PRACTICAL PORTS

Can't Git Enough?

BY BENEDICT REUSCHLING

This column covers ports and packages for FreeBSD that are useful in some way, peculiar, or otherwise good to know about. Ports extend the base OS functionality and make sure you get something done or, simply, put a smile on your face. Come along for the ride, maybe you'll find something new.

You may be aware that FreeBSD 13.0 is the first major release cut from Git instead of Subversion. This has long been in the making with a lot of careful handling of the source that makes FreeBSD so valuable. I clearly remember a certain FreeBSD devsummit in Maarsen, Netherlands many years ago, where the FreeBSD project decided to finally jump on the Subversion bandwagon. Coming from CVS (and probably rcs before that, ask the people who've been around longer than I...), switching version control systems is certainly not easy. Especially when you have a history going back to the very first days at UC Berkeley, every single change is precious. You never know when you'll need to dig up some obscure historical fact because a device driver misbehaves, or a developer needs to know why an interface was implemented in a certain way. But switching version control systems is not just a technical task, it's a social one, too. It comes with requirements for convincing and bringing onboard the people who will use it after the switch has been done. There is plenty of controversy about Git and how it behaves. Relearning some of the concepts of version control systems and how Git does it is probably the best way to deal with it, along with an open-minded approach, of course.

There is plenty of controversy about Git and how it behaves.

Certainly, Git was around the ports collection before the switch happened and many a developer has used it for years for their own personal projects or at work (sometimes without having a choice). People missing the "central-source-of-truth" approach from Subversion et al. can set up a Gitlab system using www/gitlab-ce. There are plenty of extra ports to keep you

busy setting it up on a rainy, lockdown day. The graphical user interface hides many of the perceived warts of Git behind a user-friendly interface. From replacing files by uploading a changed version that turns it into a commit and push to reading the version history, all is done without needing to know a single Git command. This and other similar UIs like www/gitea, devel/cgit, devel/git-cola, make it easy for non-developers to keep track of documents in an office setting, no matter if it is IT-related or not. The little "time machine for files" is useful to pretty much anyone needing an old version of a file the way it was before your cat managed to not only walk across the keyboard, but also save the document in the process. Extra points for the rodent hunter when you were in vi at the time.

Software development never seems to be an easy task, so all the help one can get is welcome. I've always wondered how developers start: do they think of a name for their software first or start to hack on it right away? If you can't think of a good name and asdf, qwerty, and similar are already taken (we won't discuss them here, I promise), how about doing the old reverse-y thing? This must have inspired the author of devel/tig who wrote an ncurses-based terminal interface for Git. Because why not? Browsing your version-controlled tree is quick this way and staging your next change, looking at the history, writing that bad, one-word commit message is all possible (the latter is discouraged though--give the historians a bit more information about why you made that change at 4:20 in the morning).

Software development never seems to be an easy task, so all the help one can get is welcome.

More than once in my Unix class, I have told students that the Unix developers were lazy, but the good kind of laziness. What I mean is that they sat down, figured out how their colleague's computer could do the work much better and put a lot of effort into it. Once it was done, they could be lazy because the silicon was doing all the hard work. It is not lazy for the sake of laziness, which is probably the highest form of procrastination. But whatever it may be, if you are in the same camp, take a look at devel/lazygit. A terminal UI in Go which starts with a nice rant on its project page:

"Rant time: You've heard it before, Git is powerful, but what good is that power when everything is so damn hard to do? Interactive rebasing requires you to edit a damn TODO file in your editor? Are you kidding me? To stage part of a file, you need to use a command line program to step through each hunk, and if a hunk can't be split down any further and contains code you don't want to stage, you have to edit an arcane patch file by hand? Are you KIDDING me?! Sometimes you get asked to stash your changes when switching branches only to realize that after you switch and un-stash, there weren't even any conflicts, and it would have been fine to just check out the branch directly? YOU HAVE GOT TO BE KIDDING ME!"

But after the ranting, the developer sat down and made life better for everyone. See, everyone can do it. The UI is certainly nice enough to give it a try. Another such tool for command line aficionados is devel/glab. Written for Gitlab, it lets you leave the familiar territories of the web UI and stay in the terminal where the real fun is happening.

More often than not, source code involves collaboration with others. My code certainly got better when another set of eyes took a look at it. After rolling them back from the top of their heads, people were not shy to point out where I could do better, sharing their wisdom along the way. The occasional praise was also there, so I did not start my new career as a salami smuggler. Reviewing on Github (where not only the cool kids hang out these days, but pretty much anyone needing a repo for their files) is often done using the Gerrit code review tool. If you spend a lot of time there, consider installing devel/git-review to support your coding workflow and review with others.

Beginners can take a look at devel/easygit to make the experience a little less painful or overwhelming. For those who have used it for a long time and want to brag about how many lines of code were changed just because they commit early and often, devel/gitinspector might help. Just decorate your own office—and not the cafeteria—with the resulting stats, or people will quickly remind you that it is quality not the quantity that counts.

Git is not the only version control game in town, there have been and always will be others. Its popularity certainly can't be overlooked and it must have come from a feature or two that people were missing. What's the easiest thing I can think of if you want version control but don't want Git? Keep multiple drafts in your email, attach the files you were working with or paste the content in the message body. It is even distributed if you pick up your work from another computer by going into your webmail. Surely that has its downsides, as it will be a security nightmare if you give people access to your "repo." But maybe that experience will haunt you enough to give Git a first, second, or third chance. Pick up a book, do one of the many online tutorials. The old "learning by doing" is also very effective. There can never be enough people contributing to open-source projects after all. Commit yourself to it and don't forget to pick the cherries along the way.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Together with Allan Jude, he is host of the weekly bsdnow.tv podcast.