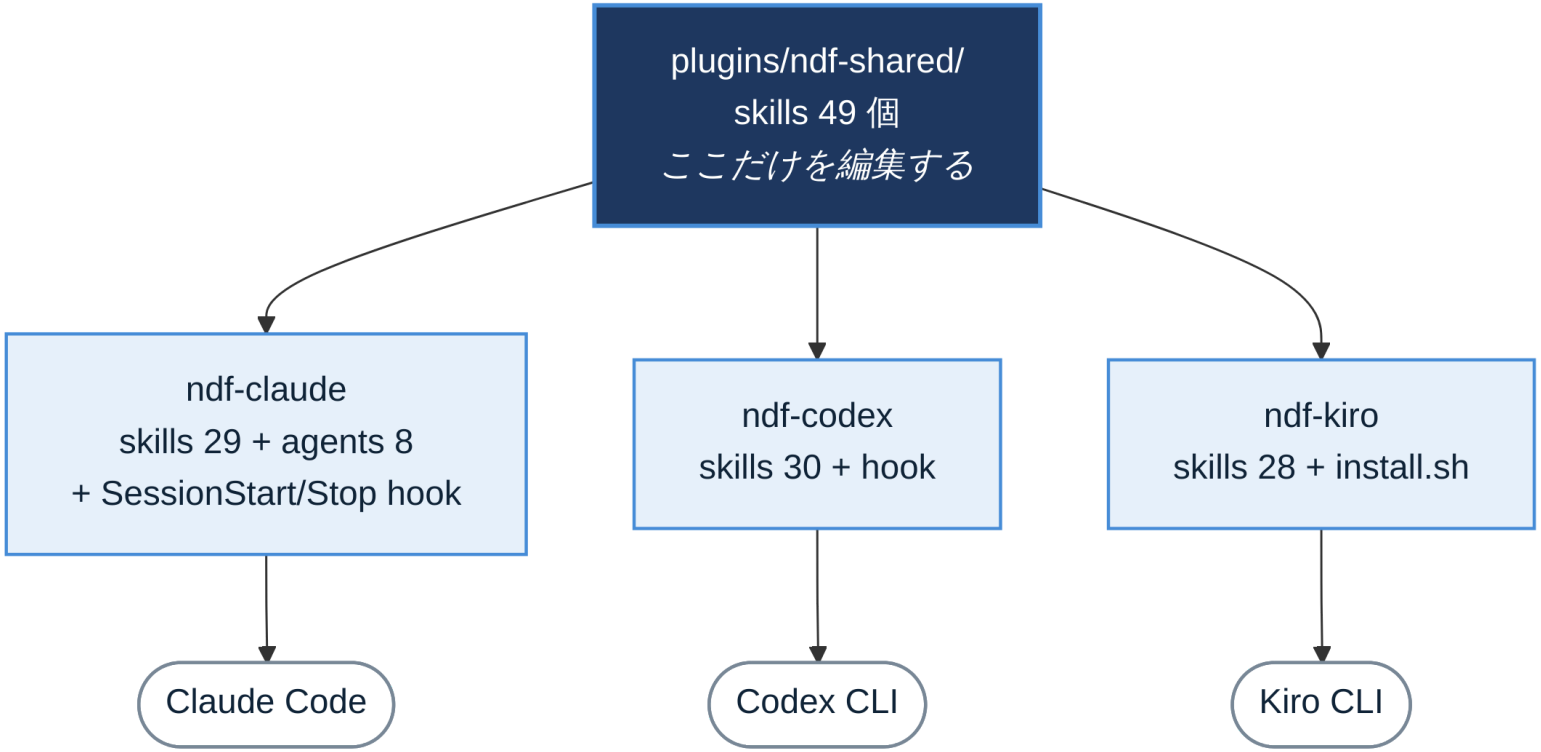
ai-plugins

ndfを使った開発ワークフロー

2026-08-06

Claude Code・Codex CLI・Kiro CLI 対応

まず全体像を1枚で



編集するのは `plugins/ndf-shared/` の **1か所だけ**。そこから3ランタイム分の配布物が生成されます。
ほかに MCP プラグインが **10個**（Serena / BigQuery / Playwright / Chrome DevTools / Redash など）。

インストールは2コマンド（Claude Code）

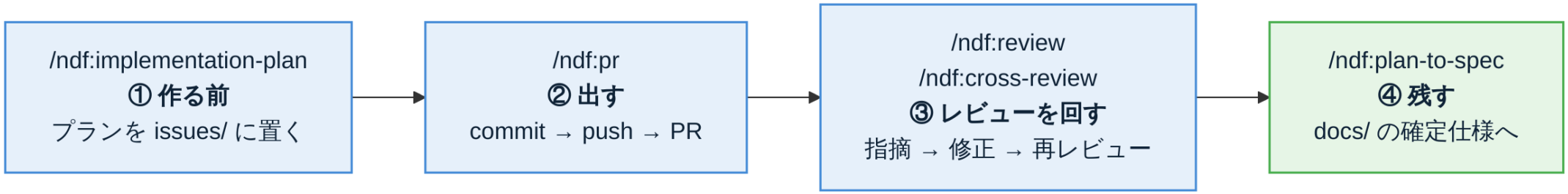
```
# 1. マーケットプレイスを登録（初回だけ）
/plugin marketplace add https://github.com/devbasex/ai-plugins

# 2. NDFを入れる
/plugin install ndf@ai-plugins
```

入れると使えるようになるもの:

スキル 29個	`/ndf:pr`, `/ndf:review` … スラッシュで直接呼べる
エージェント 8個	director / corder / qa / debugger / devops-engineer など
フック 2種	SessionStart（transcript保持を90日に維持）/ Stop（AI要約+Slack通知）

今日の地図: PRを1本出すまで



この4つに加えて、**全工程に効く「書き方・調べ方」のルール**が3つあります。

スキル	何を揃えるか
`/ndf:markdown-writing`	会話も検討過程も知らない第三者に伝わる書き方
`/ndf:investigation-rules`	「無い」と書くならエビデンスを添える
`/ndf:problem-solving`	つじつま合わせをせず、上流で直す

① `/ndf:implementation-plan` — 作る前に書く

書いてあること

`issues/` に置くプランの**雛形**。
概要 / 問題・背景 / 修正対象 /
タスク分解 / 影響範囲 / テスト計画。

作るケース

複数ファイル・新機能・
既存ロジックの大幅変更・DBマイグレーション

作らないケース

typo / 設定値だけ / 1ファイルの軽微な修正

重視していること

「なぜ」を残すこと。
後任か将来の自分が変更意図を追える状態にする。

プランとPR本文の役割を分けています。

	役割
プラン	「なぜ」「どう分解するか」の永続記録
PR本文	「何をやったか」のレビュー用サマリ

> 書かずに実装しても、PR作成時に**会話履歴 + `git log` + `git diff` から自動生成**します。

② `/ndf:pr` — commit から PR まで

```
/ndf:pr                          # main へ通常PR
/ndf:pr --draft                  # ドラフトPR
/ndf:pr "エクスポートAPIを追加"  # コミットメッセージ指定
/ndf:pr qa/staging               # base非main → cherry-pick-pr へ誘導
```

書いてあること

引数の解釈ルール（`--draft` / ブランチ名 / それ以外はコミットメッセージ）、PR本文の組み立て方、既存PRがある場合の**本文更新**手順。

重視していること

事故を止めること。

デフォルトブランチへの直接コミットを拒否。`qa/*` などへの直接PRも止めて `/ndf:cherry-pick-pr` に誘導します。

> 検証ブランチへ直接PRを出すと、そのブランチをマージしたときに環境固有コードが main に混ざる。これを構造的に防いでいます。

③ `/ndf:review` — 何を優先して見るか

```
/ndf:review 123             # Claude 自身がレビュー
/ndf:review 123 codex       # Codex CLI に委譲
/ndf:review 123 gemini      # Gemini CLI に委譲
```

レビュー観点は順番が決まっています

言語慣用性 → 可読性 → コード品質 → 保守性 → セキュリティ → テストカバレッジ

具体的なチェックポイントも明記されています。

- その言語らしい書き方（イディオム・標準ライブラリの活用）
- メモリ効率と演算性能（不要なループ・コピーの排除、キャッシュ）
- 関数50行・ファイル300行が目安。ただしプロジェクトの慣例が優先
- 重複コードは**PRの範囲にこだわらず**まとめるよう指摘する
- 柔軟性を損なう定数化を避け、DBのマスタや json / yaml への外部化を検討する

③ `/ndf:review` — 指摘の書き方が決まっている

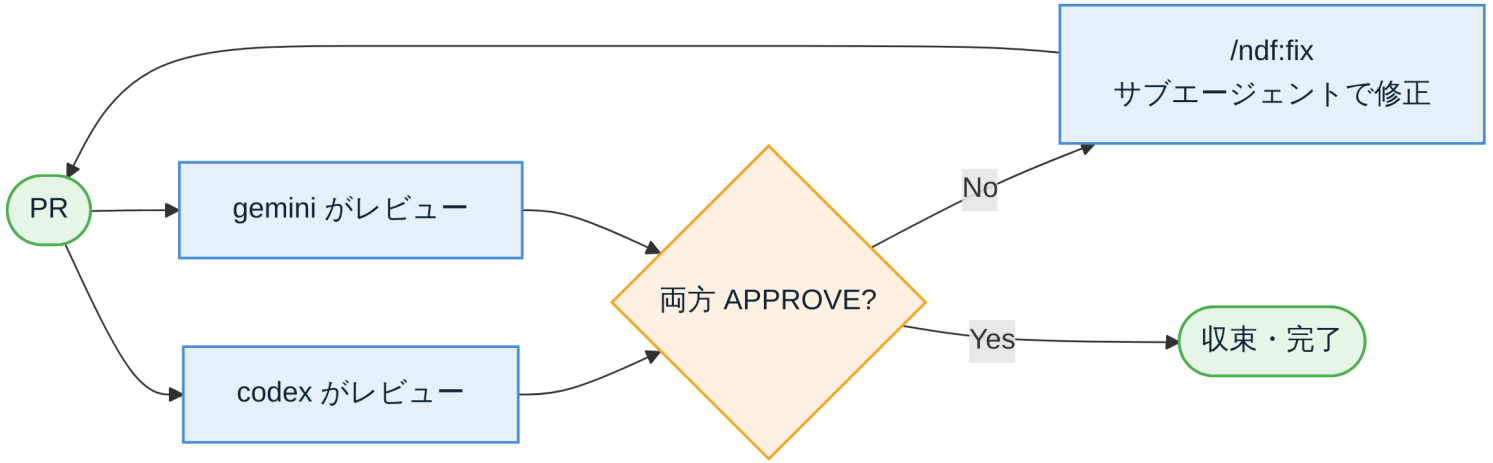
各インラインコメントの先頭に `[重要度 / カテゴリ]` を付けます。

```
[critical / セキュリティ] SQL がエスケープなしで連結されている。プレースホルダ必須。
[major / 可読性] 70 行関数。〇〇 と △△ に分割を推奨。
[minor / 言語慣用性] Python なら内包表記で 1 行化可能。
[nit / スタイル] スペースが揃っていない。
```

重要度	定義	後段の扱い
`critical`	セキュリティ・データ破損・本番障害につながる	**必ず自動修正**
`major`	保守性・性能・仕様逸脱の重要問題	**必ず自動修正**
`minor`	改善推奨だがブロッカーではない	明らかな改善のみ自動修正
`nit`	好み・スタイル	**修正しない。最後にユーザー判断へ**

> 総評ではなく**コード行に紐付くインラインコメント**を原則にしています。総評に書けるのは設計レベルの所見だけ。

④ `/ndf:cross-review` — 両AIが納得するまで回す



最大12ラウンド。8ラウンドで未収束ならPRをローテーション。修正はサブエージェントに投げるのでメインの会話は汚れません。

④ `/ndf:cross-review` — 観点はPRの中身で切り替わる

変更ファイルを全件取得して分類し、**該当する観点テンプレートだけ**を両AIに渡します。

分類	渡される観点
`docs_only`	説明の妥当性、コード・設定との整合
`db_migration`	型、NULL/default/制約/index、backfill、ロールバック
`api_contract`	互換性、schema、エラー形式、認可
`auth_security`	secret/PII、CSRF/CORS/JWT/OAuth
`performance`	N+1、I/O、ロック、cache、冪等性

ほかに `code` / `test` / `dependency` / `config_ci` / `frontend` / `deletion_rename` / `generated` / `i18n` / `infra` の全**15分類**。

```
/ndf:cross-review 123 \
  --focus "ドキュメントとコードの整合性"
```

今回だけの観点は `--focus` で追加。長いチェックリストはファイルで渡せます（`--extra-instructions-file`）。どちらも自動テンプレートの後ろに乗ります。

⑤ `/ndf:plan-to-spec` — プランを仕様書に変える

書いてあること

`docs/` 配下の標準章立て。
概要 / 背景 / 対象範囲 / 仕様 /
データ・設定 / 外部連携 / エラー処理 /
セキュリティ / 運用 / テスト観点 / 関連リンク

消すもの

実装タスクのチェックリスト、PR分割計画、
作業担当、破棄された方針、調査メモ、
AIエージェント向けの作業指示

重視していること

移動ではなく書き直し。
プランは作業中の意思決定記録なので、
完了後は「今のコードと一致する記述」だけを残す。

語尾まで変換ルールがあります。

プラン	仕様書
実装する	提供する / 保持する
修正対象	構成 / 関連ファイル
テスト計画	テスト観点 / 検証方法

> 書いたあとに**実コードと照合**します。書いた設定名やファイルパスが実在するか、実装にある重要な挙動が抜けていないか。

⑥ `/ndf:markdown-writing` — 第三者に伝わる書き方

読み手は**会話もコードベースも検討過程も知らない第三者**、という前提で書かせるルール。

#	ルール	ひとこと
1	説明文に内部識別子・略語を持ち込まない	❌ `user_subscriptions` の `plan_id` を更新 ✅ 利用者の契約プランを変更
2	検討過程の痕跡を残さない	「案A」「壁打ちの結果」「先ほど決めた」
3	変更履歴を本文に残さない	「以前はXだったが指摘を受けてYに変更した」
4	否定的な結論にはエビデンスを添える	未確認は「残リスク」として明示。省くと確認済みと読まれる
5	個人情報・機密情報を書かない	コミットメッセージは後から消しにくい
6	図はインラインで書く	mermaid / math。plantUML・HTML・`<svg>` は使わない
7	300行以内。501行以上は分割	ただし分割で理解が落ちるなら1ファイルのまま

書き終えた後の **grep セルフチェック**とチェックリストも用意されています。

⑦「無い」と言う前に — 調査とバグ修正のルール

`/ndf:investigation-rules`

否定的な結論にはエビデンス必須。

「カラムがない」「呼ばれていない」「データがない」は
読み手が最も検証しづらく、外れたときの被害が大きい。

主張	必須エビデンス
カラムが存在しない	`SHOW COLUMNS` の結果
データが存在しない	`COUNT(*)` の結果
呼び出し箇所がない	検索コマンドと結果

エビデンスなしで残課題の優先度を「低」にするのは誤判断の典型、と名指しされています。

`/ndf:problem-solving`

上流で直す。つじつま合わせをしない。

❌
異常データ → migrationで論理削除 → 再実行

✅
異常データ → なぜ入ったか調査
→ 取り込みロジックにバリデーション追加
→ 論理削除 → 再実行

修正は**コード → デプロイ → データ修復**の順。
根本原因の修正とデータ修復は別コミットにする（Revertしやすい）。

Codex / Kiro でも同じスキルが使えます

Codex CLI

```
codex plugin marketplace add \
  https://github.com/devbasex/ai-plugins
codex plugin add ndf@ai-plugins
```

セッション内では `ndf:` 接頭辞で
30個のスキルが使えます。

Claude版との差: エージェント8個とSessionStart/Stopフックは無し。代わりに Playwright 系スキル5個が入ります。

Kiro CLI

```
git clone \
  https://github.com/devbasex/ai-plugins.git
bash plugins/ndf-kiro/install.sh
kiro-cli chat
```

`.kiro/skills/` に**28個**、
`.kiro/agents/default.json` を生成。

`--with-slack` でStop時のSlack通知、`--with-codex` でCodex CLI連携を追加。何度実行しても安全（冪等）。

まとめ: スキルは「判断基準」が本体

流れ

`implementation-plan` → `pr`
→ `review` / `cross-review`
→ `plan-to-spec`

全工程に効くルール

`markdown-writing`（伝わる書き方）
`investigation-rules`（エビデンス主義）
`problem-solving`（上流で直す）

今日から試す3ステップ

1. 入れる
   `/plugin install ndf@ai-plugins`
2. 次のPRで `/ndf:pr` を叩く
3. 出したPRに `/ndf:review` を叩く

> どのスキルにも「何をやるか」より先に「何を重視するか」が書いてあります。
> それが人によってブレるところなので、そこだけ揃えるのが狙いです。

リポジトリ: https://github.com/devbasex/ai-plugins ／ 詳細は `docs/ndf-plugin-reference.md`