Software Project Naming Standards

A Practical Reference for Consistent Code

Abstract

A practical reference for maintaining consistent, readable, and professional naming across your projects. These standards reduce cognitive overhead, make codebases easier to navigate, and ensure your work remains understandable to your future self — not just collaborators. Designed to be lightweight enough for a solo developer while laying the groundwork for scaling to a team.

Contents

1 Repositories & Folders

1.1 Repository Names

- Use **lowercase kebab-case** (words separated by hyphens): `my-project`, `api-service`, `data-pipeline`
- Be descriptive but concise — aim for 2–4 words
- Prefix with a category where helpful: `lib-`, `api-`, `cli-`, `app-`
- Avoid generic names like `project1`, `test`, or `new-repo`

Examples:

Good	× Avoid
`invoice-parser`	`InvoiceParser`
`api-auth-service`	`myproject_v2`
`cli-deploy-tool`	`stuff`
`lib-date-utils`	`NEW_repo`

1.2 Folder / Directory Names

- Use **lowercase kebab-case** for project directories: `src/`, `user-profiles/`, `email-templates/`
- Use conventional names for standard directories
- Group files by **feature** rather than type in larger projects (e.g., `src/auth/` rather than putting all models in `src/models/`)

Standard directories:

Directory	Purpose
`src/`	Source code
`tests/`	Test files
`docs/`	Documentation
`assets/`	Static files (images, fonts, etc.)
`scripts/`	Utility/build scripts
`config/`	Configuration files
`dist/` or `build/`	Compiled output

2 Branches & Commits

2.1 Branch Names

Follow the pattern: `<type>/<short-description>`

Branch types:

Type	Use for
`feature/`	New features or enhancements
`fix/`	Bug fixes
`hotfix/`	Urgent production fixes
`chore/`	Maintenance, refactoring, dependency updates
`docs/`	Documentation changes only
`test/`	Adding or updating tests
`experiment/`	Exploratory / throwaway work

Examples:

```
feature/user-authentication
fix/null-pointer-on-login
hotfix/payment-gateway-timeout
chore/upgrade-node-18
docs/update-readme-setup
```

Rules:

- Use **lowercase kebab-case** only
- Keep descriptions short but meaningful (3–5 words max)
- Reference a ticket/issue number if applicable: `fix/gh-42-login-redirect`
- Avoid vague names like `my-branch`, `wip`, or `temp`

2.2 Commit Messages

Follow the **Conventional Commits** format:

```
<type>(<scope>):  <short summary>
```

Commit types:

Type	Use for
`feat`	A new feature
`fix`	A bug fix
`docs`	Documentation only
`style`	Formatting, no logic change
`refactor`	Code restructure, no feature/fix
`test`	Adding or updating tests
`chore`	Build process, dependencies
`perf`	Performance improvements

Examples:

```
feat(auth): add JWT refresh token support
fix(api): handle empty response from payment gateway
docs(readme): add local setup instructions
refactor(db): extract query builder into utility module
chore: upgrade eslint to v9
```

Rules:

- Use the **imperative mood** in the summary: "add", not "added" or "adds"
- Keep the summary under 72 characters
- Use the body (optional) for *why*, not *what* — the diff already shows what changed
- Never commit with messages like `fix`, `update`, `changes`, or `asdfgh`

3 Variables & Functions

3.1 General Rules

- Prioritize **clarity over brevity** — `userAuthToken` beats `uat`
- Avoid single-letter names except for loop counters (`i`, `j`) or mathematical functions (`x`, `y`)
- Be consistent with the conventions of the language you're using

3.2 Variables

Use **camelCase** in most languages (JavaScript, TypeScript, Java, Swift):

```
const userEmail = "hello@example.com";
let isAuthenticated = false;
const maxRetryCount = 3;
```

Use **snake_case** in Python and Ruby:

```
user_email = "hello@example.com"
is_authenticated = False
max_retry_count = 3
```

Constants — use `UPPER_SNAKE_CASE`:

```
const MAX_FILE_SIZE_MB = 10;
const API_BASE_URL = "https://api.example.com";
```

Boolean variables — prefix with `is`, `has`, `should`, `can`, or `will`:

```
isLoading, hasPermission, shouldRedirect, canEdit, willExpire
```

3.3 Functions & Methods

- Use **verbs** — functions do things: `getUser()`, `validateEmail()`, `sendNotification()`
- Be specific: `fetchUserById()` is better than `getUser()`

Common prefixes:

Prefix	Use for
`get` / `fetch`	Retrieve data
`set`	Assign a value
`create` / `build`	Construct something new
`update` / `save`	Modify existing data
`delete` / `remove`	Remove data
`validate` / `check`	Verify something
`handle` / `on`	Event handlers
`format` / `parse`	Transform data
`is` / `has`	Return a boolean

Examples:

```
function fetchUserById(userId) { ... }
function validateEmailFormat(email) { ... }
function handleFormSubmit(event) { ... }
function formatCurrencyUSD(amount) { ... }
```

3.4 Classes & Types

Use **PascalCase:**

```
class UserAuthService { ... }
type ApiResponse = { ... }
interface DatabaseConfig { ... }
enum UserRole { Admin, Editor, Viewer }
```

4 Databases & Tables

4.1 Table Names

- Use **lowercase snake_case**
- Use **plural nouns**: `users`, `orders`, `product_categories`
- Be descriptive: `email_verification_tokens` not `evt`
- Avoid generic names like `data`, `records`, `items`

Examples:

Good	× Avoid
`users`	`User`
`order_items`	`tbl_order_items`
`refresh_tokens`	`tokens2`
`product_categories`	`prodCat`

Note: Some ORMs and conventions use singular table names (`user` instead of `users`). Either is acceptable — pick one and stick with it.

4.2 Column Names

- Use **lowercase snake_case**: `first_name`, `created_at`, `is_active`
- Be explicit about units where relevant: `file_size_bytes`, `timeout_seconds`
- Avoid abbreviations: `description` not `desc`, `quantity` not `qty`

Standard column conventions:

Column	Purpose
`id`	Primary key (or `user_id` for clarity in joins)
`created_at`	Timestamp of record creation
`updated_at`	Timestamp of last update
`deleted_at`	Soft delete timestamp (nullable)
`is_active`	Boolean status flag

4.3 Indexes & Constraints

Follow the pattern: `<type>_<table>_<columns>`

```
-- Index
idx_users_email
idx_orders_created_at

-- Unique constraint
uq_users_email

-- Foreign key
fk_orders_user_id

-- Primary key
pk_users
```

4.4 Database & Schema Names

- Use **lowercase snake_case:** `my_app_db`, `analytics_schema`
- Separate environments clearly: `myapp_dev`, `myapp_staging`, `myapp_prod`

5 Quick Reference Cheat Sheet

Item	Convention	Example
Repository	`kebab-case`	`invoice-parser`
Directory	`kebab-case`	`user-profiles/`
Git branch	`type/kebab-case`	`feature/add-login`
Commit	`type(scope): summary`	`fix(auth): handle expired token`
Variable (JS/TS)	`camelCase`	`userEmail`
Variable (Python)	`snake_case`	`user_email`
Constant	`UPPER_SNAKE_CASE`	`MAX_RETRY_COUNT`
Function	`camelCase` verb	`fetchUserById()`
Class / Type	`PascalCase`	`UserAuthService`
DB table	`plural_snake_case`	`order_items`
DB column	`snake_case`	`created_at`
DB index	`idx_table_column`	`idx_users_email`

Consistency matters more than perfection.
When in doubt, follow the existing pattern in your codebase.