RFC 92 ABANDONED: Add ability to “stack diffs” to Automation

Editor: thorsten@sourcegraph.com
Status: **ABANDONED** - We moved the diff generation to src CLI in PR #8008 and thus a lot of the things in this PR are invalid.
Required approvals: Christina, Quinn, Tomás, Ryan Slade, Felix, Erik
Approvals:

Background

In the current state of Automation a user creates a campaign by first specifying input parameters, then previewing the changes that campaign would make and, finally, creating the campaign.

In order for users to preview changes, we have to compute them, that is: we need to generate diffs for each repository that is affected by the campaign (i.e. returned by the “scopeQuery” of the input parameters).

Each campaign type returns a different diff for a given repository. A Comby campaign returns a diff in which a pattern is replaced with another, a credentials campaign removes a credential, and so on.

Problem

The current model only allows for a single diff per repository. That means it’s not possible to, for example, run Comby twice over all repositories in a single Campaign, first replacing “A” with “B” and then replacing “C” with “D”, and finally combining the two produced diffs into a single one.

The result is that if a user wants to make N small search/replacements over a set of repositories, the user needs to create N campaigns.

Besides having to create multiple campaigns, there’s other downsides:

- more campaigns with small changesets lead to a lot of noise on the codehosts, due to a lot of pull requests being created
- there’s no guarantee anymore that the changesets are merged in the order the author of the campaigns intended

But most importantly: it’s not possible to **build diffs on top of each other** (stacking diffs). Example: it’s currently not possible to replace “foobar” with “barfoo” in the first run and then replace every instance of “barfoobarfoo” with something else.

That also limits what we could do with other campaign types in the future. It’s easy to imagine that a single campaign type would, for example, update a dependency, show the diff to the user, and then allow the user to update call sites in the code that are now deprecated with the updated dependency. With our current model it wouldn’t be possible to do that in two steps and allow user interaction in between. We’d have to do everything in one step.

Proposal

We make it possible for users to run campaign types multiple types over repositories, each run building on top of the diff generated by the previous run. When the campaign is then created, we create a single changeset out of the diff that's been accumulated for a repository.

In terms of git: the changeset preview is the staging area and a user can "git add" more changes to the staging area before turning them into a commit (creating the campaign)

The user flow would look similar to this:

- User defines campaign parameters
- User clicks "Preview changes"
- Diffs are computed
- Next to "Create" (which would create the campaign) another button appears: "Stage changes and add more"
- The input form for the campaign parameters is editable again
- User clicks "Preview changes"
- Diffs are computed with the previously computed diffs as their base

Open questions (need to be answered before implementation)

Question: Do we squash diffs when stacking them or do we keep them separate?
In other words: when the user decides to add another set of diffs on top of the already existing diffs, do we keep them separate and display them separately and then create separate commits when creating changesets, or do we squash the diffs into one CampaignPlan as soon as the user accepts the second CampaignPlan and then only create a single commit?

Why is that important? Because the first option — keeping diffs separate and creating separate commits — has a lot of implications for the rest of the process and is a potential source for unwanted complexity.

The biggest problem is **updating Campaigns.** If we do keep CampaignPlans separate and the user stacks 5 CampaignPlans on top of each other, what happens when the user then updates the Campaign? Can they only update the arguments of the topmost CampaignPlan? That would be a step down from the current behavior where the user can update every parameter that influences the way the diff is produced. If they can also update other CampaignPlans, we effectively need to *rebase* the other CampaignPlans on top, which is non-trivial.

What happens when the user decides to update the Campaign and remove one of the CampaignPlans and by doing that a diff for a repository is removed — do we close the changeset on the codehost? What if the user then *stacks another diff on it*? Do we then reopen it? (Since the branch name stays the same, we'd effectively have to)

In our UI we would also have to account for displaying multiple CampaignPlans, including their differing arguments. Most importantly, though, we need to build the UI in a way that allows the user to see which CampaignPlan produced which part of the diff. We would need something like a “commit list” and then allow the user to view the diff at each commit. To make that easy to understand is hard, especially since there are no existing UIs that we can use as inspiration. Every UI works on a per-repository basis, but we’d need to display a list of diffs per repository for *multiple repositories*.

Answer: ???

Question: When adding another set of diffs, which CampaignPlan arguments can the user edit?
Can you stack a “NPM credentials” CampaignPlan on top of a “Comby search & replace” CampaignPlan? Can you change the “scopeQuery”? That would have big implications for updating campaigns, since we’d need to update/close/create changesets on codehosts depending on which part of a diff was updated.

Technical notes on implementation

TODO(thorsten): I’ll fill this out in the near future

When the user decides to stage more changes, we “persist” the diff we already have as a commit on gitserver and use that as the base revision when computing another set of diffs.

Open question: how do we ensure that gitserver won’t clean up the not-yet-pushed commits?

Each CampaignPlan that is stacked on top of another then “inherits” the diffs per repository.