The history of change-packing tools at Microsoft (so far)

devblogs.microsoft.com/oldnewthing/20180122-00

January 22, 2018

Raymond Chen

Self-proclaimed "Author of unremarkable code" Jay Bazuzi posted a brief history of change-packing tools at Microsoft.

First of all, what is change-packing?

After you make some changes to code, you may want to save those changes without checking them in. Some source control systems have native support for this operation, variously called *shelving* or *stashing*. But first lets look at the requirements of change-packing.

First, a packed change needs to take the form of a single file. This allows the file to be added as an attachment to a bug, or saved away in a "Things I abandoned, but which I still want to keep around for reference" directory somewhere.

For example, you might make a change and then realize that it doesn't solve the problem. On the other hand, there was some good stuff in that change, like a new technique you discovered, and you don't want to lose the record of that change, in case you need to use that technique later. Keeping it in a single file makes it easier to manage. (A variant of this is where the bug you're trying to fix was rejected by Ship Room, so you want to save the fix away so you can bring it back at a more propitious time.)

Other use-cases for change packing are to get another developer's opinion on code you've written, either informally as a sanity check before getting in too deep, or formally as a pre-checkin code review. Or you can send the packaged change for pre-checkin validation: In the old days, you would send the changes to your buddy to verify that the code compiles, known as a *buddy build*. (The most common reason it wouldn't compile is that you forgot to include all the affected files in the package.) Your buddy would unpack the change into their local repository and try to build it. In somewhat newer days, you would send the changes to a dedicated change verification system.

Second, the contents of a packed change must be viewable on a non-developer machine. This means that you cannot assume that there is a local repository, but you are allowed to assume network connectivity. This requirement permits the packed change that you attached to a bug

to be opened in Ship Room so everybody can view the change and review it. (For bonus points, your packed change could remove the requirement that it have network access, so that it is usable even when offline.)

In the early days, Microsoft used a homemade source control system formally called Source Library Manager, but which was generally abbreviated SLM, and pronounced *slime*. It was a simple system that did not support branching. Files were centrally stored, and clients cooperated in updating the files.[1] The Windows team didn't have a change-packing tool for SLM. You typically sent around changes by just copying them to a shared location. If you wanted a code review, you brought someone to your office, and you reviewed the code in person.

Shortly after Windows 2000 shipped, the Windows source code transitioned to a source control system known as Source Depot, which was an authorized fork of Perforce. Microsoft made changes to the code base to do things like improve scalability and add new features.

It was during this era that I wrote a change-packing tool for Source Depot, which I named `bbpack` because its primary purpose was to pack changes for the purpose of a buddy build. (Its secondary purpose was to facilitate code reviews.) The package itself was a batch file that did the necessary work of applying the changes to the source code on the machine that ran the script, or if running in code review mode, to unpack the files into a temporary directory and then run a diff tool. I chose a batch file because I didn't want to solve the problem by creating a bigger problem: If I wanted somebody to run a buddy build for me, I wanted it to be as simple as possible: "Run this batch file, and then build this directory." Not "First, install this tool…"

Since the two purposes for the `bbpack` tool cared only about the difference in the package, I didn't pay attention to replaying the pending operations with full fidelity. As long as the source code after unpacking matched the source code that was packed, that was good enough. That means that if the original package was created via an integration (which is the Source Depot name for what everybody else calls a merge), the unpacked file was reported as an edit, not a merge. Similarly for other exotic file states like "undo".

Meanwhile, the Office team were also in the process of transitioning from SLM to Source Depot, and when they learned about this `bbpack` thing, Office tools magician[2] Stephan Mueller exchanged email with me to learn about `bbpack`, its design, its strengths and weaknesses, and based on this and his own powerful brain wrote a comparable change-packing tool for SLM, which he called `slmpack`. And then when Office completed their transition to Source Depot, he wrote it again, called `sdpack`. The Office team used these packages for the same thing that Windows did: Sharing changes between developers and viewing proposed changes in the Ship Room meeting.

Okay, here's where Jay Bazuzi enters the story.

> While we're talking awesome tools, here's ff.exe from @JohnWintellect: https://t.co/8kzy7yQj17 – like dir /s, but faster and more awesome! I use it many times daily.
>
> — Kirill Osenkov (@KirillOsenkov) October 26, 2017

> How do I get jjpack mentioned as an awesome tool. But, you know, organically?
>
> — Jay Bazuzi (@jaybazuzi) October 26, 2017

> remind me what it was? I have vague memories of bbpack, jjpack and other xxpacks. Did you write jjpack? What's the history there?
>
> — Kirill Osenkov (@KirillOsenkov) October 26, 2017

> bbpack was Raymond Chen. Used to send your SLM changes to another computer for a buddy build.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> When Source Depot came along, he ported it over, but it only supported add/edit/delete, and not integrations.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> I was running the team gated build (before "CI" was a thing!). We used bbpack to send changes to the system, and needed integration support.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> I started a thread where the 2 SD devs debated for a week about how integrations worked, and wrote down what they said in JJPack.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> It spread across the company, except for Windows and Office (the latter had SDPack). Surprisingly popular for a side project.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> In my first month at Tableau I was getting to know my coworkers and awkwardly injected jjpack in the convo. They both knew it.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> Today I use Perforce and it's built-in shelve feature is not as good as JJPack.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> The name is a vain riff on bbpack. The other vanity name was JRC, Jay's Resource Compiler. Only ever used in the VS Debugger.
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> Fixes a bunch of the headaches with merging resource string changes.
>
> Pronounced "jerk".
>
> — Jay Bazuzi (@jaybazuzi) October 27, 2017

> love this history lesson! I remember JRC too, didn't know what it stood for!
>
> — Kirill Osenkov (@KirillOsenkov) October 27, 2017

One sign that you wrote a useful tool is that people started using it for things that you didn't originally consider. Jay was using my `bbpack` tool to submit jobs into a gated build system, which was totally not the intended use. More than once, I received reports from people saying, "We're using your `bbpack` tool to transport changes between machines, and we find that it doesn't work reliably when there are more than 50,000 files in the package," or some other ridiculous thing. Dude, that is nowhere near what `bbpack` is for. No developer is going to change 50,000 files and send it out for code review.

There was no rivalry among the three of us over whose change-packing tool was best. Indeed, I considered `jjpack` and `sdpack` to be "next generation" packing tools and did what I could to deprecate `bbpack` and steer people toward `jjpack` and `sdpack`.

What probably settled the not-really-a-battle between `jjpack` and `sdpack` was when another team wrote a code review tool that became wildly popular all over Microsoft. That tool used `sdpack` files as its interchange format. From then on, `sdpack` was the change-packing tool of choice, and the other tools were also-rans.

Now that Windows is using git as its source control system,[3] combined with Visual Studio Team Services for its online presence, the question arises of how to package changes in git.

- Sending changes to another developer to get their opinion: Push your changes to a branch on VSTS. The other developer can view the branch online, or fetch it to their local machine for a closer look.
- Sending changes to another developer to get a buddy build: Push your changes to a branch on VSTS. The other developer can fetch it to their machine to build it. Or even better, let continuous integration do the work of the buddy build.
- Sending changes to another developer to get a code review: Push your changes to a branch on VSTS, and then create a pull request.

- Packing changes and attaching it to a bug: Push your changes to a branch on VSTS, create a pull request, and put a link to the pull request in the bug. Ship Room can open the link and view the pull request online. From there, they can add comments to the PR, or approve it via the Web page.

The one case that I still don't have a good solution for is the one where you have some changes that you decided not to commit, but you want to keep them in case they come in handy later.

Option 1: Put a commit at the tip of the branch summarizing what this branch is about, and then abandon the branch but don't delete it. You can always come back to it later. The downside is that the name of all your "things that didn't work" sit there cluttering your `git branch` output. You can mitigate this by using a naming pattern like `w/raymond/archive/blahblah`. push the branch to the server, and then delete the local version. But the branches still show up in your daily workflow.

Option 2: Create a patch and save that. The downside of this is that patches decay as the code changes, and eventually one of the hunks will fail to apply, and now you're stuck. Since patches record only a few lines of context on either side of the change, there isn't enough information to figure out where the hunk moved to. And even though the patch is the diff itself, you often want to see more context around the changes than was recorded in the patch. Patches also don't record the base revision from which they were generated; you'll have to add that information to the patch file.

Option 3: Create a bundle. The bundle is a compact representation of the objects related to your branch that aren't in the main branch. However, looking at the contents of a bundle is cumbersome because you have to fetch it into a live repo and then remember the starting point for the diff. You cannot view the contents of a bundle without a live repo.

Option 4: Push the changes to a branch on VSTS, create a pull request, and immediately reject it. Save a link to the pull request. You can delete the branch; the pull request will remain, with your commits. Bonus feature: You can annotate your changes by commenting on the pull request. ("This was my attempt to do X. It basically works, but there's a bug when a Y occurs at the same time. See additional discussion at line 123.")

I haven't found a good answer to this yet. As a stopgap, I have a script that generates an `sdpack` from a commit (or commit range), and I use the `sdpack` as my unit of cold storage.

If you have any other ideas, please share.

[1] This source control system somehow got turned into a product, named Microsoft Delta. It was very short-lived.

[2] If you're lucky, your project has a tools magician. This is the person who somehow has their hands in everything that makes your life better, whether it's improving the build system, managing the gated check-in system, setting up the automated static analysis tool, or fixing build breaks in the middle of the night so your team has a working build in the morning.

[3] Another consequence of Windows moving to git as its source control system is that I no longer get email from people asking for help with `bbpack`. Finally I succeeded in getting people to stop using it!

(Stephan hasn't been so lucky, though. He tells me that he has a customer who is packing and reapplying gigabytes of changes and cares more about speed than package size. So he's making still more changes to `sdpack` to support that scenario. "The customer is used to `jjpack`'s speed, so it's fun to have that bar to shoot for.")

Raymond Chen

Follow