FreeBSD Code Review with git-arc

BY JOHN BALDWIN

The FreeBSD Project uses the Phabricator Differential application as a tool for code review at https://reviews.freebsd.org. Phabricator itself provides several applications to support software development, but the FreeBSD Project only uses the code review tool. Users and developers can upload changes for review either by pasting diffs directly into the web application, or by using the **arc** command line tool (available via the devel/arcanist package or port). arc can upload commits from a git branch from the command line and can also modify commit logs of reviewed commits to prepare them for pushing to the public repository.

However, **arc** has a few limitations that make it awkward to use for FreeBSD development:

- Arcanist uses its own commit log template whose format does not match FreeBSD's existing template and is not easily changed.
- Arcanist presumes a model where all the commits in a development branch are uploaded for review as a single Differential revision. When working on a feature branch with multiple commits, it is usually more efficient to review each commit individually.

While these limitations can be worked around (for example, individual commits can be uploaded as separate reviews by careful use of the `--head` option), the work arounds are tedious.

The git-arc tool provides a wrapper around **arc** that mostly mitigates those limitations. git-arc is a plugin for git which adds commands to work with Phabricator.

Installing git-arc

git-arc lives in the FreeBSD source repository at `tools/tools/git`. The script and its manpage can be installed by running make install from that directory. Note that git-arc is installed to `/usr/local/bin` by default.

```
> git clone https://git.freebsd.org/src.git
...
> cd src/tools/tools/git
> make
gzip -cn /usr/home/john/work/freebsd/main/tools/tools/git/git-arc.1 >
git-arc.1.gz
> make install
install  -o root  -g wheel -m 555
/usr/home/john/work/freebsd/main/tools/tools/git/git-arc.sh
/usr/local/bin/git-arc
install  -o root -g wheel -m 444 git-arc.1.gz  /usr/share/man/man1/
```

In addition, git-arc requires the following packages to be installed: arcanist (devel/arcanist), git (devel/git), and jq (textproc/jq).

Since git-arc is a wrapper around arcanist, arcanist must also be initialized. First, create an account at https://reviews.freebsd.org. Second, install an API token for arcanist:

```
> arc install-certificate https://reviews.freebsd.org
 CONNECT  Connecting to "https://reviews.freebsd.org/api/"...
LOGIN TO PHABRICATOR
Open this page in your browser and login to Phabricator if necessary:

https://reviews.freebsd.org/conduit/login/

Then paste the API Token on that page below.

    Paste API Token from that page: cli-XXXXX
Writing ~/.arcrc...
 SUCCESS!  API Token installed.
```

Using git-arc For a Single Commit

The first step in reviewing a commit is preparing the commit for review. The commit should be a commit candidate with a suitable log message. Any fixups should be squashed back into the commit so that the commit matches what would be pushed to the tree.

Creating the Review

Once the commit is prepared, the next step is to create a review for the commit via the create subcommand. This command accepts a reference to the commit (e.g., a hash, or a symbolic reference such as HEAD) as a positional argument after any options. The create subcommand creates a new review in Phabricator using the commit's log message to set the review title and description. Reviewers can be added via the `-r` option. Multiple reviewers can be specified as a comma-separated list or via multiple `-r` options. A group can be added as a reviewer by prefixing it with the '`#`' character, e.g. `#bhyve` to tag developers working on the bhyve(8) hypervisor. This example creates a review for a commit on the **gdb_11** branch to the `devel/gdb` port marking the port maintainer (pizzamig@FreeBSD.org) as a reviewer:

```
> git log --oneline main..gdb_11
6b21ea620990 devel/gdb: Update to 11.1.
> git arc create -r pizzamig gdb_11
commit 6b21ea62099007a0d376852731cdbde4d8d522d9 (HEAD -> gdb_11)
Author: John Baldwin <jhb@FreeBSD.org>
Date:   Thu Sep 16 17:25:13 2021 -0700

    devel/gdb: Update to 11.1.

…
Does this look OK? [y/N] y
...
```

The `create` subcommand first displays the commit including both the log message and the patch. It then displays a prompt confirming if a review should be created. Answer '`y`' to continue and create the review.

Updating the Review

The update subcommand can be used to update the review after changes are made to the patch. First apply any changes to the git commit in question either by amending the commit or by committing new changes and then squashing them back into the original commit. Note that the update command only updates the patch in the review. It does not update the review's title or description, nor does it permit adding additional reviewers or subscribers. Those changes must be made in Phabriactor's web application instead.

Note: The update and stage subcommands use the first line of commit logs to find a review with an identical title. If the first line of the commit log message is changed, the title of the review must be updated in Phabricator before git-arc will properly recognize the existing review for a commit.

This example updates the review for the devel/gdb port update after amending the original commit with fixes from reviewer feedback:

```
> git arc update gdb_11
commit 8d532ac873633ae12d42be709755f56f9d86c310 (HEAD -> gdb_11)
Author: John Baldwin <jhb@FreeBSD.org>
Date:   Thu Sep 16 17:25:13 2021 -0700

    devel/gdb: Update to 11.1.

...
Does this look OK? [y/N] y
...
```

As with the `create` subcommand, update displays the log message and patch followed by a prompt to confirm the update. Answer '`y`' to continue and update the review. Next, an editor window (using the editor configured in the user's `$EDITOR` environment variable) will open. Enter a description of the changes made for this update, save the file, and exit the editor. This description will be added to the review as a comment.

Finalizing the Review and Pushing the Commit

Once the commit is ready to be published, the `stage` subcommand merges the commit to the local main branch and amends the commit log with metadata from the review. Specifically, `git arc stage` adds properly formatted 'Reviewed by' and 'Differential Revision' tags to the commit log. After the operation completes, it leaves the working tree set to the new tip of the main branch including the staged commit. The commit can then be reviewed via `git show` before pushing to the public tree. This example shows the final push of the update to `devel/gdb`:

```
> git checkout main
> git fetch freebsd
> git merge freebsd/main
> git arc stage gdb_11
> git push freebsd
```

The first three commands ensure the local main branch is up to date before staging the commit. The stage subcommand pops up an editor window with the updated commit log after merging the commit. This provides an opportunity to fix any formatting issues in the commit log. Save the commit log and exit the editor to continue.

Note that the `stage` subcommand fails if it encounters conflicts when merging the commit onto `main`. To resolve, rebase the original commit onto the updated main branch. If the changes to fix the conflict resolution warrant review, then update the review. Otherwise, re-run the stage subcommand with the rebased commit.

Using git-arc For a Branch

While git-arc is useful for individual commits, it provides the greatest benefit when working with a branch containing multiple commits. The `create`, `update`, and `stage` sub-commands all accept multiple commits in a single invocation. Commits can be identified either via individual references as in the single commit examples above, or as Git revision ranges.

Creating the Reviews

For branches, git-arc creates reviews for each commit. These commits are linked together into a **stack** in Phabricator. The review for the first commit in the branch is marked as a parent revision of the review for the second commit and so on. This permits all of the branch commits to be found in the Phabricator UI from the Stack tab when viewing a review for any individual commit in the branch.

By default, the create subcommand will display the log message and patch for each commit, prompting after each commit. For a branch with many commits, this step can be tedious, so `create` accepts an optional `-l` argument. If this argument is given, then create will list all of the candidate commits with a single confirmation prompt. If the user answers '`y`' at the prompt, then the stack of reviews are created without further prompts. This example creates reviews for all of the commits for a branch checked out in the current directory. The branch was created as a branch off of `main`:

```
> git arc create -l -s emaste main..
2f7e09973ab6 cryptodev: Use 'csp' in the handlers for requests.
fbc805bb4d62 ccp, ccr: Simplify drivers to assume an AES-GCM IV length of 12.
8390644fd45f crypto: Permit variable-sized IVs for ciphers with a reinit hook.
f08d44eaa9ee cryptosoft, ccr: Use crp_iv directly for AES-CCM and AES-GCM.
43aaceb5afef cryptodev: Permit explicit IV/nonce and MAC/tag lengths.
0190b9e740d8 cryptodev: Permit CIOCCRYPT for AEAD ciphers.
03f07b455c80 cryptodev: Allow some CIOCCRYPT operations with an empty payload.
e0caaccf1ec0 cryptocheck: Support multiple IV sizes for AES-CCM.
ae4a0338bc8b crypto: Support multiple nonce lengths for AES-CCM.
cb18504c7712 aesni: Support multiple nonce lengths for AES-CCM.
60e1a45f2201 aesni: Handle requests with an empty payload.
aa653be04078 aesni: Permit AES-CCM requests with neither payload nor AAD.
fafcdb583930 aesni: Support AES-CCM requests with a truncated tag.
75c003b9ccbb ccr: Support multiple nonce lengths for AES-CCM.
```

```
de32cc23b0f9 ccr: Support AES-CCM requests with truncated tags.
6a88ac41d972 safexcel: Support multiple nonce lengths for AES-CCM.
ff4f260a5fd9 safexcel: Support truncated tags for AES-CCM.
e46422be0eaf cryptosoft: Fix support for variable tag lengths in AES-CCM.
d98f930cd833 crypto: Test all of the AES-CCM KAT vectors.
7ee5d373884e crypto: Support Chacha20-Poly1305 with a nonce size of 8 bytes.

Does this look OK? [y/N] y
...
```

If one wishes to create individual reviews for commits on a branch without linking the reviews together, that can be done by invoking `git arc create` separately for each commit. The relationship between reviews can also be adjusted in the Phabricator web interface.

Checking Review Status

When working with a branch, it can be useful to examine the status of the individual reviews associated with a branch. This can be done via the `list` subcommand. The list subcommand accepts one or more commit names or commit ranges and outputs a single line of status for each commit. For commits without an associated review, the status is reported as `No Review`.

This example shows the status of several commits on a `gcc9_universe` branch. For this particular branch, the author has chosen to review commits individually rather than as a series. As such, some commits are not yet ready for review while others have been approved for merging to `main`.

```
> git arc list main..gcc9_universe
f0f665a2f4a5 Accepted     D26202: Switch to GCC 9 for the GCC tinderbox.
a98a78e2dabc Accepted     D26203: Pass -msecure-plt to GCC for 32-bit powerpc.
b4412a18ab23 Needs Review D31933: hyperv storvsc: Don't abuse struct sglist to hold virtual addresses.
a0eaf413441a Accepted     D31934: kernel: Disable errors for -Walloca-larger-than for GCC.
daf618e9a8c4 No Review    : Fix various places which cast a pointer to a vm_paddr_t or vice versa.
8c46bb47a57f Needs Review D31938: bhyve: Add an empty case for event types in mevent_kq_fflags().
46d2ac2e7b3b Accepted     D31941: Use a char * to avoid alignment warnings.
b2d94deacfa1 Needs Review D31945: libmd: Only define SHA256_Transform_c when using the ARM64 ifunc.
c9be458cee94 Accepted     D31948: mana: Cast an unused value to void to quiet a warning.
8a9b7debfc0c No Review    : arm64: Add compat macros for system registers for GNU as.
```

Updating Reviews

Using the update subcommand with a branch is not quite as straightforward as the other commands. In particular, the update subcommand is not able to determine if the review associated with a commit is already up to date (and thus should not be updated). Instead, it will always update all reviews if it is given the list of reviews for a branch. The update subcommand also prompts for a description for each commit. With a branch, it is generally better to use the update subcommand for individual commits after they have been changed rather than running the subcommand against an individual branch.

Finalizing the Reviews and Pushing the Branch

Once all the commits are ready for pushing, the stage subcommand can be used to stage all of the commits in the branch in a single operation. The user's editor will be invoked to finalize the commit log of each commit. Once the operation completes, the branch can be pushed via `git push` as described earlier in the single commit example.

Individual commits in a branch can also be pushed by using the stage subcommand with specific commits. After pushing those commits, rebase the branch to remove the pushed commits from the branch. The `list` subcommand can then be used to examine the status of the remaining commits on the branch.

Limitations and Caveats

While git-arc provides a streamlined interface on top of **arc**, it has several limitations of its own:

- Matching a commit with an existing review requires the first line of the commit log to match the title of the review. This means that if you update the first line of the commit log for a commit with an existing review, the review title must be manually updated in Phabricator before the update, `list`, or `stage` subcommands will recognize the review.
- Similarly, the update subcommand is only able to update the patch for a given review. It does not update the review description if the commit log message has changed.
- If commits are dropped from a branch while it is in review, git-arc does not provide a way to notice that commits are dropped or to abandon the associated reviews. Abandoning the reviews must be done in Phabricator instead.
- If additional commits are added to a branch while it is in review, the list subcommand can be used to find those commits and the create subcommand can be used to create reviews. However, git-arc does not provide a way to auto-adjust the parent / child relationships in Phabricator to update the stack to match the new layout of the branch. This must be done manually in Phabricator instead.
- git-arc is not able to determine if a commit's review is stale (that is, if a commit has been updated since the review was created or last updated). Such functionality would be useful to annotate stale revisions in the `list` subcommand. It could also make the update subcommand more useful with revision ranges by omitting updates to unchanged commits.

JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for 20 years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger and LLVM. John lives in Concord, California, with his wife, Kimberly, and three children: Janelle, Evan, and Bella.