Episode Book

EPISODE #6163

Forking a Keyboard for One Voice Feature

PUBLISHED	20 October 2026 at 16:22 UTC
DURATION	22:07
LISTEN	myweirdprompts.com/episode/forking-keyboard-voice-patch-49354/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	anthropic/claude-sonnet-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.11
BOOK GENERATED	21 October 2026 at 07:50 UTC

SYNOPSIS

Daniel has a working voice model that flips between English and Hebrew, shipping as a standalone Android dictation app. The obvious next move — fork an open source keyboard and wire the engine in — comes with a catch: the moment you fork, you inherit the upstream's entire future. This episode works through the patch-based alternative: keep your integration as a tiny replayable patch, let automation handle the mechanical parts, and let a human test the build before it ships. Along the way: what HeliBoard's voice key might already do, why GPL versus Apache decides your architecture, why a new application ID is non-negotiable, and why crediting the chain of volunteers matters more than the label you pick.

TRANSCRIPT



Corn

A voice model that follows a sentence from English into Hebrew and back without choking is a rare thing. Daniel has trained one, on top of Parakeet, and it ships today as an Android app that downloads its weights on first run. It only does dictation. People keep HeliBoard or FlorisBoard or whatever they like, and his app sits beside it. The itch is that two installs is one more than most people will tolerate. So he's wondering about forking an open source keyboard, wiring the engine in, and handing people one app. The catch is that the second you fork somebody's keyboard, you inherit its entire future.



Herman

Yes, it can be done, and the answer hangs on one condition. The voice integration has to be a small, isolated patch that gets reapplied on top of every upstream release. Not a branch you merge into forever, and not a pile of edits scattered through the keyboard's code. Daniel's worry is that if he merges everything, upstream changes will trample his integration, and if he merges selectively he's cherry-picking for the rest of his life. That's the wrong frame. He doesn't ignore upstream changes that collide with him. He keeps his work as a patch, takes the new upstream release whole, and replays the patch on top.



Corn

So the condition is that the integration has to be small enough to be a little embarrassing.



Herman

Ideally it's the microphone key handing off to his engine, and almost nothing else. Every extra file he touches is one more place upstream can move underneath him.



Corn

Small until upstream redesigns the toolbar. Then your one button has no wall to hang on.



Herman

Right, and that's the day the automation earns its keep. When the patch no longer applies cleanly, the pipeline doesn't try to be clever. It stops, opens an issue, and ships nothing. Daniel looks at it when he looks at it.



Corn

Ships nothing, as in everyone stays on last month's build.



Herman

Which is fine, Corn. Last month's build works. A broken release that goes out automatically is the thing you can't take back. What would you do, honestly, if it's a Tuesday and upstream shipped something you can't patch onto?



Corn

Leave it. Nobody has ever been hurt by a keyboard that didn't update on Tuesday.



Herman

Here's roughly what I'd build. A scheduled job looks for a new upstream release tag. If there isn't one, it goes back to sleep. If there is, it checks out that exact tag into a clean directory, applies the voice patch, builds, signs, and parks the result as a draft release. Any step that fails, it opens an issue with the log and stops.



Corn

A patch that applies is not a patch that works. Upstream could move the mic key somewhere and your patch still lands, on the wrong thing.



Herman

That's the nasty one, yes. Applying without complaint just means the surrounding lines still match. So the draft stays a draft until he's tried it on a phone, and a build check that the key is actually there would catch the cheap cases. I'm not sure how much he can automate beyond that, honestly. UI behaviour is hard to test from a robot.



Corn

So it's automated to the doorstep and a human carries it in.



Herman

For a one-man project, that's the right amount of automated. There's a precedent for the shape of this. VSCodium is a rebuild of Microsoft's editor, and its own readme says "This is not a fork." It keeps a folder of patches, and a daily job checks whether Microsoft has put out a new release. If there isn't one, it exits early. If there is, it fetches that release, applies the patches, and builds. The patches are the product.



Corn

A fork that insists it isn't one.



Herman

It's a different relationship to upstream. A long-lived fork carries its own history and merges in the other side. VSCodium carries a stack of instructions and starts fresh from each release. For one person with one tiny feature, I think the second shape suits Daniel better, because the history he'd be maintaining is nothing but his own changes. You can also do the same thing inside git, by rebasing. Your commits get lifted off the old upstream and set down on the new one, one at a time, and git stops at any that no longer fit. A folder of patch files does it more bluntly, and it's easier to read for one small change. Either way it comes to replaying, not merging. Merging blends two histories, and once they're blended, finding which lines were yours is archaeology.



Corn

Archaeology on your own code. That's a bleak weekend. And the automatic part. Is that the hard bit?



Herman

The pulling part is easy. There's a GitHub Action, written by a developer named aormsby, that syncs a fork with its upstream on a schedule. You give it the upstream repo and branch, and it reports whether there were new commits. What it doesn't document is what happens when there's a conflict. So you wouldn't lean on it to decide anything. You'd use it, or something like it, to notice that upstream moved, and then a script of your own does the replay and fails loudly.



Corn

So the cron job is the doorbell, not the butler.



Herman

Fair. A doorbell that asks a build to run.



Corn

Where does the patch go, though? If I'm HeliBoard, I already have a voice key. People report one, anyway.



Herman

People do, and I haven't seen it confirmed in the source how that key works. I'd rather say that plainly. It might hand off to another input method, it might use Android's own speech recognition intent, or it might be something else. And the answer matters a lot, because that decides whether he needs a fork at all.



Corn

How so?



Herman

Android's recognition intent is a request to start whatever speech recogniser is installed. You launch it for a result, and you have to catch the case where nothing answers. If HeliBoard's key already works that way, or hands to an installed voice input method, then Daniel's app might already be the thing it hands to, no fork. I think of it as the split we went through before, keyboard on one side and voice engine on the other, and Android designs for them to be separable.



Corn

Which is a bit of a blow for the fork. You've done all the licence reading and it turns out the key was already pointing at you.



Herman

It might be. I don't know. People do report pinning a voice button in HeliBoard and pairing it with other voice apps, so the road exists. FlorisBoard is different. Its project page says nothing about voice input or a microphone key that I can find. So there he'd be adding a key, not rewiring one, and the patch gets bigger.



Corn

There goes tiny.



Herman

Bigger, yes. Still a patch, but you'd be asking yourself whether you picked the right base.



Corn

Wrong base, and tiny is tiny in name only.



Herman

Before we dig deeper into the fork question, I believe we have a call from a listener. And Corn, it's your mum. How wonderful.

D

Dorothy

Hello dear, it is your mother. Time for my grim fact, dear.

D

Dorothy

There is a little tool that keeps a fork in step with the project it came from. Its own page says it is not currently in active development. So it sits there, half forgotten, while everyone's forks still lean on it.

D

Dorothy

Anyway. Are you eating? I made too much casserole, and your laundry is still here.



Corn

Mum. Right. So, the licence, which is where a base gets chosen whether Daniel likes it or not.



Herman

HeliBoard is GPL version three overall. People hear Apache in connection with it and assume it's the permissive one, but the Apache licence applies to code it carries over from the Android open source keyboard. The project as a whole is GPL, and it describes itself as a fork of OpenBoard.



Corn

So a fork of it is GPL too.



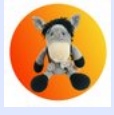
Herman

The licence says the whole work has to go out under it. And whatever is linked into the keyboard, the engine glue, the model loader, is inside that work. If he ships the binary, he has to offer the matching source, under the same licence.



Corn

He's open source anyway. Is that a problem or a Tuesday?



Herman

Probably a Tuesday. But he should choose it on purpose. There's a clause about separate programs sitting side by side in an aggregate, and as I read it, a standalone voice input method that talks to the keyboard over Android's ordinary messaging looks a lot more like that than code compiled into the same app does. I wouldn't call that settled. I'd call it a reason to decide the architecture rather than drift into it.



Corn

Meaning if he wants the engine under a different licence, or keep the weights out of it, the standalone app is the clean way.



Herman

Yes, though the model weights have their own licensing question, and that's a separate conversation I haven't dug into. FlorisBoard is plainer. It's Apache two, copyright held by the project's contributors. He can combine it with his own code more freely, so long as he keeps the notices.



Corn

What notices?



Herman

Four things worth knowing. Modified files carry a prominent note saying they were changed. Existing copyright and attribution lines stay. If the project ships a notice file, its attribution is reproduced in what you distribute. And you get no permission to use the project's name or trademarks beyond saying where it came from. GPL has its own version of the first one: a prominent notice that you modified it, with a date.



Corn

With a date. That's the clause that wants a changelog.



Herman

It's surprisingly easy to be sloppy about.



Corn

Name, then. What does he call this thing?



Herman

Not HeliBoard. Not HeliBoard Voice. HeliBoard's readme says nothing I could find about forks, project names or rebranding, so there's no permission to lean on, and no prohibition either. The safe default is a new name, a new application ID, and a plain line saying what it's built on. Ideally he asks the maintainer first. Even if the answer is a shrug, he's asked.



Corn

New application ID matters for more than manners.



Herman

It does, because otherwise his build and the original are the same app as far as Android is concerned. Updates collide. People install his build over theirs, or the reverse, and bug reports end up in the wrong inbox. Also, a fork needs its own signing key, and I'd treat that key as the most precious file he owns, since people can't just carry on updating if it's lost. I haven't looked at how people store those for fork builds, so take that as general Android, not research.



Corn

What do you call this arrangement, in the trade? Daniel asked.



Herman

Downstream fork is the plain term. Some people say a patched build or a distribution, when the project mostly tracks upstream and adds a layer. There's no official word that I know of. What he says on the project page matters more than the label.



Corn

The ethics, then. I'd say they're mostly the licence plus one extra thing. You're riding on years of volunteer work.



Herman

The extra thing is generosity. Credit by name in the app itself, not buried in a licence file. HeliBoard's own readme credits OpenBoard, the Android open source keyboard, LineageOS, Simple Keyboard, Indic Keyboard and FlorisBoard, and the icon artist. That's a chain of people who each did the same thing, and he's the next link.



Corn

So credit the chain, not just the last link.



Herman

Yes, and send fixes upstream when the fix isn't about voice. If he finds a bug in the keyboard, the right move is a pull request, not a private patch he carries. Otherwise he's just making upstream's work harder to follow and calling it a feature. We did the general fork-or-contribute question a while back, and the upkeep of long feature branches, so I won't repeat it, but the short version holds here.



Corn

Sure, though the sloth in me notes a fork is the more strenuous way to be useful to someone. There's a wrinkle in that chain. HeliBoard is itself a fork. If Daniel builds on it, he's the third link in a row.



Herman

And that's reassuring, not worrying. It tells you forks of keyboards can live for years. It also gives him an exit. If HeliBoard slows down or changes direction, a patch that's small and kept separate can be replayed onto a different base. A big tangled fork can't. You're locked to whoever you forked.



Corn

The small patch is also the escape hatch.



Herman

Right, so the discipline pays twice.



Corn

And tell users where to take their complaints. Keyboard bugs go to the people who wrote the keyboard.



Herman

Only after checking it isn't his patch. Which gets us to the part nobody enjoys: he now owns support for a keyboard. Autocorrect, gesture typing, a field in some banking app that eats the cursor, and every one of those arrives as a message addressed to him.



Corn

Daniel's current app has exactly one job. If dictation fails, it's his fault, and he knows where to look.



Herman

Whereas a keyboard fork fails in a hundred ways that aren't his. I think of it as the cost, not a reason to stop. Releases are the other part. F-Droid has settings for how it notices a new upstream release. One mode checks git tags for the highest version. Another creates new build entries automatically when it detects an update. Or he can set it so nothing goes out without him. For a fork where the patch can break, the manual setting is the one I'd start with.



Corn

Start manual and automate when it's earned its keep. I'd do the same, and I do things slowly on principle.



Herman

One thing I'm not sure about. F-Droid has a list of flags it can put on an app, like non-free dependencies or non-free assets, and none of them is about forks. Whether a model that's downloaded after install touches any of them, I don't know. He'd want to ask them before building his release plan around it.



Corn

Ask before you build the plan. Revolutionary.



Herman

It saves a surprising amount of rework, Corn. HeliBoard itself ships through F-Droid, GitHub releases and IzzyOnDroid, so there's a template for where his users already look. One forum report, which I'd hold loosely, says a downstream HeliBoard build had its voice key vanish from the toolbar for one version and come back in the next. That's the kind of thing a patch layer can do to you quietly.



Corn

My view is he should find out the cheap thing first. Spend an afternoon reading how the voice key works before deciding anything about forks.



Herman

Same. And then there's a test I'd give him. Make the patch on a branch, count the files it touches, and decide from the count. One or two files, a fork is realistic. A dozen, and he's writing a keyboard whether he likes it or not.



Corn

A dozen files is the keyboard growing a second owner.



Herman

Which is the day he finds out what a keyboard costs.



Corn

So where do I land. I'd keep the standalone voice app as the product. It's one job and he does it well.



Herman

I'd push the other way, Corn. People install one app. Two installs, two permissions screens, a setup page telling them to enable one input method and switch to another. Every step loses somebody, and Daniel can't see which ones.



Corn

A fork loses people too. Whoever opens it and finds their autocorrect is a little off.



Herman

Fair, but think about who installs a dictation app for Hebrew and English mixed together. They're already someone who typed a message to a friend and watched the keyboard butcher half of it. Would that person switch keyboards for a better microphone? I'd guess yes. You tell me if I'm wrong.



Corn

I'd guess they'd switch once, and then never forgive a single autocorrect miss. People's keyboards are their handwriting. The download on first run is the same either way. The weights come down once, whichever app asks for them.



Herman

Granted. So what would you do, if you had to ship one?



Corn

Both. The voice app stays the product, the thing he's proud of and supports. The fork is a convenience build, made from the same engine, with a patch that stays tiny. If upstream breaks it, nobody loses anything. The standalone app still works.



Herman

That actually sounds right to me. The engine lives in one place, and the keyboard build is a thin wrapper around it that he's allowed to let lapse. If the patch ever gets unmanageable, he drops the wrapper and nobody's dictation breaks.



Corn

And if the convenience build gets popular?



Herman

Then he'll know the one-app instinct was right, and he can decide whether to take on the keyboard properly. That's a good problem to arrive at on evidence.



Hilbert

Yes, keep it small. I've got the manual for a desk fan. Model number seven hundred and twelve, grey, three speeds, which has not been in the building since the ceiling tile came down on it. I kept the manual. The receipt is stapled inside the front cover. Forty-two pounds ninety-nine, from a shop in Edgware that is now a nail bar.



Herman

Paper outlasts the product more often than people expect.



Hilbert

Thirty-one pages. The manufacturer wrote two. Twenty-nine of them are mine. I rewrote it, you see. Edition one is the manufacturer's, with my corrections in pencil. Edition two I wrote in March, and I do not accept edition two.



Corn

You wrote it and you don't accept it.



Hilbert

Tuesday-me wrote edition two. Thursday-me is the one who has to live by it. Page nineteen, the part on oiling the bearing, says every fourth Sunday. Thursday-me has been oiling every third. For a fan that is not there. I oil the air.



Corn

Every third Sunday, it sounds like.



Hilbert

Every third Sunday. With a can I bought for the fan, which I also have the receipt for. Three pounds ten. The can is empty. It was empty when I bought it.



Herman

Applying a procedure to nothing is a very clean test of the procedure.



Hilbert

Herman, you said "applies cleanly" like it was a man you knew. Edition two does not apply cleanly. I sent it back.



Corn

Sent it back to whom?



Hilbert

To the flat. Registered post, signature required, eleven pounds forty each time. I sign for it at twenty to seven. I refuse delivery at ten to. The slip says addressee gone away. It is my own handwriting on the slip, and I am left-handed on that one.



Corn

Which you aren't.



Hilbert

Not on any other form. Page nineteen is all it is. There's also a footnote on page twenty-six I am treating as a separate matter. It says the fan may be used out of doors in a covered porch. Nobody I know has a covered porch. Both of us agree about the porch and neither of us will sign the same line. And the wiring diagram on page nine, which neither of us touches. We have not spoken since the nineteenth. I took a stamp out for it. A rubber one, it says NOT ACCEPTED, and it took nineteen days to get here, so I used the pencil in the meantime. Edition two took eleven days to write. The pencil took nineteen to wear down.



Corn

The stamp is the slow part.



Hilbert

The stamp was always going to be the slow part. Edition two I could do in eleven days. The shop wanted nineteen to engrave NOT ACCEPTED, with the full stop.



Herman

Rejecting the one page and keeping the other thirty is at least a method.



Hilbert

There's a drawer for what's rejected. Fourteen sheets so far. I count them on Fridays, and the count has come out at fifteen twice, which I have not looked into.



Corn

Somebody's adding sheets.



Hilbert

The diagram on page nine is very good work, and whichever of us drew it knew exactly what he was doing.



Corn

Right. Then the paperwork goes first. The name, the signing key and the application ID take as long as they take, and the patch can be written while they're on their way.



Herman

Thanks to Hilbert Flumingtop for producing, as always.



Corn

This has been My Weird Prompts. There's more of it at my weird prompts dot com.



Herman

See you tomorrow.