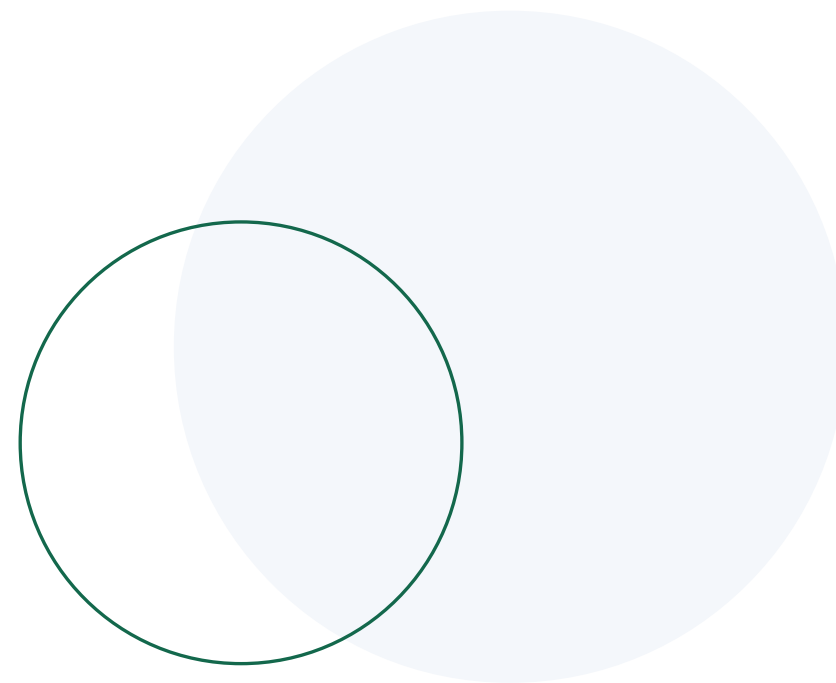


Grift CLI

経歴は「盛れる」。証拠は盛れない。

git履歴から「確かめられる事実」を取り出すコマンドラインツール



中身の手法: TEP (Technical Evidence Pipeline) / Cor.inc / pipx install grift-cli

「どんな仕事をしてきた人か」を、確かめる方法がなかった

エンジニア本人

- 経歴書に「Java 5年」と書いても、中身は伝わらない
- コミット数やGitHubの草(活動グラフ)は、AIで誰でも量産できる時代になった

紹介会社・採用側

- 経歴書と面談だけでは、本当のところが分からない
- 確かめる手段がないから、結局「年数」と「印象」で判断するしかない

共通する問題はひとつ — 自己申告を確かめる手段がない。

これまでのものさは、なぜ頼りにならないのか

ものさし	分かること	弱点
経験年数・人月	在籍していた長さ	「いた」ことしか分からない。中身は自己申告で、確かめられない
資格・学歴	知識があること	ふだんの仕事の進め方までは分からない
GitHubのスター・草	人気と活動量	AIと自動化で誰でも量産できる。「盛れる」
コーディングテスト	その場の瞬発力	数時間の試験。何年分の仕事ぶりは見えない

共通の弱点 — どれも「日々の仕事の実物」を見ていない。

Grift CLI の中身 — TEP（テップ） という公開手法

Grift CLI = 道具。手元で動くコマンド

TEP = 中身の考え方。git履歴から証拠を取り出す公開手法（定義も検証結果もすべて公開）

- エンジニアの仕事は git という記録に何年分も残っている。いつ・何を・どう変えたか
- TEPはこの記録から事実だけを取り出す。例: 「本番コードを直すとき、テストも一緒に直していた割合」

健康診断にたとえるとー

TEPが出すのは血圧や血糖値のような「検査値」。「健康偏差値78点」のような点数は出さない。

検査値を見て判断するのは人間（面接する人・提案する人・本人）。

TEPの仕事は、嘘のつけない検査値をそろえて渡すこと。

名前の由来 — TEP は何の略か

Technical（技術の）

対象は技術の仕事。コードと、その変更の記録

Evidence（証拠）

出すのは「証拠」。評価や判定ではない。名前の真ん中にこの語を置いたのが設計思想

Pipeline（流れ作業）

git履歴 → 区別 → 観測 → 検算できるレポート、という決まった手順で流す。途中で人の主観やAIの判断を挟まないから、誰が実行しても同じ結果になる

「評価のパイプライン」ではなく「証拠のパイプライン」。名前が、そのまま設計原則。

なぜ、ひとつの点数にまとめないのか

- 点数にした瞬間、点数を盛る競争が始まる（試験対策と同じことが起きる）
- 仕事はひとつの数字に潰せない。「テストを丁寧に書く人」と「障害対応が速い人」は、どちらが上でもない
- 点数を出さないから、盛る対象がそもそも無い。残るのは事実だけ

数字で、この人がどんな実装をしてきたかが分かる。

役割によって、大事になる数字は違う。

だから、数字をもとに人が評価することはあっても、数字だけで評価はしない。

技量は等級にしない。経験は観測できる。

具体的に何を測るのか — 「現場」と「仕事」の2段構え

1 まず、リポジトリそのもの（現場）を見る

- このリポは動いているか — 直近180日で実際に開発の手が動いた日数
- 何人で作っているか — botや重複アカウントを除いた実人数（例: 世界的OSSのclickでは467人）
- 特定の1人に頼りすぎているか — 最多貢献者の占有率（例: clickは28%）

2 次に、その中でのあなたの仕事を見る

- どの変更があなたのものか — 本人・他人・bot・自動生成・fork元などを11通りに区別してから数える
- 本番コードを直すとき、テストも一緒に直していたか
- 直した場所を、すぐまた直す羽目になっていないか（手戻り）

「量」は数える前に無効化する。区別が済むまで、数を数えることすら許さない設計。

測る項目① — あなたの仕事について

項目	何を測っているか
帰属の区別 (origin)	どの変更が誰のものか。本人・他人・bot・自動生成・fork元などを11通りに区別する。すべての数値の前提
テスト併修率 (test co-change)	本番コードを直すとき、同じ変更の中でテストも書き換えていた割合
手戻り率 (corrective rework)	全変更のうち、直近21日以内に触った場所への「直し直し」だった割合。少ないほど一発で決めている観測 (バグ数ではない)
磨き直し (path retouch)	一度作った場所に戻って、手を入れ続けている割合
定着率 (survival)	書いた行が、180日たっても残っている割合 (時間がかかるためオプション実行)

使っているテストの道具 (pytest等) の有無も記録される。どの項目にも「観測の限界」の但し書きが付く。

測る項目②－現場（リポジトリ）について

項目	何を測っているか
実働日数	直近180日のうち、実際に開発の手が動いた日数。ここから活発／維持／休眠を区別する
実人数	botや重複アカウントを除いた、実際に作っている人数
占有率	いちばん多く書いている人が全体に占める割合。1人依存の度合い
開発の進め方	PR（レビュー）経由の変更の割合、コミット規約の使用率
リポの育ち	リポの年齢、言語構成

これらをまとめて「文脈プロフィール」と呼ぶ。仕事の数字は、必ずこの文脈とセットで読む。

数値には、必ず「日本語の意味」がある

0.2664

(73/274)

世界的に使われているOSS pallets/click での実測

この人が本番コードに触れた変更は274回。うち73回は、同じ変更の中でテストも書き換えていた。およそ4回に1回。

読み方3点

- 数字には必ず分子と分母が付く（73/274）— だから受け取った側が検算できる
- 「4回に1回が多いのか？」は、似た文脈のリポの分布（参照分布 v2026.11）と並べて位置が分かる。
優劣の等級ではない
- 高い＝良い、ではない。ドキュメント中心の持ち場なら低くて当然。だから必ず文脈とセットで読む

テスト以外の項目もすべて同じ型 — 手戻り率も定着率も、分子・分母と日本語の意味つきで出る。

実際の出力 — report.md の抜粋（pallets/click・リポ全体スコープ）

```
① ## Shared block (copy-pasteable)
- tool: grift 0.5.1 (method TEP, definition tep-v0.5.1-2026-08-22)
- scope: repo (analyzed at commit `2c8cd3ac958a`)
- test co-change: 0.3178 ratio (444 of 1397 commits)
- 読み方: 本レポートは証拠であり判定ではない (docs/norms.md 参照)
  (中略)
## Test co-change
② - co-change 0.3178 (repo scope) — reference corpus v2026.11 (n=68) decile 9
## Rework
③ - corrective rework ...: 0.073 ratio (102 of 1397 commits; 21 day window)
## Survival (tau=180 days)
④ - not observed (survival_scan_disabled)
## Context profile (observational; not a ranking)
- collaboration: community (467 resolved human actors; top actor share 0.28 ratio)
- lifecycle: active (active_days_180d 90 days; days_since_last_human_commit 0;
repo age 4501 days)
⑤ ## この数値ではいけない判断
- 単独の数値での合否判断・他者との比較表の作成は TEP 非準拠です (docs/norms.md 参照)
```

① 先頭は「共有ブロック」— 経歴書やプロフィールにそのまま貼れる数行。道具・定義版・分析SHAつきで、受け取った側が検算できる

② decile = 似た文脈のリポ68本と並べたときの位置（10分割の帯）。優劣の等級ではない

③ 手戻り率 — 1397回の変更のうち102回は、直近21日以内に触った場所の直し直しだった

④ 測っていない項目は not observed と理由つきで明示。母数が20未満の割合はそもそも表示しない

⑤ 末尾には必ず「してはいけない判断」が付く

この方法は本当に効くのか — 117リポで確かめた

効いたこと

「検証文化のある本物のOSS」と「テンプレ量産リポ」を、この観測だけで見分けられるか試した。

→ 18リポずつを比べて、ほぼ完全に分かれた（統計的な分離度 $\delta=0.9198$ 。1.0が完全分離）。

→ 「量は誰でも作れるが、検証の痕跡は作り込めない」の実証。

まだ効かないこと

「AIを活用しつつ検証文化もあるリポ」との見分けは、まだ不完全（ $\delta=0.291$ ）。

→ 隠さずそのまま公開している。AIを使っても検証文化があるなら、それは本物の開発 — そもそも“見分ける”必要が変わりつつある、が正直な読み。

効かなかった結果も、基準を下げずに公開する。それがこの手法の信用の担保。

似た研究や道具はあるのか — 系譜と立ち位置

名前	何をしているか	TEPとの関係
DORA（DevOps研究）	組織のソフトウェア提供の力を、調査と統計で毎年公開する研究	対象はチーム・組織。TEPは「個人の経験の証拠」を扱う
SPACE（研究の枠組み）	「開発者の生産性は単一の指標では測れない」を整理した研究（2021）	「ひとつの点数にしない」というTEPの設計は、この研究側の合意と同じ側に立つ
GitClear（商用SaaS）	git履歴から変更の質を分析する商用サービス。行の価値の重み付けを行う	TEPは重み付けをしない。かわりに定義を公開し、手元で検算できる形を取る
リポジトリマイニング研究（MSR）	git履歴から開発の実態を調べる学術分野。本番コードとテストの共進化の研究（2008～）など	テスト併修率はこの系譜の観測。TEPはそれを個人が使える道具にした
bus factor 研究	「特定の1人がいなくなったら困るか」をgit履歴から推定する研究	実人数・占有率の観測はこの系譜

個々の観測には先行研究がある。「個人の経験を・点数にせず・手元で・検算できる形で出す」という組合せが、TEPの立ち位置。だから定義も検証結果も公開している。

使い方 — 5分で、自分のパソコンの中だけで

1 入れる: `pipx install grift-cli`

2 測る: `grift analyze . --scope repo`（リポ全体）／ `--scope tenant`（自分の仕事だけ）

3 読む: `report.md`（人間が読む用）＋ `report.json`（機械・検算用）

すべて自分のパソコンの中で完結。どこにも送信されない（送信機能そのものが実装されていない）。

「盛れない」を支える仕組み — 誰でも検算できる

- 受け取ったレポートは `grift verify report.json` の1コマンドで検算できる — 1箇所でも書き換えれば検出される
- 計算方法は全部公開している（定義・参照分布・うまくいかなかった指標まで）。隠し味がない
- 同じリポ・同じ版なら、誰が実行しても同じ数値になる（LLM不使用・毎回同じ結果）

だから「自己申告」ではなく「証拠」として扱える。

エンジニアのあなたは、こう使える

経歴書に1行添える —

「Java 5年」の隣に、検算できる事実ブロックを貼る。受け取った側は verify で確かめられる

面接で語る材料にする —

「テストは4回に1回、一緒に直しています。チームの方針で...」。数字が話の入口になる

自分の現場の健康診断にする —

1人依存になっていないか、テスト文化が痩せていないか。転職先リポの下見にも使える

紹介会社・採用側は、こう使える — エージェントの方へ

提案書に「確かめられる事実」を添えられる —

「経験5年」だけの提案から、「本番修正の4回に1回でテストも更新（本人実行・検算可能）」を添えた提案へ

面談の質問が具体的になる —

「このリポではテストとの併修が多いですね。どういう進め方でしたか?」。話が深くなる

前提のルール（これが信用の源）

- 数値は候補者本人が自分で実行して共有する — 同意なしに他人を測ることは利用規範で禁止
- 数値だけで落とす使い方は禁止（単独の足切り不可）
- 点数やランク付けはそもそも出ない — だから候補者との関係を壊さない

よくある質問（1/2）

Q. 結局、数字を盛ることはできるのでは？

A. 量は盛れます。だからTEPはまず、bot・自動生成・fork元を11通りに区別して「量」を無効化します。そのうえ残るのは「本番変更にテストが伴っていた」のような検証の痕跡で、これを何年分も作り込む手間は、実際にその仕事をするのとほぼ同じです。しかも数値は検算可能なので、作った主張は照合で崩れます。

Q. 数値が低い人は不利になりませんか？

A. 「低い＝悪い」という読み方を、仕様の側で禁止しています。持ち場（ドキュメント中心・レビュー中心など）によって数値の出方は変わるため、必ず文脈とセットで読む・母数不足は表示しない・レポートに「してはいけない判断」を明記する、の3点で守ります。数値だけの足切りは利用規範違反です。

Q. privateリポしかありません。使えますか？

A. 使えます。自分の手元で実行し、集計値だけを自分の意思で共有できます。コードやリポの中身を見せる必要はありません。

よくある質問（2/2）

Q. AIでコードを書いていたら不利になりますか？

A. なりません。TEPには減点という仕組みがそもそも無く、AIで書いたかどうかを検出して罰する機能也没有ありません。むしろ「AIを使いつつ、テストなどの検証を伴っている」という仕事の型が痕跡として残ります。検証の伴うAI活用は、これからの標準的な仕事の形です。

Q. 何かデータが送信されますか？

A. 一切ありません。送信機能そのものが実装されていません。共有するかどうか・何を共有するかは、すべて本人が決めます。

Q. 経験年数や人月の代わりにになりますか？

A. 置き換えではなく「裏付け」です。「5年」という申告に、「その5年に何の痕跡があるか」を添えるための道具です。年数の会話をやめる必要はありません — 確かめられるようにするだけです。

量は誰でも作れる時代。 確かめられる事実だけが残る。

技量スコアではない。技量は等級にしない。経験は観測できる。

- まず自分のリポで5分 — `pipx install grift-cli` → `grift analyze` .
- 結果の違和感を教えてほしい — GitHub Issues へ。「自分の感覚と違う」が定義を良くする最高の貢献

Grift CLI / github.com/Cor-Incorporated/grift-cli / PyPI: `grift-cli` / 利用規範: [docs/norms.md](#)