonicalId: string,
  hints?: { default_branch?: string; all_remotes?: string[] },
): Promise<string> {
  const effectiveId = canonicalId.trim() || "_unassociated";
  const displayName = deriveDisplayName(effectiveId);
  const id = generateId();  // Fresh ULID — only used if INSERT succeeds

  const metadata: Record<string, unknown> = {};
  if (hints?.all_remotes?.length) {
    metadata.all_remotes = hints.all_remotes;
  }

  const [row] = await sql`
    INSERT INTO workspaces (id, canonical_id, display_name, default_branch,
metadata)
    VALUES (${id}, ${effectiveId}, ${displayName}, ${hints?.default_branch ?? null},
${JSON.stringify(metadata)})
    ON CONFLICT (canonical_id) DO UPDATE SET updated_at = now()
    RETURNING id
  `;

  return row.id;
}
```

Key invariants:

- **Idempotent**: Multiple concurrent calls with the same canonical_id are safe — only one row is created
- **First-seen preservation**: `default_branch` and `metadata` are set on INSERT, never overwritten on conflict
- **Empty normalization**: Empty or whitespace-only canonical IDs normalize to `"_unassociated"`
- **ULID generation**: A fresh ULID is generated every call but discarded if the workspace already exists

6.2 Hint Extraction

The event processor extracts workspace hints from specific event types:

```
function extractHints(event: Event): { default_branch?: string } | undefined {
  if (event.type === "session.start") {
    const branch = event.data.git_branch;
    if (typeof branch === "string" && branch.length > 0) {
      return { default_branch: branch };
    }
  }
  return undefined;
}
```

Only `session.start` events carry the `git_branch` hint. The first session in a workspace captures the default branch; all subsequent sessions' branch hints are ignored (first-seen wins via the ON CONFLICT clause).

---

7. The Bash/TS Normalization Parity Contract

Workspace canonical IDs are derived in **two places**:

Implementation	Language	Location	Called By
`resolve-workspace.sh`	Bash	`packages/hooks/git/`	Git hooks (post-commit, pre-push, etc.)

`resolveWorkspace()`	TypeScript	`packages/hooks/claude/_helpers/resolve-workspace.ts`	CC hooks (SessionStart, SessionEnd)
`normalizeGitRemote()`	TypeScript	`packages/shared/src/canonical.ts`	Everything else (backend, CLI, core)

Critical contract: The bash and TS implementations **must produce identical canonical IDs** for the same repository. If they diverge, git activity from hooks and sessions from CC hooks would resolve to different workspaces.

7.1 Bash Implementation

```
# resolve-workspace.sh

# Remote priority: origin first, then first alphabetically
REMOTE_URL=$(git remote get-url origin 2>/dev/null)
if [ -z "$REMOTE_URL" ]; then
  FIRST_REMOTE=$(git remote 2>/dev/null | sort | head -1)
  REMOTE_URL=$(git remote get-url "$FIRST_REMOTE" 2>/dev/null)
fi

# No remote → local:<sha256>
if [ -z "$REMOTE_URL" ]; then
  FIRST_COMMIT=$(git rev-list --max-parents=0 HEAD 2>/dev/null | head -1)
  echo "local:$(echo -n "$FIRST_COMMIT" | shasum -a 256 | cut -d' ' -f1)"
  exit 0
fi

# SSH normalization: git@host:path → host/path
if [[ "$REMOTE_URL" =~ ^[a-zA-Z0-9._-]+@([^:]+):(.+)$ ]]; then
  HOST=$(echo "${BASH_REMATCH[1]}" | tr '[:upper:]' '[:lower:]')
  PATH_PART="${BASH_REMATCH[2]%.git}"
  echo "${HOST}/${PATH_PART}"
fi

# HTTPS normalization: https://host/path → host/path
if [[ "$REMOTE_URL" =~ ^https?://([^/]+)/(.+)$ ]]; then
  HOST=$(echo "${BASH_REMATCH[1]}" | tr '[:upper:]' '[:lower:]')
  PATH_PART="${BASH_REMATCH[2]%.git}"
  PATH_PART="${PATH_PART%/}"
  echo "${HOST}/${PATH_PART}"
fi
```

7.2 TS Implementation (Hook Helper)

```
// resolve-workspace.ts
export async function resolveWorkspace(cwd: string): Promise<WorkspaceInfo> {
  // 1. Check git repo
  // 2. Get current branch
  // 3. Prefer origin, fall back to first alphabetically
  // 4. If no remote, try first commit hash
  // 5. Derive canonical ID via deriveWorkspaceCanonicalId()
  // Never throws — returns "_unassociated" on any failure
}
```

The TS hook helper delegates to `deriveWorkspaceCanonicalId()` → `normalizeGitRemote()` which handles the same URL formats as the bash script.

7.3 Parity Gaps

Aspect	Bash	TypeScript	Risk
SSH regex	`[a-zA-Z0-9._-]+@`	`(?:[^@]+@)?([^:/]+):(?!//)`	TS is slightly broader (handles special chars in username)
Port stripping	Not handled	`host.replace(/:\d+$/, "")`	SSH with port: bash produces `host:2222/path`, TS produces `host/path`
Unknown formats	Echo as-is	Return `null` → `"_unassociated"`	Divergent for edge-case URLs
`shasum` vs `createHash`	`shasum -a 256`	`crypto.createHash("sha256")`	Identical output for same input

The port stripping gap is the most likely to cause real divergence. An SSH URL like `ssh://git@github.com:443/user/repo` would normalize differently in bash vs TS.

---

8. Query Surfaces (API + CLI)

8.1 API Endpoints

`GET /api/workspaces` — List with keyset pagination

`packages/server/src/routes/workspaces.ts`:

Query params:

- `limit` (1-250, default 50)
- `cursor` (opaque base64 token)

Response:

```
{
  "workspaces": [
    {
      "id": "01HXYZ...",
      "canonical_id": "github.com/user/repo",
      "display_name": "repo",
      "default_branch": "main",
      "metadata": { "all_remotes": ["origin"] },
      "first_seen_at": "2026-02-01T00:00:00Z",
      "updated_at": "2026-02-27T12:00:00Z",
      "session_count": 42,
      "active_session_count": 1,
      "last_session_at": "2026-02-27T12:00:00Z",
      "device_count": 2,
      "total_cost_usd": 3.14,
      "total_duration_ms": 360000,
      "total_tokens_in": 500000,
      "total_tokens_out": 125000
    }
  ],
  "next_cursor": "eyJ1Ijo...",
  "has_more": true
}
```

The aggregates come from a CTE that LEFT JOINs sessions:

```
WITH workspace_agg AS (
  SELECT w.*,
    COUNT(s.id) AS session_count,
    COUNT(CASE WHEN s.lifecycle IN ('detected', 'capturing') THEN 1 END) AS
active_session_count,
    MAX(s.started_at) AS last_session_at,
    COUNT(DISTINCT s.device_id) AS device_count,
    COALESCE(SUM(s.cost_estimate_usd), 0) AS total_cost_usd,
    COALESCE(SUM(s.duration_ms), 0) AS total_duration_ms,
    COALESCE(SUM(s.tokens_in), 0) AS total_tokens_in,
    COALESCE(SUM(s.tokens_out), 0) AS total_tokens_out
  FROM workspaces w
  LEFT JOIN sessions s ON s.workspace_id = w.id
  GROUP BY w.id
)
SELECT * FROM workspace_agg
WHERE (${cursorTs}::timestamptz IS NULL OR (last_session_at, id) < (${cursorTs},
${cursorId}))
ORDER BY last_session_at DESC NULLS LAST, id DESC
LIMIT ${fetchLimit}
```

Pagination uses compound keyset `(last_session_at, id)` for stable ordering even when multiple workspaces have the same last activity time.

`GET /api/workspaces/:id` — Workspace detail

The `:id` parameter is polymorphic:

1. **26-char alphanumeric** → ULID lookup on primary key
2. **Contains `/`** → Treated as canonical_id (exact match)
3. **Anything else** → Case-insensitive match on `display_name` or exact match on `canonical_id`

Returns 400 if multiple workspaces match the same display_name (e.g., two repos both named "api").

Response includes 4 parallel queries:

```
{
  "workspace": { /* full Workspace object */ },
  "recent_sessions": [ /* last 10 sessions with device names */ ],
  "devices": [ /* linked devices with hook status */ ],
  "git_summary": {
    "total_commits": 150,
    "total_pushes": 30,
    "active_branches": ["main", "feature-x"],
    "last_commit_at": "2026-02-27T10:00:00Z"
  },
  "stats": {
    "total_sessions": 42,
    "active_sessions": 1,
    "total_duration_ms": 360000,
    "total_cost_usd": 3.14,
    "total_messages": 1200,
    "total_tokens_in": 500000,
    "total_tokens_out": 125000,
    "first_session_at": "2026-02-01T00:00:00Z",
    "last_session_at": "2026-02-27T12:00:00Z"
  }
}
```

8.2 CLI Commands

`fuel-code workspaces` — List command

`packages/cli/src/commands/workspaces.ts` :

Displays a table:

```
WORKSPACE  SESSIONS  ACTIVE  DEVICES  LAST ACTIVITY  TOTAL TOKENS  TOTAL TIME
fuel-code        42       1        2        2m ago      500K/125K     6h 0m
other-repo       10       0        1        3d ago      120K/30K      1h 30m
```

- Workspaces with active sessions get **bold** names
- Supports `--json` for machine output
- Always fetches all workspaces (limit: 250) — no pagination in CLI

`fuel-code workspace <name>` — Detail command

```
fuel-code
  Canonical ID:  github.com/user/fuel-code
  Branch:        main
  First seen:    26d ago
  Sessions:      42  |  Tokens: 500K/125K  |  Time: 6h 0m

Devices:
  macbook-pro (local)  CC: ✓  Git: ✓  last active: 2m ago

Recent Sessions:
  STATUS      DEVICE      DURATION  TOKENS  STARTED   SUMMARY
  summarized   macbook-pro  15m       50K/12K 2m ago    Fixed auth bug
  summarized   macbook-pro  30m       100K/25K 1h ago   Added workspace view

Git Activity:
  Commits: 150  |  Pushes: 30
  Active branches: main, feature-x
  Last commit: 2m ago
```

---

9. Name Resolution (CLI Fuzzy Matching)

`packages/cli/src/lib/resolvers.ts` :

The `resolveWorkspaceName(api, nameOrId)` function implements a 5-step resolution cascade:

```
1. Is it a ULID? (26-char uppercase alphanumeric)
   → Return as-is, no API call needed

2. Exact match on display_name (case-insensitive)?
   → Return matching workspace's ULID

3. Exact match on canonical_id (case-insensitive)?
   → Return matching workspace's ULID

4. Single prefix match on display_name?
   → Return the unique match's ULID

5. Multiple prefix matches?
   → Throw ApiError(400) with candidate list

6. No matches at all?
   → Throw ApiError(404) with available workspace names
```

This allows users to type `fuel-code workspace fu` and get `fuel-code` if it's the only workspace starting with "fu".

The same resolution pattern is used for devices via `resolveDeviceName()` — same file, same cascade logic.

---

10. Gap Analysis: What's Missing or Misaligned

Gap 1: `hooks_installed` Ghost Column

Severity: Low

The `workspace_devices` table has a `hooks_installed` column from the initial migration. It's never read or written by any code. The functional column is `git_hooks_installed` . The ghost column exists alongside it, causing confusion about which one to use.

Impact: Dead schema weight. No functional consequence but misleading for anyone reading the migration.

Gap 2: Display Name Collisions Unresolved

Severity: Medium

Two workspaces with different canonical IDs can have the same `display_name`:

- `github.com/alice/api` → display_name: `"api"`
- `github.com/bob/api` → display_name: `"api"`

The API handles this by returning 400 "ambiguous workspace name" when looking up by display name. But:

- The workspaces list view shows both as `"api"` with no distinguishing info
- The CLI `workspaces` command doesn't show canonical_id in the table
- Users have to use `fuel-code workspace github.com/alice/api` to disambiguate

Potential fix: Show `"alice/api"` and `"bob/api"` (include parent path segment) when collisions exist.

Gap 3: No Workspace Rename or Delete

Severity: Low

There is no endpoint or CLI command to:

- Rename a workspace (change `display_name`)
- Delete a workspace (with cascade to sessions, events, etc.)
- Merge two workspaces (e.g., if a repo changed its remote URL)

Workspaces can only be created implicitly via the event processor.

Gap 4: `default_branch` Never Updated

Severity: Low

The `default_branch` is captured from the first `session.start` event and never updated. If a repo changes its default branch from `master` to `main`, the workspace still shows `master`. The ON CONFLICT clause explicitly does NOT update this field.

Gap 5: No Workspace-Level Settings

Severity: Low

There's no per-workspace configuration (e.g., "use git hooks in this repo", "tag all sessions in this workspace as X"). The `metadata` JSONB column could theoretically hold settings, but nothing reads workspace metadata for behavioral configuration.

Gap 6: Bash/TS Normalization Port Handling Divergence

Severity: Medium

As documented in Section 7.3, the bash hook doesn't strip ports from SSH URLs while the TS normalization does. An SSH URL with an explicit port (e.g., from a self-hosted GitLab behind a non-standard port) would produce different canonical IDs from git hooks vs CC hooks.

Impact: Git commits from the bash hook would go to workspace `host:2222/repo`, while CC sessions from the TS hook would go to workspace `host/repo`. Two workspaces for the same repo.

Gap 7: Case Sensitivity in Canonical ID

Severity: Low-Medium

The canonical ID preserves case in the path portion (`github.com/User/Repo` vs `github.com/user/repo`). GitHub is case-insensitive for paths, so these point to the same repo but produce different canonical IDs.

Impact: If a user clones with `git@github.com:User/Repo.git` on one machine and `git@github.com:user/repo.git` on another, they get two separate workspaces. This is a deliberate conservative choice (other Git hosts may be case-sensitive) but can cause confusion with GitHub.

Summary Table

Gap	Severity	Impact	Fix Complexity
`hooks_installed` ghost column	Low	Dead schema weight	Low (migration to drop)
Display name collisions	Medium	CLI UX confusion	Low (show parent segment)
No rename/delete/merge	Low	Cannot clean up workspaces	Medium
`default_branch` frozen	Low	Stale branch info	Low (update on conflict)
No workspace settings	Low	Future extensibility	Medium
Bash/TS port divergence	Medium	Duplicate workspaces	Low (add port strip to bash)
Path case sensitivity	Low-Med	GitHub-specific confusion	Medium (normalize GitHub paths)

---

11. References

#	File	Description
1	`packages/shared/src/canonical.ts`	Canonical ID derivation `normalizeGitRemote()`, `deriveWorkspaceCanonic` `deriveDisplayName()`
2	`packages/shared/src/types/workspace.ts`	TypeScript type definit `UNASSOCIATED_WORKSPACE` constant
3	`packages/core/src/workspace-resolver.ts`	`resolveOrCreateWorkspa` `getWorkspaceByCanonica` `getWorkspaceById()`
4	`packages/core/src/workspace-device-link.ts`	`ensureWorkspaceDeviceL` — junction table upsert
5	`packages/core/src/event-processor.ts`	Event processing pipeli workspace resolution s
6	`packages/server/src/db/migrations/001_initial.sql`	Database schema: `workspaces`, `workspace_d`

		tables
7	`packages/server/src/db/migrations/004_add_git_hooks_prompt_columns.sql`	Hook prompt columns `workspace_devices`
8	`packages/server/src/routes/workspaces.ts`	API: `GET /api/workspace` (list), `GET /api/workspaces/:id` (de
9	`packages/cli/src/commands/workspaces.ts`	CLI: `fuel-code workspac` (list), `fuel-code workspa` `<name>` (detail)
10	`packages/cli/src/lib/resolvers.ts`	`resolveWorkspaceName()` fuzzy name/ULID resolu
11	`packages/hooks/git/resolve-workspace.sh`	Bash workspace resolu for git hooks
12	`packages/hooks/claude/_helpers/resolve-workspace.ts`	TS workspace resolutic CC hooks
13	`packages/cli/src/lib/api-client.ts`	API client: `listWorkspac` `getWorkspace()`