

From Craft to Engineering: Standardizing Prompts for DevOps Teams

Implementing a versioned, multi-level, and validated
source of truth for your AI code assistants.

{PJ} Prompt Unifier

The Current Chaos: Managing Prompts as a Team

Without a centralized system, prompts and coding rules become technical debt. They are scattered, difficult to maintain, and inconsistent.



Local Files

Valuable prompts stored in precise files on individual machines, inaccessible to others.



Wikis and Documentation

Often obsolete, without versioning or a clear validation process.



Discussion Threads (Chat)

'Magic recipes' shared in an ephemeral and hard-to-find way.



Lack of Versioning

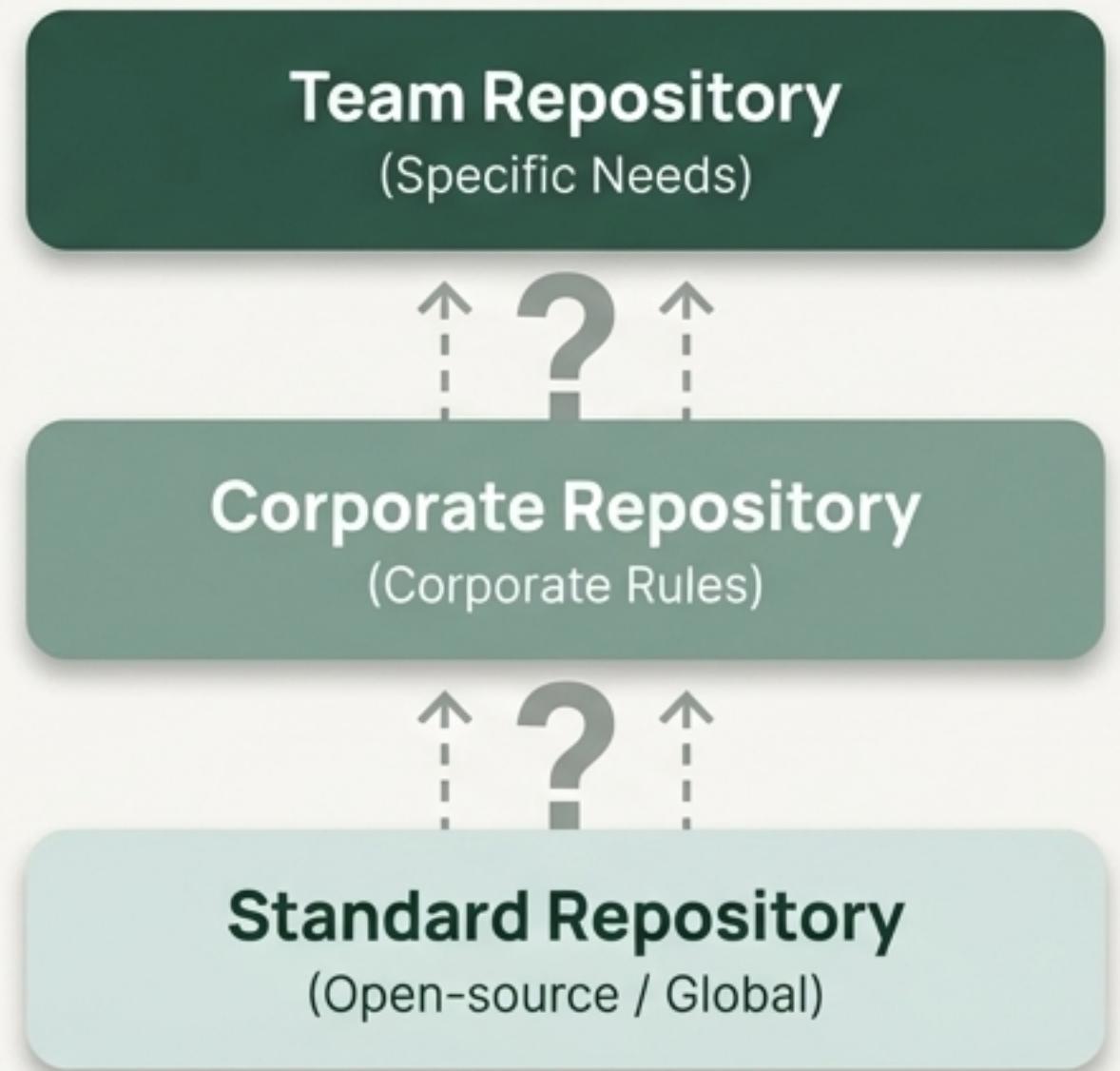
No traceability of changes. How do you know which version of a prompt generated a specific result?



Lack of Standardization

Each developer uses their own variants, leading to inconsistent results and code quality.

The Challenge of Governance at Scale

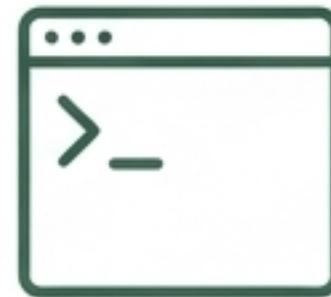


**How do you reconcile global standards,
corporate rules, and team-specific needs?**

The Solution: The Prompt Unifier GitOps Ecosystem

A GitOps approach to prompt management, transforming chaos into a structured, collaborative, and reliable workflow. The ecosystem is built on two pillars: a CLI tool and a repository standard.

The Engine



``prompt-unifier``

The Orchestration Engine

A Python CLI tool to synchronize, validate, and deploy your prompts and rules from multiple sources.



The Data



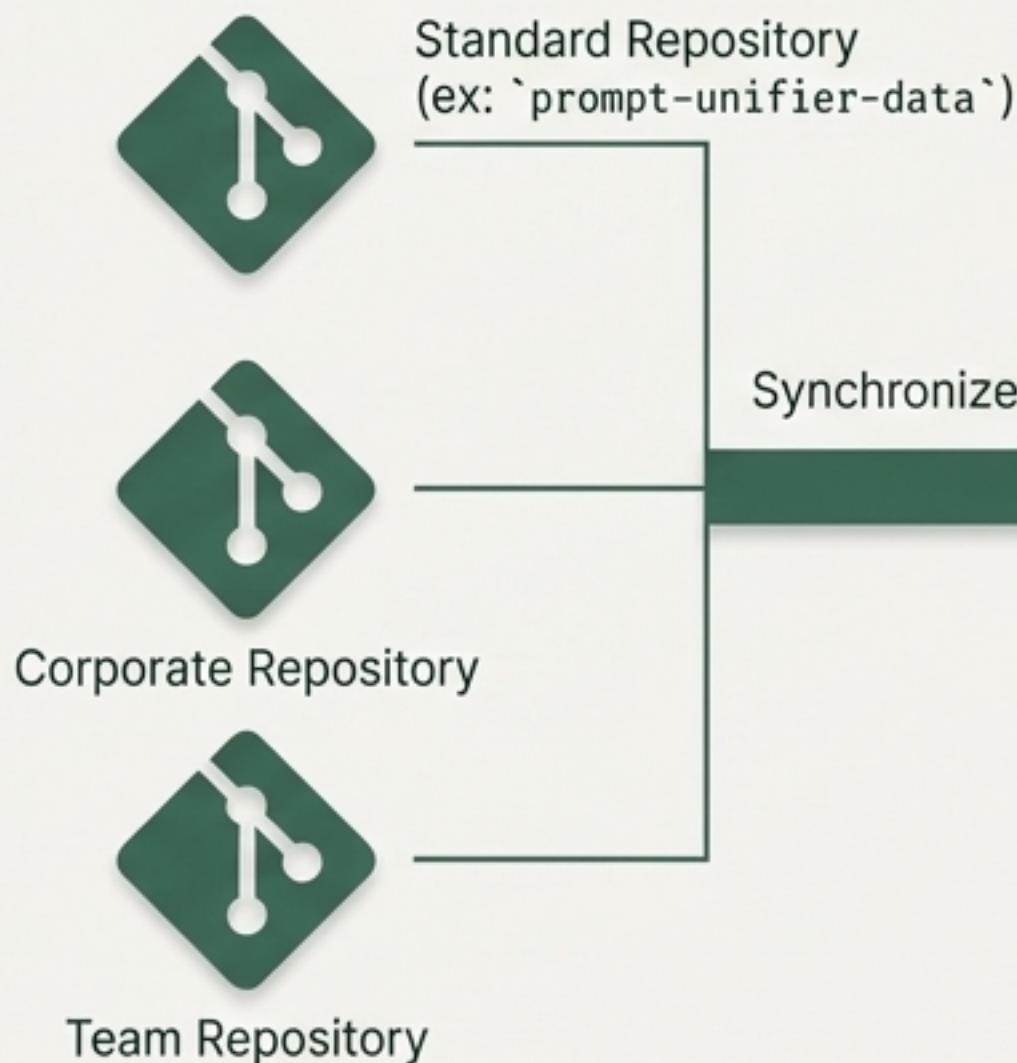
``prompt-unifier-data``

The Source of Truth

A structured Git repository serving as a centralized and versioned collection for all prompts and rules.

A Unified Workflow for Multiple Sources

Phase 1: Remote Git Repositories

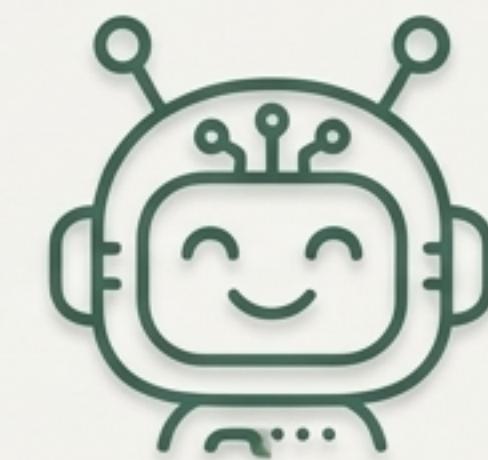


Phase 2: Centralized Local Storage



Merged prompts and rules.
"Last-wins" strategy: the team repo overrides the corporate, which overrides the standard.

Phase 3: Active Use



Prompts and rules ready
for use in your tools
(Continue, Kilo Code, etc.).

Zoom #1: Anatomy of a Data Repository

A clear and predictable structure, consistent across all source repositories (standard, corporate, team).

```
└── prompts/ // Templates to instruct the AI
    ├── airflow/
    ├── kubernetes/
    └── python/

└── rules/   // Standards and best practices
    ├── global/
    ├── airflow/
    └── python/
```

For `prompts/`

Contains the templates used to instruct AI models for specific tasks (generation, refactoring, analysis, etc.).

For `rules/`

Contains guides, standards, and best practices. These files serve as enriched context for the AI, ensuring that the results adhere to team norms.

Zoom #2: Anatomy of a Rule: Context and Standards

'rules/airflow/01-airflow-dag-standards.md'

```
---  
title: "Airflow DAG Standards"  
description: "Core standards for designing and  
structuring Airflow DAGs..."  
tags: [airflow, dag, standards]  
version: 1.0.0  
category: "standards"  
applies_to: ["dags/**/*.py"]  
---
```

Airflow DAG Standards

1. Core DAG Principles

Idempotency and Determinism

- **Idempotency**: Every task must be idempotent...

Structured Metadata (YAML Frontmatter): Allows for categorizing, versioning, and filtering rules. The `applies_to` field links the rule to specific files.

Readable Content (Markdown): Easy to write and maintain for humans.

Context for the AI: Can be injected into a prompt to guide code generation according to defined standards.

Zoom #3: Anatomy of a Prompt: The S.C.A.F.F. Framework

Prompts are templates for instructing an AI. They follow the **S.C.A.F.F.** framework to ensure clarity, context, and precision.

S	C	A	F	F
Situation What is the starting context?	Challenge What is the precise task to accomplish?	Audience Who is the result for?	Format What is the expected output structure?	Foundations What are the rules or principles to respect?

prompts/python/01-python-refactoring.md

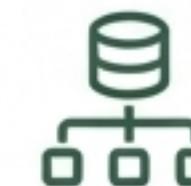
```
---
title: "Python Refactoring Assistant"
description: "Refactor Python code to improve readability..."
---
You are an expert Python developer...
### Situation
The user provides a piece of Python code...
### Challenge
Rewrite the provided code to improve its quality...
```



The Engine: `prompt-unifier`, the CLI at the Heart of the System



Git Versioning



Centralized Management



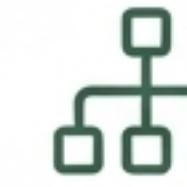
Integrated Validation



Easy Deployment



Multi-Repository Support



Structured Organization

Harmonize standards while preserving team autonomy.

The CLI in Action (1/3): Multi-Level Synchronization

init

Creates a `'.prompt-unifier/config.yaml'` configuration file and prepares the local storage directory.

```
prompt-unifier init
```

sync

Clones or updates prompts from multiple repositories. The "last wins" strategy is ideal for overriding standards.

```
# Synchronize multiple repositories (last one wins)
prompt-unifier sync \
    # 1. Base of standards (open-source)
    --repo https://gitlab.com/waewoo/prompt-unifier-data.git \
    # 2. Corporate standards (override the base)
    --repo https://gitlab.com/my-company/company-prompts.git \
    # 3. Team prompts (override corporate)
    --repo https://gitlab.com/my-team/team-prompts.git
```

The CLI in Action (2/3): Verification and Exploration

Validate the syntax and conformity of your files *after* they are merged, and easily explore the available content.

validate

Verifies the YAML frontmatter syntax, the presence of required fields, and the structure of files across the entire merged content.

```
# Validate the integrity of the synchronized storage
prompt-unifier validate
```

```
# Validate only prompts with detailed output
prompt-unifier -v validate --type prompts
```

list and status

Displays a table of all available prompts/rules.
Checks the synchronization status of your repositories.

```
# List all content sorted by date
prompt-unifier list --sort date
```

```
# Check if new commits are available
prompt-unifier status
```

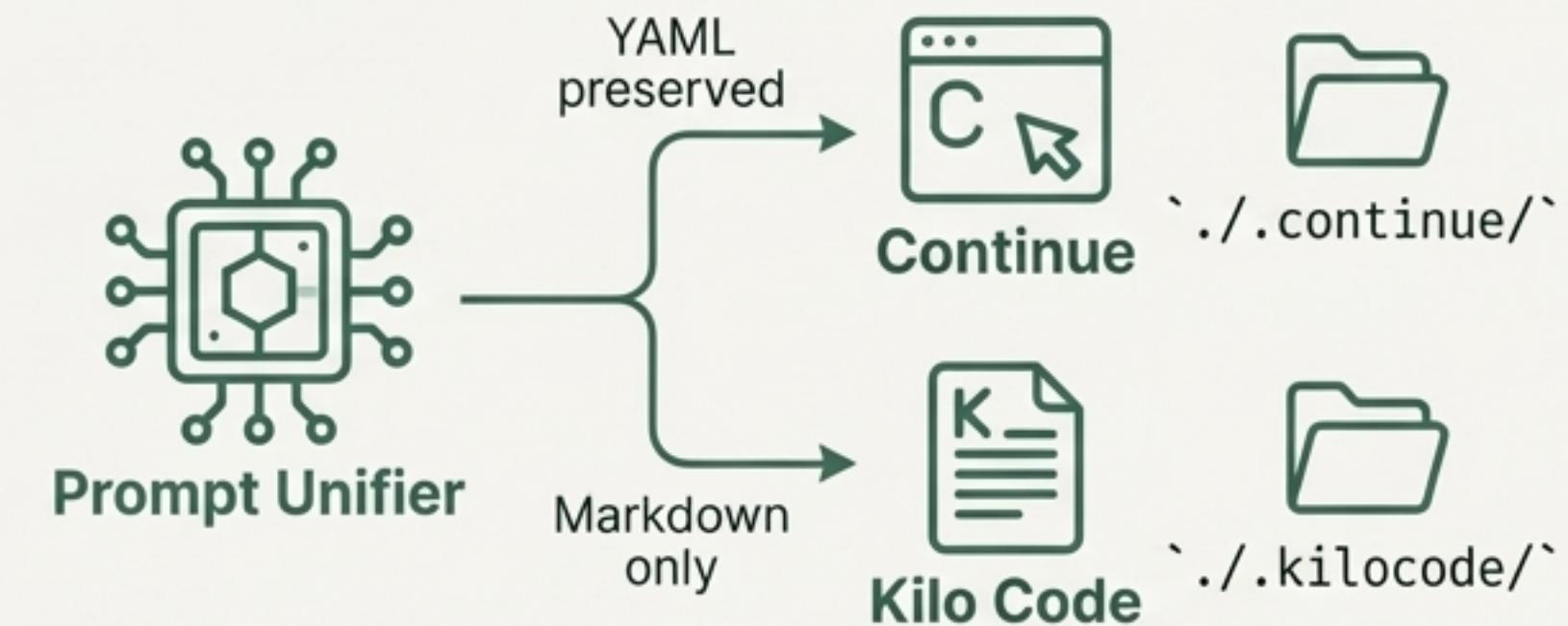
The CLI in Action (3/3): Deployment to Your Tools

Deploy your validated prompts and rules to your AI code assistants. The system handles the specifics of each tool via 'handlers'.

Supported Handlers

Continue: Full support, preserves YAML frontmatter. Destination: `./.continue/`

Kilo Code: Converts to pure Markdown (no frontmatter). Destination: `./.kilocode/`



Advanced Deployment Examples

```
# Deploy only prompts tagged "python"
prompt-unifier deploy --tags python

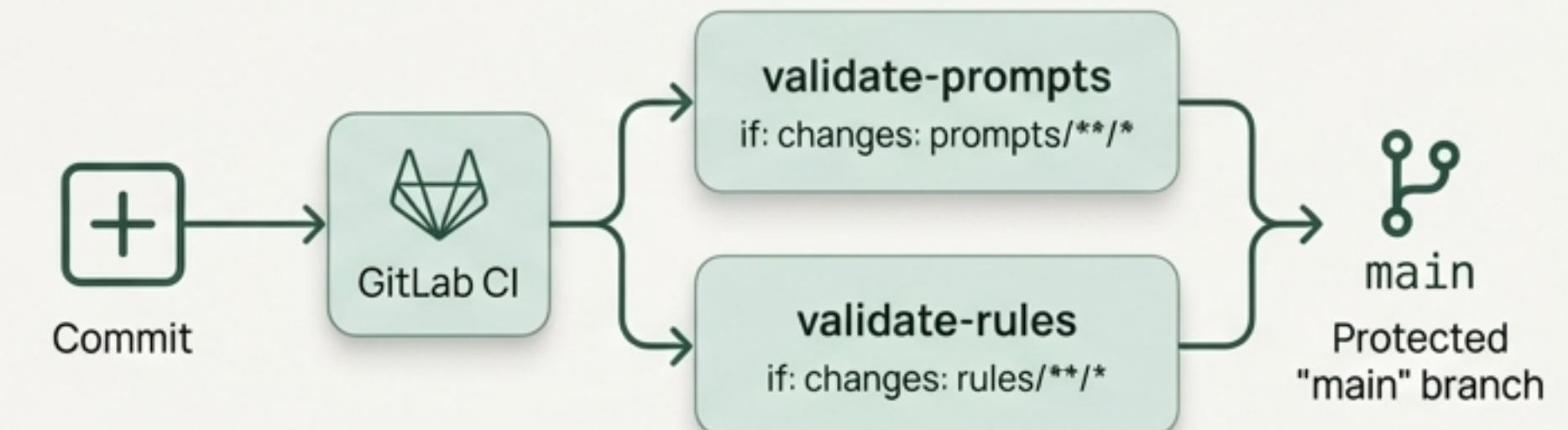
# Deploy to Kilo Code and clean up old files
prompt-unifier deploy --handlers kilocode --clean

# Simulate a deployment to see the changes
prompt-unifier deploy --dry-run
```

Quality at Scale: Continuous Integration (CI/CD)

The `prompt-unifier-data` repository includes a GitLab CI pipeline that guarantees every contribution is automatically validated, preventing errors from reaching the main branch.

- **Quality Automation:** No manual validation required.
- **Fast Feedback:** Developers know immediately if their changes are compliant.
- **Confidence and Reliability:** Only validated prompts and rules are merged, guaranteeing the stability of the source of truth.



Take Action: Adopt a Structured Approach

Quick Start Steps

1. Install the CLI

```
pip install prompt-unifier
```

2. Initialize in your project

```
prompt-unifier init
```

3. Use the example repo as a base

Fork the repository to create your own corporate or team library.

`<https://gitlab.com/waewoo/prompt-unifier-data>`

Contribution

This project is open source (MIT License). Your contributions are welcome.

Before submitting, please ensure your changes pass local validation:

```
make validate
```

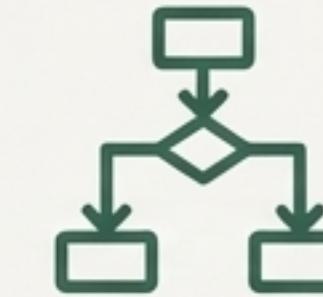
The Transformation: From Craft to Prompt Engineering

The Prompt Chaos



Inconsistent, Opaque, Fragile, Isolated.

Prompt Engineering



Standardized, Reliable, Collaborative, Scalable.

The Prompt Unifier ecosystem provides the structure and tools necessary to make AI assistance a reliable engineering discipline within your team.