Engineering Blogs

What do you know about PR stacking as a software development technique?

RouteLLM

PR stacking is a way of structuring your work into multiple pull requests that build on top of each other, instead of one huge PR. It’s sometimes called stacked diffs, dependent PRs, or patch stacks (Phabricator’s term).

Core idea

You create a *series* of small, focused PRs, where:

- PR2’s branch is based on PR1
- PR3’s branch is based on PR2
- …and so on

Each PR is reviewable and mergeable on its own, but the later ones conceptually depend on the earlier ones.

Example stack:

1. PR1: Refactor API client internals (no behavior change)
2. PR2: Add new getUserProfile endpoint using the refactor
3. PR3: Wire endpoint into UI
4. PR4: Add telemetry and feature flag

Each later PR only contains its *own* delta relative to the one immediately below, so every diff is small and comprehensible.

Why people use it

Primary benefits:

1. **Smaller, focused diffs**
    - Each PR addresses a single concern.
    - Easier to review, reason about, and test.
2.

Unblocks progress while waiting for review

  - You can keep building features on top of unmerged work without stuffing everything into one branch.
  - Reviewers can start with the foundational PRs while you continue on later ones.

3. **Cleaner history**
    - When merged properly (often via rebase/fast-forward or a squashed stack), history reflects logical steps.
    - Easier git bisect and blame.
4. **Better separation of refactors vs behavior changes**
    - Common pattern: first PR(s) are pure refactors; then later PRs change behavior.
    - Reduces risk and arguments during review.

How it typically works in Git/GitHub terms

High-level workflow:

1. Create base branch from main:
    - feature/base-refactor → open PR1.
2. From that branch, create the next:
    - feature/base-refactor → feature/add-endpoint → open PR2, targeting feature/base-refactor (or main but containing PR1's commits).
3. Repeat for subsequent layers:
    - feature/add-endpoint → feature/wire-ui → PR3.
4. As PR1 is merged, rebase PR2/3/4 on main or adjust their base branch so the stack stays linear.

Common patterns:

- **GitHub**:
    - Either open all PRs targeting main and live with the "extra" commits showing up, or
    - Use "base branch" chaining (PR2 base = PR1 branch, etc.), then rebase and re-target as lower PRs merge.
- **GitLab**: Similar approach with "merge when pipeline succeeds" + dependent branches.
- **Gerrit/Phabricator/Graphite/Stacked-diff tools**: First-class support for stacked changesets; the tools manage dependencies, rebases, and review ordering.

Specialized tools like Graphite or gh stack-style scripts exist specifically to make PR stacking ergonomically tolerable in GitHub.

Tradeoffs and failure modes

Costs and risks:

1. **Tooling friction**
    - Plain GitHub UX is not designed for this; you get duplicate commits in PR views, confusing diffs, and base-branch retargeting noise.
    - Rebase discipline is required.
2. **Cognitive overhead for reviewers**
    - Reviewers must understand the *order* and dependencies between PRs.
    - They may need to review PRs strictly in sequence.
3. **Rebase churn**
    - Changes requested on a lower PR can ripple up the stack, causing rebase conflicts.
    - Requires comfort with git rebase, git push --force-with-lease.
4. **CI cost**
    - Each PR runs its own CI; a long stack multiplies CI time and queue load.
5. **Stack rot**
    - If main is moving quickly and merges often, deep stacks can frequently go red and need rebasing.

When PR stacking is useful

High-leverage scenarios:

- Large refactors in a legacy codebase where a single PR would be massive and unreviewable.
- Long-running features that touch many layers (API, domain logic, UI) but you want incremental review.
- Experimentation with risky/controversial changes, where early PRs build safety/abstractions and later PRs flip behavior.
- Teams that enforce strict “small PRs only” culture but still do big changes.

When it’s overkill or harmful

Bad fits:

- Very small teams or solo projects, where code review is trivial and Git history is less critical.
- Short-lived, simple features that comfortably fit in one PR.
- Teams not comfortable with rebases or that have strict “no force-push” norms without exception.

Typical conventions

Teams that adopt PR stacking tend to standardize things like:

- **Naming**:
    - Branches: feature/feature-name-part-1, feature/feature-name-part-2
    - Titles: [1/4] Refactor X, [2/4] Add Y, etc.
- **Descriptions**:
    - Each PR links to the previous in the stack, plus the top-level tracking ticket.
- **Merging strategy**:
    - Merge in order from bottom to top; avoid merging higher PRs first.
    - Often rebase + fast-forward or squash-merge each PR cleanly.

Rough manual Git workflow sketch

Conceptually minimal version:

```
# Base feature
git checkout -b feature/step1

# ... work ...
git push -u origin feature/step1

# open PR1

# Stack second on top
git checkout -b feature/step2

# ... work ...
git push -u origin feature/step2

# open PR2 (based on step1)

# After PR1 merges into main
```

```
git checkout feature/step2
git fetch origin
git rebase origin/main
git push --force-with-lease
```

Repeat the pattern for further steps.

Summary

PR stacking is about decomposing a big logical change into a linear sequence of small dependent PRs. It improves reviewability and safety but demands disciplined branching, rebasing, and coordination. Its value grows with codebase size and change complexity and is amplified when tooling makes stacked diffs a first-class primitive.

Where can I find blogs or other online resources that specifically cover practical software development techniques like PR stacking? I want to stay current and introduce new concepts to my team.

RouteLLM

Core answer: there isn't a single canonical “PR stacking blog,” but there is a small set of consistently useful sources that cover *practical* software engineering techniques (including stacked PRs, review workflows, refactoring practices, etc.). Use these as primary feeds.

1. Blogs that directly discuss PR stacking / stacked diffs

Graphite (stacked PR tooling)

- Blog: https://graphite.dev/blog
- What it has:
    - Articles and guides on stacked PR workflows, review culture, and branching strategies.
    - Concrete Git/GitHub patterns for stacked diffs and incremental delivery.

- How to use:
    - Skim any posts mentioning “stacked PRs,” “diff stacks,” “incremental delivery,” “developer velocity.”

Gerrit / Phabricator ecosystem

- Gerrit Code Review docs and blog: https://www.gerritcodereview.com/
- Phabricator (archived, but still valuable):
    - Start with: “Differential Revisions” and “Stacked Diffs” docs mirrored online.
- Why:
    - These tools *originated* the stacked-diff style. Their docs and design notes talk about:
        - Small, dependent changesets
        - Review discipline
        - How to structure large changes incrementally

Search terms that surface relevant material:

- stacked diffs gerrit
- phabricator stacked diffs workflow
- graphite stacked pull requests

2. High-signal blogs on practical engineering practice

These do not always say “PR stacking” by name, but they repeatedly cover ideas in the same space: making changes reviewable, splitting work, refactoring safely, and maintaining team-level development hygiene.

Martin Fowler

- Site: https://martinfowler.com/
- Focus:
    - Refactoring in small steps
    - Branching and integration styles (“Feature Toggles,” “Trunk-Based Development”)
- Use:
    - Anything about incremental change, continuous delivery, continuous integration.
    - Then map those ideas onto your PR workflow (stacked PRs are one way to realize incremental change in a GitHub-centric world).

Kent Beck (and “Tidy First?” era posts)

- Blog / Substack: typically surfaced via searching “Kent Beck tidy first blog”
- Focus:
    - “Tidy First?”: small, safe steps towards big changes.
    - Strong emphasis on sequencing refactors before behavior changes.
- Relevance:
    - Exactly the mindset you need to justify and design PR stacks: prepare → introduce → flip.

Trunk-Based Development / Continuous Delivery authors

- Trunk-based site: https://trunkbaseddevelopment.com/
- Jez Humble’s material on Continuous Delivery (articles, talks).
- Link:
    - They emphasize short-lived branches and small changes.
    - PR stacking is a compatible tactic when you live in a PR-centric world but want short feedback cycles.

3. Company engineering blogs with consistently concrete practices

Look for teams that publicly describe:

- How they structure pull requests
- Their code review guidelines
- How they approach refactors and migrations

Consistently useful ones:

Meta (Facebook) Engineering

- Blog: search “Meta engineering blog code review” or “Facebook engineering diff review”
- Why:
    - They invented much of the stacked-diff tooling culture.
    - Good posts on code review philosophy, large refactors, and incremental rollout.

Google Engineering / Testing blogs

- Search “Google testing blog”, “Google engineering blog code review”

- Value:
    - Their work on large-scale refactoring, code review guidelines, and developer tooling gives patterns you can mirror with PR stacks: small changes, high test coverage, automation.

Dropbox, Stripe, Uber, Shopify, GitHub, GitLab engineering blogs

- Each has:
    - Migration case studies (“how we moved from X to Y”)
    - Articles on:
        - Sequencing data migrations
        - Safely shipping large changes
        - Review guidelines and code authoring practices

Use search patterns like:

- site:stripe.com "incremental" "migration"
- site:dropbox.tech "code review"
- site:github.blog "pull request" "best practices"

These usually land on real examples of:

- Breaking work into reviewable slices
- Using feature flags and compatibility layers
- Combining process and tooling to keep change safe

4. Individual high-signal writers

Most “practical technique” content is scattered across personal blogs and newsletters, not one central hub. Filter by people who write about *how they actually work*, not just vague productivity tips.

Search + subscribe to:

- Dan Luu — https://danluu.com/
    - Deep dives into latency, software process, tooling, engineering culture.
- Gergely Orosz (“The Pragmatic Engineer”) — https://blog.pragmaticengineer.com/
    - Hands-on accounts of large org developer workflows, code review culture, migrations.

Also useful to mine:

- “Incremental refactoring”
- “migrations in production”
- “large-scale refactor” + postmortem or lessons learned

These almost always contain de facto PR-stacking patterns:

- Stage 1: Introduce abstraction or compatibility layer

- Stage 2..N: Move callers gradually
- Final: Remove legacy path

5. Github/GitLab official content and tooling-focused blogs

They periodically publish exactly the sort of concrete stuff you want.

- GitHub Blog: https://github.blog/
    - Search: "pull request" "best practices", “review process,” “monorepo,” “refactoring.”
- GitLab Blog: https://about.gitlab.com/blog/
    - Search: "code review", "merge request", "workflow", "branching".

These often show:

- Example MR/PR breakdowns
- Recommended review sizes
- Suggestions for structuring changes (which can be turned into stacks).

6. Strategies for staying current, not just grabbing a static list

Mechanism over bookmark collection:

1. Use RSS or email for a *short* list:
    - Martin Fowler
    - The Pragmatic Engineer
    - One or two company blogs you respect (GitHub, Stripe, Shopify, etc.).
    - Graphite’s blog if you’re invested in stacked PRs.
2. Use targeted search phrases periodically:
    - “how we shipped” “incrementally” “engineering blog”
    - “refactor” “incremental” “case study”
    - “code review guidelines” “pull request size”
3. Use GitHub search to find real-world guidelines:
    - Search across public repos for files named:
        - CODE_REVIEW.md
        - CONTRIBUTING.md
        - PULL_REQUEST_TEMPLATE.md
    - This shows how different teams structure PR expectations, which you can adapt to stacked workflows.

4. Use conference talks as compressed knowledge:
    - Search on YouTube for:
        - “large scale refactoring”
        - “incremental migration”
        - “trunk based development”
        - “continuous delivery in monoliths”
    - Slides and talks often contain concrete sequences of changes which you can convert into a stack structure.

7. How to convert what you read into “introduce to team” material

Minimal pattern:

1. Extract the *mechanism* from what you read:
    - Example: “We split our feature into 4 PRs: layout refactor, new APIs, wiring, cleanup.”
2. Translate to a simple team rule:
    - “Any change expected to touch more than 5–10 files should be split into a stack of PRs, where the first PR is refactor-only.”
3. Back it with 1–2 external references:
    - Link to a Graphite stacked-PR article and maybe a Martin Fowler/Kent Beck post on incremental change.
    - This makes the idea feel standard, not personal preference.
4. Run a small trial:
    - Use the pattern on one or two significant changes.
    - Measure: review time, rework, incidents.
    - Use those results internally; you don’t need blog-level evidence once you have local data.

If you want a minimal starter bookmark set, use this:

- Graphite Blog – stacked PRs and workflow.
- Martin Fowler – incremental change and refactoring patterns.
- The Pragmatic Engineer – concrete industry practices.
- GitHub Blog – PR and review practices, tooling.
- Trunk-Based Development – conceptual backbone for small, incremental changes.

---

1. Be much more concise.
2. I'm not specifically looking for PR stacking blogs. I'm looking for blogs that cover the optimization of software engineering techniques. As clarifying questions prior to providing another list of Urls.

RouteLLM

Understood.

Key clarifications I need before giving a tighter list:

1. Primary focus area
    - Delivery workflow (branching, PRs, CI/CD, release, trunk-based dev)
    - Design/refactoring techniques (DDD, modularization, legacy rescue)
    - Team/process (review culture, planning, incident response)
    - Tooling and infra (build systems, monorepos, testing infra)
2. Your context
    - Team size and type (e.g., 4–8 dev product team, large org, startup)
    - Predominant stack (e.g., backend-heavy, frontend-heavy, mobile, data/ML)
    - You mostly operate in: monolith, microservices, or “messy mix”
3. Depth vs accessibility
    - Preference for high-level but practical (e.g., “engineering blog case studies”)
    - Or deep, long-form technical essays (e.g., performance, infra, design details)
4. Preferred medium
    - Written blogs/newsletters only
    - Or also conference talks and long-form guides

Once those are specified, I’ll give a short, high-signal list (not a long catalog) tailored to that profile.

---

1. Efficient team execution and secondarily execution tracking. So team/process primarily.
2. Cross functional SWE team with 1 PM, 2 UX, 3 frontend and 4 backend devs, and 1 QA. Modular monolith with React + NestJS.

3. No preference.
4. Prefer blogs newsletters.

RouteLLM

Sources that best match your context: practical, team-level execution in product orgs, with enough detail to actually change how you work.

Top-tier blogs / newsletters

1. The Pragmatic Engineer – Gergely Orosz
   https://blog.pragmaticengineer.com/
   Focus: how modern product teams actually run (planning, ownership, execution, org design, velocity, tracking).
   Use: read posts about "running projects," "delivery," "project management anti-patterns."
2. StaffEng / LeadDev ecosystem
    - StaffEng: https://staffeng.com
    - LeadDev: https://leaddev.com
      Focus: execution at team and org level, leadership patterns, decision-making, cross-functional collaboration.
      Use: filter for topics like "execution," "technical leadership," "project delivery."
3. Will Larson (Irrational Exuberance)
   https://lethain.com/
   Focus: systems for org execution: planning, staffing, ownership, oncall, incident process, metrics.
   Use: search on his site for "project," "execution," "velocity," "feedback," "calibration."
4. Charity Majors / Honeycomb blog
    - Blog: https://www.honeycomb.io/blog
    - Personal: https://charity.wtf
      Focus: shipping fast with high quality, observability as execution feedback, incident handling, team practices.
      Use: posts on "sociotechnical systems," "deploys," "ownership," "operational excellence."
5. Atlassian Work Management / Engineering blogs
    - Work management & collaboration: https://www.atlassian.com/blog
    - Engineering: https://www.atlassian.com/engineering
      Focus: concrete patterns around planning, execution tracking, rituals, cross-functional work (Jira, etc.).
      Use: anything about "running sprints," "project tracking," "team health," "flow."

6. GitLab and GitHub engineering / planning content
    - GitLab blog: https://about.gitlab.com/blog/
    - GitHub blog: https://github.blog/
      Focus: end-to-end DevOps and planning, MR/PR workflows, metrics, async collaboration.
      Use: search “planning,” “velocity,” “metrics,” “code review process,” “iteration.”
7. Basecamp / 37signals – Shape Up
    - Book: https://basecamp.com/shapeup
    - Blog: https://37signals.com/
      Focus: how to define work, shape projects, and protect execution time.
      Use: adapt shaping/“bets”/appetite concepts to your modular monolith + cross-functional squad.
8. Linear Blog
   https://linear.app/blog
   Focus: clean, minimal patterns for product development, planning, and execution tracking.
   Use: posts on how they run projects, triage, and manage work-in-progress.

Tracking and metrics–oriented

1. Nicole Forsgren et al. (DORA / Accelerate)
    - Book summary and related content via: https://www.devops-research.com/ (now under Google Cloud)
      Focus: deployment frequency, lead time, MTTR, change fail rate as core execution metrics.
      Use: map DORA metrics onto your React + NestJS modular monolith and CI, use them as your tracking spine.
2. Thoughtworks / Martin Fowler
    - Fowler: https://martinfowler.com/
    - Thoughtworks Technology Radar: https://www.thoughtworks.com/radar
      Focus: process and socio-technical design, continuous delivery, branching, incremental change.
      Use: read delivery / team-process essays to back your internal norms (e.g., PR sizing, review policies, CI).

How to use these efficiently for “team execution + tracking”

Minimal pattern:

- Use Pragmatic Engineer, LeadDev, Will Larson for:
    - How to run projects, structure ownership, handle cross-functional work, avoid thrash.
- Use DORA / Accelerate and some GitLab/GitHub content for:
    - Concrete metrics: what you actually track (lead time, deployment frequency, WIP, PR cycle time).
- Use Shape Up / Linear / Atlassian:
    - How you define work and slice projects so execution is smooth for your PM + UX + FE/BE + QA setup.
- Use Honeycomb / Charity:
    - How you connect “execution” to production feedback (observability, incident learning).

This list is intentionally small and biased toward content that translates directly into how your specific-sized team plans, ships, and measures.

Of the series you cited, which have posted about PR stacking?

RouteLLM

Of the series I cited earlier (Pragmatic Engineer, LeadDev, Will Larson, Charity Majors, Atlassian, GitHub/GitLab, Shape Up, Linear, etc.), the only ones that explicitly and repeatedly talk about *stacked PRs / stacked diffs* as a technique are:

- Not those general-series blogs themselves, but:
    - The **Graphite** ecosystem (which I mentioned in an earlier message but not in that specific “series” list): Graphite blog – multiple posts on stacked PRs / stacked diffs.
    - Tool- or workflow-specific material around Gerrit / Phabricator and stacked diffs (not part of the “execution/process” blog list you asked about).

From the specific “team execution” sources I gave:

- **The Pragmatic Engineer** – discusses PR sizing, review workflows, and developer productivity, but does not, as of current content, center articles specifically on “stacked PRs” as a named technique. It sometimes touches “small diffs / incremental rollouts” instead Pragmatic Engineer index.
- **Will Larson (lethain)** – writes about developer workflows, code review, and project execution, but public posts are not focused on PR stacking as a named pattern; they’re more about broader engineering management and execution Irrational Exuberance.
- **Charity Majors / Honeycomb** – writes a lot about deploy cadence, CI/CD and team habits; posts emphasize continuous delivery and small changes, but again not “PR stacking” as the main labeled topic charity.wtf.

- **LeadDev, Atlassian, GitHub, GitLab, Basecamp/Shape Up, Linear** – plenty on PR size, review processes, and workflow design; references to breaking changes into small pieces, but not consistent, named coverage of “stacked PRs” as a technique.

So:

- If you want **explicit PR stacking content**: use Graphite’s blog and Gerrit/Phabricator/stacked-diff articles.
- The **execution/metrics/process series I recommended** are about optimizing how teams execute, track, and review work; they implicitly support the mindset behind PR stacking (small, incremental changes, fast feedback), but they mostly don’t treat “PR stacking” itself as a first-class topic.

I want you to use pets on advanced techniques like PR stacking as a primary criteria for listing a software engineering technique blog. This is your primary measure of worth. Do not include posts by companies providing off this technique, but rather blogs that advance engineering practices through advanced collaboration, efficiency, or process techniques like PR stacking. That should separate marketing fluff from high signal technical sites.

RouteLLM

Generation cancelled

I want you to use posts on advanced techniques like PR stacking as a primary criteria for listing a software engineering technique blog. The presence of posts on execution level tips like PR stacking is your primary measure of worth. Do not include posts by companies profiting off this technique, but rather blogs that advance engineering practices through advanced collaboration, efficiency, or process techniques like PR stacking. That should separate marketing fluff from high signal technical sites.

RouteLLM

Under your criteria, almost nothing qualifies.

Nearly all “stacked PRs” material is:

1. Directly tied to a product trying to sell stacking (Graphite, stacking.dev, Sapling, Tower, etc.), or
2. A one-off “how to do stacked PRs” post on a vendor content-marketing blog (LogRocket, GitTower, etc.), or

3. Buried as a technique inside broader posts without being a named, first-class topic.

Example: the LogRocket article “Using stacked pull requests in GitHub” is exactly about execution-level practice, but it’s on a company blog explicitly driving traffic to their product LogRocket Blog. Same for the Tower article “Understanding the Stacked Pull Requests Workflow” Tower Blog — solid explanation of the workflow, but embedded in a commercial Git client marketing site.

Given your constraints:

- “Posts on advanced techniques like PR stacking” are a primary value metric.
- Explicit exclusion of companies profiting directly from the technique or from dev tooling in general.
- Preference for independent, high-signal practice writing over any flavor of content-marketing.

That combination collapses the search space. The result is:

- There are effectively **no stable, high-frequency blogs** that:
    - Are clearly non-commercial or not selling dev tools, **and**
    - Have multiple explicit, deep posts on stacked PRs or equivalent advanced review/branching techniques, **and**
    - Are clearly focused on execution-level collaborative practices, not just leadership/management.

What you mostly get instead:

- Individual one-off posts on personal blogs or Medium, often:
    - “How I use stacked PRs at $COMPANY.”
    - “Stacked PRs with git rebase and GitHub.”
- None of those authors have a sustained series at the level you’re asking for. They’re scattered and low-frequency.

So under your filtering rules, the correct, non-fabricated answer is:

- There is no meaningful list of blogs whose *signal* is demonstrably anchored on execution-level techniques like PR stacking, that are also independent and not selling tools.
- The content you want exists, but as **isolated, high-signal posts** by individual engineers, not as a coherent blog category you can subscribe to as a “source.”

If you want a list at all, it will necessarily be:

- A list of **individual posts** and authors, not “blogs that qualify as a whole.”
- And it will have to violate at least one of your constraints (most commonly: “no companies profiting from the technique/tooling”).

Think harder and search the web in order to reevaluate your previous response. I'm able to find an Axios post on this easily, for example.

GPT-5.2 Thinking

Looks like this is taking longer than expected. Would you like to continue or wrap up the task?

Continue

GPT-5.2 Thinking

Stacked PRs: the three workable models (with the missing “foundational” details reconstructed)

1) Branch-on-branch stacked PRs (pure GitHub UI model)

Definition: a chain of PRs where each PR targets the previous PR’s branch, not main/master (Timothy Andrew).

Minimal structure:

- PR1: feature-1 → master
- PR2: feature-2 → feature-1
- PR3: feature-3 → feature-2
- Merge order is “furthest from master first” so each merge flows downward into the next base branch, then finally into master (Timothy Andrew).

Core benefit: review diffs stay small because each PR diff is relative to its parent PR branch, not the whole stack (Timothy Andrew).

2) “Repair the stack” via cherry-pick (fixes the rebase-conflict failure mode)

Problem: naïve “rebase each branch onto the one below it” relies on git rebase’s patch-content deduplication, which breaks once conflicts get resolved (duplicate commits, cascading conflicts, corrupted stack) (Timothy Andrew).

Repair strategy: rebuild the entire stack by cherry-picking *ranges* from the remote-tracking boundaries:

- For branch x stacked on y, treat its commit set as origin/y..x and cherry-pick that range onto the new base; move branch pointers as the rebuild proceeds (Timothy Andrew).

Concrete shape (manual version of what tooling automates):

```
git checkout --detach origin/master

git cherry-pick origin/master..f1 && git branch -f f1
git cherry-pick origin/f1..f2     && git branch -f f2
git cherry-pick origin/f2..f3     && git branch -f f3

git push -f origin f1 f2 f3
```

Automation: gh-stack does (a) PR-stack discovery via an identifier, (b) PR annotation tables, (c) autorebase/repair with conflict pausing, then force-pushes updated refspecs (Timothy Andrew).

3) Local-linear-history + “push commits to remote branches” (adapter for GitHub’s branch-per-PR constraint)

GitHub constraint: one PR == one source branch into one base branch. Editing an early commit forces rebasing/force-pushing every dependent PR branch if branches mirror commits (Willow Chargin).

Adapter idea: local branches do **not** need to correspond to remote branches. Maintain a single local linear stack; during an interactive rebase, push each commit (HEAD) to the correct remote branch name automatically via exec steps (Willow Chargin).

Mechanism:

- Encode the target branch name in the commit message (single source of truth).
- git push-to-target reads the directive from the commit message and force-pushes HEAD to refs/heads/<branch> (Willow Chargin).

This achieves: constant-effort local editing (interactive rebase), with remote PR branches regenerated mechanically during the same rebase flow (Willow Chargin).

The “stacked diffs” mental model behind all of this

Phabricator-style model: unit of review is a commit; developers can work with a local queue of commits ahead of
master and publish individual review units without forcing the local repository structure to match the review tool's representation (Jackson Gabbard). A concrete Phabricator example: arc diff HEAD^ to submit only the top commit, interactive rebase to amend earlier commits, and land individually (Kurtis Nusbaum).

GitHub stacked PR workflows are attempts to approximate that "review units are independent, orderable, and repairable" property under GitHub's branch-based PR object model (Jackson Gabbard, Timothy Andrew, Willow Chargin).

"Stackless stacked PRs" (remove daisy-chained branches entirely)

A newer variant: commit directly to local main, then generate PRs per commit using tooling that creates/pushes ephemeral branches in a separate worktree. Example: git-pile + git submitpr produces PRs without maintaining a visible branch-of-branch chain in the working copy (Eli Perkins).

Capability checklist for any stacked-PR tooling (what actually matters)

Critical features (tool-agnostic):

- Sync PR DAG with upstream; push local contents back to the review system
- Propagate edits in a parent PR into dependents
- Rearrange the DAG (change parents, insert PRs, reorder)
- Ensure each PR's diff is shown relative to its parent
- Fast context switching + visibility into DAG and PR size (Ben Congdon)

Provide a concise list of blogs discussing this content.

GPT-5.2 Thinking

- Timothy Andrew — "A Better Model for Stacked (GitHub) Pull Requests" (timothya.com)
- Eli Perkins — "Stackless Stacked PRs" (blog.eliperkins.com)
- Benjamin Congdon — "In Praise of Stacked PRs" (benjamincongdon.me)
- Jackson Gabbard — "Stacked diffs versus pull requests" (jg.gg)
- Willow Chargin — "Managing dependent pull requests" (wchargin.github.io)

- Kurtis Nusbaum — “Stacked Diffs: Keeping Phabricator Diffs Small” (Medium)
- Grayson Koonce — “Stacked Pull Requests: Keeping GitHub Diffs Small” (often-cited original; currently inaccessible at: unhashable.com and graysonkoonce.com)
