B4 end-user docs

Kernel.org

Sep 11, 2023

CONTENTS

B4 is a tool created to make it easier for project developers and maintainers to use a distributed development workflow that relies on patches and distribution lists for code contributions and review.

This documentation is split into two main areas – one aimed at maintainers who primarily receive, review, and apply patches to their trees, and the other at developers who submit patches or patch series for maintainer review.

CHAPTER

ONE

INSTALLATION AND CONFIGURATION

1.1 Installing b4

B4 is packaged for many distributions, so chances are that you will be able to install it using your regular package installation commands, e.g.:

```
# dnf install b4
```

or:

```
# apt install b4
```

Note, that b4 is under heavy development, so it is possible that the version packaged for your distribution is not as recent as you'd like. If that is the case, you can install it from other sources.

1.1.1 Installing with pip

To install from PyPi:

```
python3 -m pip install --user b4
```

This will install b4 locally and pull in any required dependencies. If you are not able to execute `b4 --version` after pip completes, check that your `~/.local/bin/` is in your `$PATH`.

Upgrading

If you have previously installed from PyPi, you can upgrade using pip as well:

```
python3 -m pip install --user --upgrade b4
```

1.1.2 Running from the checkout dir

If you want to run the latest development version of b4, you can run it directly from the git repository:

```
git clone https://git.kernel.org/pub/scm/utils/b4/b4.git
cd b4
git submodule update --init
pip install --user -r requirements.txt
```

You can then either symlink the `b4.sh` script to your user-bin directory:

```
ln -sf $HOME/path/to/b4.sh ~/bin/b4
```

or you can add an alias to your shell's RC file:

```
alias b4="$HOME/path/to/b4/b4.sh"
```

Using a stable branch

If you don't want to use the master branch (which may not be stable), you can switch to a stable branch instead, e.g.:

```
git switch stable-0.9.y
```

Updating the git checkout

It should be sufficient to just turn `git pull`:

```
git pull origin master
git submodule update
```

If you notice that `requirements.txt` has been updated, you may wish to run the pip command again:

```
pip install --user -r requirements.txt
```

1.2 Configuration options

B4 doesn't have a separate configuration file but will use `git-config` to retrieve a set of b4-specific settings. This means that you can have three levels of b4 configuration:

- system-wide, in `/etc/gitconfig`
- per-user, in `$HOME/.gitconfig`
- per-repo, in `somerepo/.git/config`

Since the purpose of b4 is to work with git repositories, this allows the usual fall-through configuration that can be overridden by more local settings on the repository level.

1.2.1 Per-project defaults

Note: This feature is new in v0.10.

A project may ship their own b4 config with some defaults, placed in the toplevel of the git tree. If you're not sure where a configuration option is coming from, check if there is a `.b4-config` file in the repository you're currently using.

Configuration options

All settings are under the `b4` section. E.g to set a `b4.midmask` option, you can just edit your `~/.gitconfig` or `.git/config` file and add the following section:

```
[b4]
  midmask = https://some.host/%s
```

1.2.2 Core options

These options control many of the core features of b4.

`b4.midmask`
: When retrieving threads by message-id, b4 will use `midmask` to figure out from which server they should be retrieved.

  Default: `https://lore.kernel.org/%s`

`b4.linkmask`
: When automatically generating `Link:` trailers, b4 will use this setting to derive the destination URL. If you want a shorter option, you can also use `https://msgid.link/%s`, which is an alias for lore.kernel.org.

  Default: `https://lore.kernel.org/%s`

`b4.searchmask` (v0.9+)
: If the public-inbox server provides a global searchable index (usually in `/all/`, this setting can be used to query and retrieve matching discussion threads based on specific search terms – for example, to retrieve trailer updates using a series `change-id` identifier.

  Default: `https://lore.kernel.org/all/?x=m&t=1&q=%s`

`b4.linktrailermask` (v0.13+)
: This allows overriding the format of the Link: trailer, in case you want to call it something other thank "Link". For example, some projects require "Message-Id" trailers, so you can make b4 behave the way you like by setting:

  ```
  linktrailermask = Message-Id: <%s>
  ```

  The `%s` will be replaced by the message-id.

  Default: `Link: https://lore.kernel.org/%s`

`b4.listid-preference` (v0.8+)
: Messages are frequently sent to multiple distribution lists, and some servers may apply content munging to modify the headers or the message content. B4 will deduplicate the results and this configuration option defines the priority given to the `List-Id` header. It is a simple comma-separated string with shell-style globbing.

  Default: `*.feeds.kernel.org, *.linux.dev,*.kernel.org,*`

b4.save-maildirs
The "mbox" file format is actually several incompatible formats ("mboxo" vs "mboxrd", for example). If you want to avoid dealing with this problem, you can choose to always save retrieved messages as a Maildir instead.

Default: no

b4.trailer-order
This lets you control the order of trailers that get added to your own custody section of the commit message. By default, b4 will apply these trailers in the order they were received (because this is mostly consumed by tooling and the order does not matter). However, if you wanted to list things in a specific order, you could try something like:

```
trailer-order = link*,fixes*,acked*,reviewed*,tested*,*
```

The "chain of custody" is an important concept in patch-based code review process, with each "Signed-off-by" trailer indicating where the custody section of previous reviewer ends and the new one starts. Your own custody section is anything between the previous-to-last "Signed-off-by" trailer (if any) and the bottom of the trailer section. E.g.:

```
Fixes: abcde (Commit info)
Suggested-by: Alex Reporter <alex.reporter@example.com>
Signed-off-by: Betty Developer <betty.developer@example.com>
Acked-by: Chandra Acker <chandra.acker@example.com>
Reviewed-by: Debby Reviewer <debby.reviewer@example.com>
Signed-off-by: Ezri Submaintainer <ezri.submaintainer@example.com>
Link: https://msgid.link/some@thing.foo
Tested-by: Finn Tester <finn.tester@example.com>
Signed-off-by: Your Name <your.name@example.com>
```

Your custody section is beneath "Ezri Submaintainer", so the only trailers considered for reordering are "Link" and "Tested-by" (your own Signed-off-by trailer is always at the bottom of your own custody section).

Note: versions prior to v0.10 did not properly respect the chain of custody.

Default: *

b4.trailers-ignore-from (v0.10+)
A comma-separated list of addresses that should never be considered for follow-up trailers. This is useful when dealing with reports generated by automated bots that may insert trailer suggestions, such as the "kernel test robot." E.g.:

```
[b4]
  trailers-ignore-from = lkp@intel.com, someotherbot@example.org
```

Default: None

b4.cache-expire
B4 will cache retrieved threads by default, and this allows tweaking the time (in minutes) before cache is invalidated. Many commands also allow the --no-cache flag to force remote lookups.

Default: 10

1.2.3 shazam settings

These settings control how `b4 shazam` applies patches to your tree.

b4.shazam-am-flags (v0.9+)
Additional flags to pass to `git am` when applying patches.

Default: `None`

b4.shazam-merge-flags (v0.9+)
Additional flags to pass to `git merge` when performing a merge with `b4 shazam -M`

Default: `--signoff`

b4.shazam-merge-template (v0.9+)
Path to a template to use when creating a merge commit. See `shazam-merge-template.example` for some info on how to tweak one.

Default: `None`

1.2.4 Attestation settings

b4.attestation-policy
B4 supports domain-level and end-to-end attestation of patches using the patatt library. There are four different operation modes:

- `off`: do not bother checking attestation at all
- `check`: print green checkmarks when attestation is passing, but nothing if attestation is failing (**DEPRECATED**, use `softfail`)
- `softfail`: print green checkmarks when attestation is passing and red x-marks when it is failing
- `hardfail`: exit with an error when any attestation checks fail

Default: `softfail`

b4.attestation-checkmarks
When reporting attestation results, b4 can output fancy unicode checkmarks, or plain old ascii ones:

- `fancy`: uses ✓/ checkmarks and colours
- `plain`: uses x/v checkmarks and no colours

Default: `fancy`

b4.attestation-check-dkim
Controls whether to perform DKIM attestation checks.

Default: `yes`

b4.attestation-staleness-days
This setting controls how long in the past attestation signatures can be made before we stop considering them valid. This helps avoid an attack where someone resends valid old patches that contain a known vulnerability.

Default: `30`

b4.attestation-gnupghome
This allows setting `GNUPGHOME` before running PGP attestation checks using GnuPG.

Default: `None`

b4.gpgbin
If you don't want to use the default `gpg` command, you can specify a path to a different binary. B4 will also use git's `gpg.program` setting, if found.

Default: `None`

b4.keyringsrc
See `patatt` for details on how to configure keyring lookups. For example, you can clone the kernel.org pgpkeys.git repository and use it for attestation without needing to import any keys into your GnuPG keyring:

```
git clone https://git.kernel.org/pub/scm/docs/kernel/pgpkeys.git
```

Then set the following in your `~/.gitconfig`:

```
[b4]
  keyringsrc = ~/path/to/pgpkeys/.keyring
```

Default: `None`

1.2.5 Thank-you (ty) settings

These settings control the behaviour of `b4 ty` command.

b4.thanks-pr-template, b4.thanks-am-template
These settings take a full path to the template to use when generating thank-you messages for contributors. See example templates provided with the project.

Default: `None`

b4.thanks-commit-url-mask
Used when creating summaries for `b4 ty`, and can be set to a value like:

```
thanks-commit-url-mask = https://git.kernel.org/username/c/%.12s
```

If not set, b4 will just specify the commit hashes.

See this page for more info on convenient git.kernel.org shorterners: https://korg.docs.kernel.org/git-url-shorteners.html

Default: `None`

b4.thanks-from-name (v0.13+)
An custom from name for sending thanks, eg:

```
thanks-from-name = Project Foo Thanks Bot
```

Default: `None` - falls back to user name.

b4.thanks-from-email (v0.13+)
An custom from email for sending thanks, eg:

```
thanks-from-email = thanks-bot@foo.org
```

Default: `None` - falls back to user email.

b4.thanks-treename
Name of the tree which can be used in thanks templates.

Default: `None`

b4.email-exclude (v0.9+)
: A comma-separated list of shell-style globbing patterns with addresses that should always be excluded from the recipient list.

  Default: None

b4.sendemail-identity (v0.8+)
: Sendemail identity to use when sending mail directly from b4 (applies to `b4 send` and `b4 ty`). See `man git-send-email` for info about sendemail identities.

  Default: None

b4.ty-send-email (v0.11+)
: When set to `yes`, will instruct `b4 ty` to send email directly instead of generating .thanks files.

  Default: no

1.2.6 Patchwork integration settings

If your project uses a patchwork server, these settings allow you to integrate your b4 workflow with patchwork.

b4.pw-url (v0.10+)
: The URL of your patchwork server. Note, that this should point at the toplevel of your patchwork installation and NOT at the project patch listing. E.g.: `https://patchwork.kernel.org/`.

  Default: None

b4.pw-key (v0.10+)
: You should be able to obtain an API key from your patchwork user profile. This API key will be used to perform actions on your behalf.

  Default: None

b4.pw-project (v0.10+)
: This should contain the name of your patchwork project, as seen in the URL subpath to it (e.g. `linux-usb`).

  Default: None

b4.pw-review-state (v0.10+)
: When patchwork integration is enabled, every time you run `b4 am` or `b4 shazam`, b4 will mark those patches as with this state. E.g.: `under-review`).

  Default: None

b4.pw-accept-state (v0.10+)
: After you run `b4 ty` to thank the contributor, b4 will move the matching patches into this state. E.g.: `accepted`.

  Default: None

b4.pw-discard-state (v0.10+)
: If you run `b4 ty -d` to delete the tracking information for a patch series, it will also be set on the patchwork server with this state. E.g.: `deferred` (or `rejected`).

  Default: None

1.2.7 Contributor-oriented settings

b4.send-endpoint-web (v0.10+)
The web submission endpoint to use (see *Authenticating with the web submission endpoint*).

Default: `None`

b4.send-series-to (v0.10+)
Address or comma-separated addresses to always add to the To: header (see *Prepare the list of recipients*).

Default: `None`

b4.send-series-cc (v0.10+)
Address or comma-separated addresses to always add to the Cc: header (see *Prepare the list of recipients*).

Default: `None`

b4.send-no-patatt-sign (v0.10+)
Do not sign patches with patatt before sending them (unless using the web submission endpoint where signing is required).

Default: `no`

b4.send-auto-to-cmd (v0.10+)
Command to use to generate the list of To: recipients. Has no effect if the specified script is not found in the repository.

Default: `scripts/get_maintainer.pl --nogit --nogit-fallback --nogit-chief-penguins --norolestats --nol`

b4.send-auto-cc-cmd (v0.10+)
Command to use to generate the list of Cc: recipients. Has no effect if the specified script is not found in the repository.

Default:: `scripts/get_maintainer.pl --nogit --nogit-fallback --nogit-chief-penguins --norolestats --nom`

b4.send-same-thread (v0.13+)
When sending a new version of a series, make it part of the same thread as the previous one. The first mail will be sent as a reply to the previous version's cover letter.

Default: `no`

b4.prep-cover-strategy (v0.10+)
Alternative cover letter storage strategy to use (see *Cover letter strategies*).

Default: `commit`

b4.prep-cover-template (v0.10+)
Path to the template to use for the cover letter.

Default: `None`

To document

b4.gh-api-key
: Deliberately undocumented because the feature is incomplete and poorly tested.

CHAPTER

TWO

FOR MAINTAINERS

2.1 Maintainer overview

The primary goal of b4 is to make it easier for maintainers to retrieve patch series, verify their authenticity, apply any follow-up code review trailers, and apply the patches to their maintained git trees.

This functionality works best when coupled with a public-inbox aggregator service, such as the one running on lore.kernel.org, but can be used with local mailboxes and maildirs, thus providing fully decentralized, experience with robust end-to-end attestation.

2.1.1 Working with patches sent to distribution lists

Patches sent to distribution lists remains the only widely used decentralized code review framework. RFC2822-conformant ("email") messages adhere to an established standard that ensures high level of interoperability between systems, and it remains one of the remaining few truly decentralized communication platforms.

Note, that "distribution lists" may not necessarily mean "patches sent via email". In addition to SMTP, RFC2822 messages can be also delivered via any number of push and pull mechanisms, such as NNTP, web archives, public-inbox repositories, etc. In the case of lore.kernel.org, the messages are collated from a large number of sources then replicated across multiple frontends.

2.2 mbox: retrieving threads

Note: If you are looking for a way to continuously retrieve full threads (or even full search results) from a public-inbox server, the `lei` tool provides a much more robust way of doing that.

Retrieving full discussion threads is the most basic use of b4. All you need to know is the message-id of any message in the thread:

```
b4 mbox 20200313231252.64999-1-keescook@chromium.org
```

Alternatively, if you have found a thread on lore.kernel.org and you want to retrieve it in full, you can just use the full URL:

```
b4 mbox https://lore.kernel.org/lkml/20200313231252.64999-1-keescook@chromium.org/#t
```

By default, b4 will save the thread in a mailbox format using the message-id of the message as the filename base:

```
$ b4 mbox 20200313231252.64999-1-keescook@chromium.org
Grabbing thread from lore.kernel.org/all/20200313231252.64999-1-keescook%40chromium.org/
↪t.mbox.gz
5 messages in the thread
Saved ./20200313231252.64999-1-keescook@chromium.org.mbx
```

2.2.1 Option flags

-m LOCALMBOX, --use-local-mbox LOCALMBOX
: By default, b4 will retrieve threads from remote public-inbox servers, but it can also use a local mailbox/maildir. This is useful if you have a tool like `mbsync` or `lei` copying remote messages locally and you need to do some work while offline. You can pass `-` to read messages from stdin.

--stdin-pipe-sep STDIN_PIPE_SEP (0.11+)
: When reading input from stdin, split messages using the string passed as parameter. Otherwise, b4 expects stdin to be a single message or a valid mbox.

  This is most useful when piping threads directly from mutt. In your `.muttrc` add the following configuration parameter:

  ```
  set pipe_sep = "\n---randomstr---\n"
  ```

  Then invoke b4 with `-m - --stdin-pipe-sep='\n---randomstr---\n'`

-C, --no-cache
: By default, b4 will cache the retrieved threads for about 10 minutes. This lets you force b4 to ignore cache and retrieve the latest results.

-o OUTDIR, --outdir OUTDIR
: Instead of writing the .mbox file to the current directory, write it to this location instead. You can also pass a path to an existing mbox or maildir location to have the results appended to that mailbox instead (see also the `-f` flag below).

-c, --check-newer-revisions
: When retrieving patch series, check if a newer revision is available. For example, if you are trying to retrieve a series titled `[PATCH v2 0/3]`, b4 will use a number of mechanisms to check if a `v3` or later revision is also available and will add these results to the retrieved thread.

-n WANTNAME, --mbox-name WANTNAME
: By default, the resulting mailbox file will use the message-id as the basis for its filename. This option lets you override this behaviour.

-M, --save-as-maildir
: By default, the retrieved thread will be saved as an mbox file. However, due to subtle incompatibilities between various mbox formats ("mboxo" vs "mboxrd", etc), you may want to instead save the results as a Maildir directory.

-f, --filter-dupes
: When adding messages to existing mailbox or maildir (with `-o`), this will check all existing messages and will only add those messages that aren't already present. Note, that this uses simple message-id matching and no other checks for correctness are performed.

-r MBOX, --refetch MBOX (v0.12+)
: This allows you to refetch all messages in the provided mailbox from the upstream public-inbox server. For example, this is useful when you have a .mbx file prepared by `b4 am` and you want to send a response to one of the patches. Performing a refetch will restore the original message headers that may have been dropped or modified by `b4 am`.

2.2.2 Using with mutt

If you are a mutt or neomutt user and your mail is stored locally, you can define a quick macro that would let you quickly retrieve full threads and add them to your inbox. This is handy if you are cc'd in the middle of a conversation and you want to retrieve the rest of the thread for context.

Add something like the following to your `~/.muttrc`:

```
macro index 4 "<pipe-message>b4 mbox -fo ~/Mail<return>"
```

Now selecting a message in the message index and pressing "4" will retrieve the rest of the thread from the public-inbox server and add them to the local maildir (`~/Mail` in the example above).

2.3 am,shazam: retrieving and applying patches

Most commonly, b4 is used to retrieve, prepare, and apply patches sent via distribution lists. The base functionality is similar to that of `b4 mbox`:

```
b4 am 20200313231252.64999-1-keescook@chromium.org
```

This will do the following:

1. look up if that message-id is known on the specified public-inbox server (e.g. lore.kernel.org)
2. retrieve the full thread containing that message-id
3. process all replies to collect code review trailers and apply them to the relevant patch commit messages
4. perform attestation checks on patches and code review follow-ups
5. put all patches in the correct order and prepare for "git am"
6. write out the resulting mailbox so it is ready to be applied to a git tree

For example:

```
$ b4 am 20200313231252.64999-1-keescook@chromium.org
Analyzing 5 messages in the thread
Checking attestation on all messages, may take a moment...
---
  ✓ [PATCH v2 1/2] selftests/harness: Move test child waiting logic
  ✓ [PATCH v2 2/2] selftests/harness: Handle timeouts cleanly
  ---
  ✓ Signed: DKIM/chromium.org
---
Total patches: 2
---
Cover: ./v2_20200313_keescook_selftests_harness_handle_timeouts_cleanly.cover
 Link: https://lore.kernel.org/r/20200313231252.64999-1-keescook@chromium.org
 Base: not specified
       git am ./v2_20200313_keescook_selftests_harness_handle_timeouts_cleanly.mbx
```

2.3.1 b4 am vs. b4 shazam

Note: `b4 shazam` was added in version v0.9.

The two commands are very similar – the main distinction is that `b4 am` will prepare the patch series for application to the git tree, but will not make any modifications to your current branch.

The `b4 shazam` command will do the same as `b4 am` *and* will apply the patch series to the current branch (if it is possible to do so cleanly).

2.3.2 Common flags

The following flags are common to both commands:

`-m LOCALMBOX, --use-local-mbox LOCALMBOX`
: By default, b4 will retrieve threads from remote public-inbox servers, but it can also use a local mailbox/maildir. This is useful if you have a tool like `mbsync` or `lei` copying remote messages locally and you need to do some work while offline. You can pass `-` to read messages from stdin.

`--stdin-pipe-sep STDIN_PIPE_SEP` (0.11+)
: When reading input from stdin, split messages using the string passed as parameter. Otherwise, b4 expects stdin to be a single message or a valid mbox.

  This is most useful when piping threads directly from mutt. In your `.muttrc` add the following configuration parameter:

```
set pipe_sep = "\n---randomstr---\n"
```

  Then invoke b4 with `-m - --stdin-pipe-sep='\n---randomstr---\n'`

`-C, --no-cache`
: By default, b4 will cache the retrieved threads for about 10 minutes. This lets you force b4 to ignore cache and retrieve the latest results.

`-v WANTVER, --use-version WANTVER`
: If a thread (or threads, when used with `-c`) contains multiple patch series revisions, b4 will automatically pick the highest numbered version. This switch lets you pick a different revision.

`-t, --apply-cover-trailers`
: By default, b4 will not apply any code review trailers sent to the cover letter (but will let you know when it finds those). This lets you automatically apply these trailers to all commits in the series. **This will become the default in a future version of b4.**

`-S, --sloppy-trailers`
: B4 tries to be careful when collecting code review trailers and will refuse to consider the trailers where the email address in the From: header does not patch the address in the trailer itself.

  For example, the following message will not be processed:

```
From: Alice Maintainer <alice@personalemail.org>
Subject: Re: [PATCH v3 3/3] Some patch title

> [...]
Reviewed-by: Alice Maintainer <alice.maintainer@workemail.com>
```

  In such situations, b4 will print a warning and refuse to apply the trailer due to the email address mismatch. You can override this by passing the `-S` flag.

-T, --no-add-trailers
This tells b4 to ignore any follow-up trailers and just save the patches as sent by the contributor.

-s, --add-my-sob
Applies your own `Signed-off-by:` trailer to every commit.

-l, --add-link
Adds a `Link:` trailer with the URL of the retrieved message using the `linkmask` template. Note, that such trailers may be considered redundant by the upstream maintainer.

-P CHERRYPICK, --cherry-pick CHERRYPICK
This allows you to select a subset of patches from a larger series. Here are a few examples.

This will pick patches 1, 3, 5, 6, 7, 9 and any others that follow:

```
b4 am -P 1,3,5-7,9- <msgid>
```

This will pick just the patch that matches the exact message-id provided:

```
b4 am -P _ <msgid>
```

This will pick all patches where the subject matches "iscsi":

```
b4 am -P *iscsi*
```

--cc-trailers
Copies all addresses found in the message Cc's into `Cc:` commit trailers.

--no-parent
Break thread at the msgid specified and ignore any parent messages. This is handy with very convoluted threads, for example when someone replies with a different patch series in the middle of a larger conversation and b4 gets confused about which patch series is being requested.

--allow-unicode-control-chars
There are some clever tricks that can be accomplished with unicode control chars that make the code as printed on the screen (and reviewed by a human) to actually do something totally different when processed by a compiler. Such unicode control chars are almost never legitimately useful in the code, so b4 will print a warning and bail out when it finds them. However, just in case there are legitimate reasons for these characters to be in the code (e.g. as part of documentation translated into LTR languages), this behaviour can be overridden.

2.3.3 Flags only valid for `b4 am`

The following flags only make sense for `b4 am`:

-o OUTDIR, --outdir OUTDIR
Instead of writing the .mbox file to the current directory, write it to this location instead. You can also pass a path to an existing mbox or maildir location to have the results appended to that mailbox instead (see also the `-f` flag below).

When `-` is specified, the output is dumped to stdout.

-c, --check-newer-revisions
When retrieving patch series, check if a newer revision is available. For example, if you are trying to retrieve a series titled `[PATCH v2 0/3]`, b4 will use a number of mechanisms to check if a `v3` or later revision is also available and will add these results to the retrieved thread.

-n WANTNAME, --mbox-name WANTNAME
By default, the resulting mailbox file will use the message-id as the basis for its filename. This option lets you override this behaviour.

-M, --save-as-maildir
By default, the retrieved thread will be saved as an mbox file. However, due to subtle incompatibilities between various mbox formats ("mboxo" vs "mboxrd", etc), you may want to instead save the results as a Maildir directory.

-Q, --quilt-ready
Saves the patches as a folder that can be fed directly to quilt. If you don't know what quilt is, you don't really need to worry about this option.

-b GUESSBRANCH [...], --guess-branch GUESSBRANCH [...]
When using `--guess-base`, you can restrict which branch(es) b4 will use to find the match. If not specified, b4 will use the entire tree history.

--guess-lookback GUESSDAYS
When using `--guess-base`, you can specify how far back b4 should look *from the date of the patch* to find the base commit. By default, b4 will only consider the last 14 days prior to the date of the patch, but you can expand or shrink it as necessary.

-3, --prep-3way
This will try to prepare your tree for a 3-way merge by doing some behind the scenes git magic and preparing some fake loose commits.

--no-cover
By default, b4 will save the cover letter as a separate file in the output directory specified. This flag turns it off (this is also the default when used with `-o -`).

--no-partial-reroll
For minor changes, it is common practice for contributors to send follow-ups to just the patches they have modified. For example:

```
[PATCH v1 1/3] foo: add foo to bar
[PATCH v1 2/3] bar: add bar to baz
 \- [PATCH v2 2/3] bar: add bar to baz
[PATCH v1 3/3] baz: add baz to quux
```

In this case, b4 will properly create a v2 of the entire series by reusing `[PATCH v1 1/3]` and `[PATCH v1 3/3]`. However, sometimes that is not the right thing to do, so you can turn off this feature using `--no-partial-reroll`.

2.3.4 Flags only valid for `b4 shazam`

By default, `b4 shazam` will apply the patch series directly to the git tree where the command is being executed. However, instead of just running `git am` and applying the patches directly on top of the current branch, it can also treat the series similar to a git pull request and either prepare a `FETCH_HEAD` that you can merge manually, or even automatically merge the series using the series cover letter as the basis for the merge commit.

-H, --make-fetch-head
This will prepare the series and place it into the `FETCH_HEAD` that can then be merged just as if it were a pull request:

1. b4 will prepare a temporary sparse worktree
2. b4 will apply the series to that worktree
3. if `git am` completed successfully, b4 will fetch that tree into your current tree's `FETCH_HEAD` (and get rid of the temporary tree)
4. b4 will place the cover letter into `.git/b4-cover`
5. b4 will offer the command you can run to merge the change into your current branch, e.g.:

```
git merge --no-ff -F .git/b4-cover --edit FETCH_HEAD --signoff
```

Generally, this command is also a good test for "will this patch series apply cleanly to my tree." You can perform any actions with the FETCH_HEAD as you normally would, e.g. run `git diff`, make a new branch out of it using `git checkout`, etc.

-M, --merge
Exactly the same as `--make-fetch-head`, but will actually execute the suggested `git merge` command.

Please also see the *shazam settings* section for some configuration file options that affect some of `b4 shazam` behaviour.

2.4 diff: comparing patch series

The `diff` subcommand allows comparing two different revisions of the same patch series using `git range-diff`. Note, that in order to perform the `range-diff` comparison, both revisions need to cleanly apply to the current tree, which may not always be easy to achieve.

The easiest way to use it is to prepare two mbox files of the series you would like to compare first:

```
$ b4 am --no-cover -n ver1 msgid-of-ver-1
$ b4 am --no-cover -n ver2 msgid-of-ver-2
$ b4 diff -m ver1.mbx ver2.mbx
```

2.4.1 Optional flags

-g GITDIR, --gitdir GITDIR
Specify a path to the git tree to use, if not running the command inside a git tree.

-C, --no-cache
By default, b4 will cache the retrieved threads for about 10 minutes. This lets you force b4 to ignore cache and retrieve the latest results.

-v WANTVERS [WANTVERS ...], --compare-versions WANTVERS [WANTVERS ...]
To properly work, this requires that both versions being compared are part of the same thread, which is rarely the case. In the future, this may work better as more series use the `change-id` trailer to keep track of revisions across discussion threads.

Example: `b4 diff <msgid> -v 2 3`

-n, --no-diff
By default, `b4 diff` will output the results of the range-diff command. However, this can be a wall of text, so instead you may want to just view the command that you can run yourself with the ranges prepared by b4. This additionally allows you to tweak the `git-range` flags to use.

-m AMBOX AMBOX, --compare-am-mboxes AMBOX AMBOX
Compares two mbox files prepared by `git am` instead of querying the public-inbox server directly.

-o OUTDIFF, --output-diff OUTDIFF
(DEPRECATED) Sends `range-diff` output into a file. You should use `-n` instead and redirect output from the actual `git range-diff` command.

-c, --color
(DEPRECATED) Show colour output even when outputting into a file. You should use `-n` instead and modify flags to `range-diff`.

2.5 pr: working with pull requests

In addition to working with patches and patch series, b4 is also able to work with pull requests. It provides the following benefits as opposed to using git directly:

- it can check if the pull request has already been applied before performing a git fetch
- it will check the signature on the tag (or tip commit)
- it can track applied pull requests and send replies to submitters (using `b4 ty`)
- it can explode a pull request into a series of patches for code review purposes

Basic usage is very similar to `b4 am`:

```
b4 pr <msgid>
```

By default, this will fetch the pull request into `FETCH_HEAD`.

2.5.1 Optional flags

`-g GITDIR, --gitdir GITDIR`
: This specifies (or overrides) the git directory where the pull request should be applied.

`-b BRANCH, --branch BRANCH`
: After fetching the pull request into `FETCH_HEAD`, check it out as a new branch with the name specified.

`-c, --check`
: Check if the specified pull request has already been applied.

2.5.2 Exploding pull requests

Pull requests are useful, but if the maintainer needs to do more than just accept or reject it, providing code review commentary on a PR can be difficult. For this reason, b4 can convert a pull request into a mailbox full of patches, as if the pull request was sent as a patch series. The exploded pull request will retain the correct author and To/Cc headers.

`-e, --explode`
: Instructs b4 to convert a pull request to a series of patches and save them as a mailbox file.

`-o OUTMBOX, --output-mbox OUTMBOX`
: If `-o` is not provided, the mailbox name will be based on the message-id of the pull request and saved in the local directory. This allows overriding that with a different path and name.

Explode archival features

Note: These are experimental features that were developed for internal kernel.org use.

The following flags are mostly useful when b4 is used for archival purposes. One of the goals of this feature was to make it possible to save pull requests, which are transient by nature, into an archival public-inbox so they can be analyzed by archivists at a later date if necessary.

`-l, --retrieve-links`
: Will attempt to retrieve URLs specified in Link: trailers, in case they are discussion threads. If successful, they will be compressed and attached to the cover letter as .mbx.gz files.

-f MAILFROM, --from-addr MAILFROM
(DEPRECATED) When exploding pull requests, use this email address in the From header, instead of reusing the same From as in the pull request.

-s SENDIDENTITY, --send-as-identity SENDIDENTITY
(DEPRECATED) When resending pull requests as patch series, use this sendemail identity.

--dry-run
(DEPRECATED) Force a –dry-run on `git-send-email` invocation.

2.6 ty: sending automated contributor feedback

B4 makes it easier to send automated developer feedback when you apply patches or pull requests to your git tree.

2.6.1 Tracking retrieved patches and PRs

Any patches or pull requests you retrieve with `b4 am,shazam,pr` will be automatically tracked by b4 in your homedir (usually, in `$HOME/.local/share/b4`, but may vary if your `$XDG_DATA_HOME` is set to a different value). There are four kinds of files in that directory:

- .am: contain information about patches retrieved with `b4 am` or `b4 shazam`
- .pr: contain information about pull requests retrieved with `b4 pr`
- .sent: either .am or .pr tracked files that were successfully sent using `b4 ty`
- .discarded: either .am or .pr tracked files that were deleted using `b4 ty`

All of these files contain JSON data about the series or pull requests being tracked.

2.6.2 Using the Auto-Thankanator

If you've retrieved and applied some patches to your tree, you should be able to fire up the "auto-thankanator", which uses patch-id and commit subject tracking to figure out which series from those you have retrieved were applied to your tree. The process is usually pretty fast and fairly accurate.

2.6.3 Manually listing and thanking

If you don't want to use the auto-thankanator, or if it's not finding a patch series (e.g. because you've made changes to a commit before applying it to your tree), you can use a more manual process. First, list all tracked series:

```
$ b4 ty -l
```

Identify the series that you're sure got applied, then generate the thank-you message:

```
$ b4 ty -t 1
```

This will write out a .thanks file in the current directory, which you can then modify and send out.

2.6.4 Sending out mail vs. writing .thanks files

By default, `b4 ty` will write out .thanks files in the current directory, which allows you to edit the body of the message before sending it out, e.g. using `mutt`:

```
$ mutt -f foo.thanks
```

However, if you have a configured `sendemail` section, you can also tell b4 to send out the thanks message directly:

```
$ b4 ty -aS --dry-run
```

The above command will locate all tracked series that got applied to your tree and show the messages that are going to be sent out. If you're happy with the results, you can omit the `--dry-run` switch to actually send the mail.

If you want `b4 ty` to always send mail, you can make the `-S` switch permanent by setting the `b4.ty-send-email` config variable to `yes`.

2.6.5 Editing the templates

You can edit the default templates that are provided with b4 to customize the thank-you message. Once you have your own versions, you can specify the path to the template to use via the `b4.thanks_am_template` and `b4.thanks_pr_template` configuration parameters. See *Thank-you (ty) settings* for details.

2.6.6 Optional flags

-g GITDIR, --gitdir GITDIR
: The git tree to use instead of the current working directory.

-o OUTDIR, --outdir OUTDIR
: Where to write the .thanks files if not into the current directory. Has no effect when `-S` is used.

-l, --list
: Lists all tracked patch series and pull requests.

-t THANKFOR, --thank-for THANKFOR
: From the listing generated by `--list`, specify which thank-you notes should be sent. This command accepts comma-separated values and ranges, including open-ended ranges, e.g.: `-t 1,3,5-7,9-`. It also accepts `all`.

-d DISCARD, --discard DISCARD
: From the listing generated by `--list`, specify which thank-you notes should be discarded. This command accepts comma-separated values and ranges, including open-ended ranges, e.g.: `-t 1,3,5-7,9-`. It also accepts `all`.

-a, --auto
: The auto-thankanator: uses patch-id and commit subject matching to figure out which tracked series or pull request have been applied to your tree.

-b BRANCH, --branch BRANCH
: When using `--auto`, specify which git branch should be used if not the currently active branch.

--since SINCE
: When using `--auto`, this lets you adjust how far back b4 will look to find your own commits. Takes the same format as `--since` flags passed to git, with the default of `1.week`.

-S, --send-email
: Instead of writing .thanks files, send the email directly. Requires that the `sendemail` section is present in your git configuration.

--dry-run
When used with -S, will not actually send email, just print them out to stdout.

--pw-set-state PW_SET_STATE
When patchwork integration is configured, sets the specified patchwork state instead of the default specified in config settings (use with -a, -t or -d). See *Patchwork integration settings* for more details.

2.7 kr: working with contributor keys

This subcommand allows maintaining a local keyring of contributor keys.

Note: This functionality is under active development and the set of available features will be expanded in the near future.

2.7.1 Patatt keyrings

B4 uses the patatt patch attestation library for its purposes, and it uses patatt-style keyrings. You can read more information about managing patatt keyrings at the following page:

- https://pypi.org/project/patatt/#getting-started-as-a-project-maintainer

2.7.2 b4 kr –show-keys

At this stage, b4 has limited support for keyring management, but there are plans to expand this functionality in one of the future versions. At most, you can view what keys were used to sign a set of patches in a thread, e.g.:

```
$ b4 kr --show-keys <msgid>
Grabbing thread from lore.kernel.org/all/<msgid>/t.mbox.gz
---
alice.developer@example.org: (unknown)
    keytype: ed25519
     pubkey: AbCdzUj91asvincQGOFx6+ZF5AoUuP9GdOtQChs7Mm0=
     krpath: ed25519/example.org/alice.developer/20211009
   fullpath: /home/user/.local/share/b4/keyring/ed25519/example.org/alice.developer/
↪20211009
---
For ed25519 keys:
    echo [pubkey] > [fullpath]
```

At this time, if you want to store this public key in your local keyring, you can run the command suggested above:

```
echo AbCdzUj91asvincQGOFx6+ZF5AoUuP9GdOtQChs7Mm0= > \
    /home/user/.local/share/b4/keyring/ed25519/example.org/alice.developer/20211009
```

Now if you come across a signed set of patches from alice.developer, you should be able to view the attestation status in the `b4 am` output.

CHAPTER

THREE

FOR DEVELOPERS

3.1 Contributor overview

Note: `b4 prep`, `b4 send` and `b4 trailers` are available starting with version 0.10.

Even though b4 started out as a tool to help maintainers, beginning with the version `0.10` there is also a set of features aimed at making it easier for contributors to submit patch series:

- `b4 prep` allows to get your patch series ready for sending to the maintainer for review
- `b4 send` simplifies the process of submitting your patches to the upstream maintainer even if you don't have access to a compliant SMTP server
- `b4 trailers` simplifies the process of retrieving code-review trailers received on the distribution lists and applying them to your tree

Warning: This is a very new set of features and should be considered experimental. While a lot of work has gone into making sure that your git tree is not harmed in any way, it is best to always have backups and to always check things with `--dry-run` when that option is available.

If you come across a bug or unexpected behaviour, please report the problem to the Tools mailing list.

3.1.1 Do I still need to be able to send email?

While `b4 send` makes it possible to submit patches without having access to an SMTP server, you still need a reasonable mail server for participating in conversations and code review.

The main benefit of `b4 send` is that you no longer have to really care if your mail server performs some kind of content mangling that causes patches to become corrupted, or if it doesn't provide a way to send mail via SMTP.

3.1.2 What is the b4 contributor workflow?

The workflow is very much git-oriented, so you should expect to need to know a lot about such git commands like `git amend` and `git rebase -i`. In general, the process goes like this:

1. Prepare your patch series by using `b4 prep` and queueing your commits. Use `git rebase -i` to arrange the commits in the right order and to write good commit messages.
2. Prepare your cover letter using `b4 prep --edit-cover`. You should provide a good overview of what your series does and why you think it will improve the current code.
3. When you are almost ready to send, use `b4 prep --auto-to-cc` to collect the relevant addresses from your commits. If your project uses a `MAINTAINERS` file, this will also perform the required query to figure out who should be included on your patch series submission.
4. Review the list of addresses that were added to the cover letter and, if you know what you're doing, remove any that you think are unnecessary.
5. Send your series using `b4 send`. This will automatically reroll your series to the next version and add changelog entries to the cover letter.
6. Await code review and feedback from maintainers.
7. Apply any received code-review trailers using `b4 trailers -u`.
8. Use `git rebase -i` to make any changes to the code based on the feedback you receive. Remember to record these changes in the cover letter's changelog.
9. Unless series is accepted upstream, GOTO 3.
10. Clean up obsolete prep-managed branches using `b4 prep --cleanup`

Please read the rest of these docs for details on the `prep`, `send`, and `trailers` subcommands.

3.2 prep: preparing your patch series

The first stage of contributor workflow is to prepare your patch series for submission upstream. It generally consists of the following stages:

1. start a new topical branch using `b4 prep -n topical-name`
2. add commits as usual and work with them using `git rebase -i`
3. prepare the cover letter using `b4 prep --edit-cover`
4. prepare the list of recipients using `b4 prep --auto-to-cc`

3.2.1 Starting a new topical branch

When you are ready to start working on a new submission, the first step is to create a topical branch:

```
b4 prep -n descriptive-name [-f tagname]
```

It is important to give your branch a short descriptive name, because it will become part of the unique `change-id` that will be used to track your proposal across revisions. In other words, don't call it "stuff" or "foo".

This command will do the following:

1. Create a new branch called `b4/descriptive-name` and switch to it.
2. Create an empty commit with a cover letter template.

Note: Generally, you will want to fork from some known point in the history, not from some random HEAD commit. You can use `-f` to specify a fork-point for b4 to use, such as a recent tag name.

You can then edit the cover letter using:

```
b4 prep --edit-cover
```

This will fire up a text editor using your defined `$EDITOR` or `core.editor` and automatically update the cover letter commit when you are done.

Cover letter strategies

By default, b4 will keep the cover letter in an empty commit at the start of your series. This has the following benefits:

- it is easy to keep track where your series starts without needing to keep a "tracking base branch" around
- you can view and edit the cover letter using regular git commands (`git log`, `git rebase -i`)
- you can push the entire branch to a remote and pull it from a different location to continue working on your series from a different system

However, keeping an empty commit in your history can have some disadvantages in some less-common situations:

- it complicates merging between branches
- some non-native git tools may drop empty commits
- editing the cover letter rewrites the commit history of the entire branch

For this reason, b4 supports alternative strategies for storing the cover letter, which can be set using the `b4.prep-cover-strategy` configuration variable.

`commit` strategy (default)
: This is the default strategy that keeps the cover letter and all tracking information in an empty commit at the start of your series. See above for upsides and downsides.

  This strategy is recommended for developers who mostly send out patch series and do not handle actual subsystem tree management (merging submissions from sub-maintainers, cherry-picking, etc).

`branch-description` strategy
: This keeps the cover letter and all tracking information outside of the git commits by using the branch description configuration value (stored locally in `.git/config`).

  Upsides:

  - this is how git expects you to handle cover letters (see `git format-patch --cover-from-description`)
  - editing the cover letter does not rewrite commit history
  - merging between branches is easiest

  Downsides:

  - the cover letter cannot be pushed to a remote and only exists local to your tree
  - you have to rely on the base branch for keeping track of where your series starts

`tip-commit` strategy
: This is similar to the default `commit` strategy, but instead of keeping the cover letter and all tracking information in an empty commit at the start of your series, it keeps it at the end ("tip") of your series.

Upsides:

- allows you to push the series to a remote and pull it from a different location to continue working on a series
- editing the cover letter does not rewrite commit history, which may be easier when working in teams

Downsides:

- adding new commits is a bit more complicated, because you have to immediately rebase them to be in front of the cover letter
- you have to rely on the base branch for keeping track of where your series starts

---

Note: At this time, you cannot easily switch from one strategy to the other once you have created the branch with `b4 prep -n`. This may be supported in the future.

Enrolling an existing branch

If you've already started working on a set of commits without first running `b4 prep -n`, you can enroll your existing branch to make it "prep-tracked."

For example, if you have a branch called `my-topical-branch` that was forked from `master`, you can enroll it with b4:

```
b4 prep -e master
```

Once that completes, you should be able to edit the cover letter and use all other b4 contributor-oriented commands.

Creating a branch from a sent series

If you have previously sent a patch series, you can create your new topical branch from that submission by passing the `--from-thread` parameter to `b4 prep -n`. All you need is the msgid of the series, e.g.:

```
b4 prep -n my-topical-branch -F some-msgid@localhost
```

If the series was submitted using `b4 send` it will even contain all the preserved tracking information, but it's not a requirement and should work reasonably well with most patch series.

3.2.2 Working with commits

All your commits in a prep-tracked branch are just regular git commits and you can work with them using any regular git tooling:

- you can rebase them on a different (or an updated) branch using `git rebase`
- you can amend (reword, split, squash, etc) commits interactively using `git rebase -i`; there are many excellent tutorials available online on how to use interactive rebase

Unless you are using a very old version of git, your empty cover letter commit should be preserved through all rebase operations.

Note: You can edit the cover letter using regular git operations, though it is not recommended (best to do it with `b4 prep --edit-cover`). If you do want to edit it directly using `git rebase -i`, remember to use `git commit --allow-empty` to commit it back into the tree.

What if I only have a single patch?

When you only have a single patch, the contents of the cover letter will be mixed into the "under-the-cut" portion of the patch. You can just use the cover letter for extra To/Cc trailers and changelog entries as your patch goes through revisions. If you add more commits in the future version, you can fill in the cover letter content with additional information about the intent of your entire series.

3.2.3 Prepare the list of recipients

When you are getting ready to submit your work, you need to figure out who the recipients of your series should be. By default, b4 will send the series to any address mentioned in the trailers (and to any other addresses you tell it to use).

For the Linux kernel, a required step is to gather the recipients from the output of `get_maintainer.pl`, which b4 will do for you automatically when you run the `auto-to-cc` command:

```
b4 prep --auto-to-cc
```

The recipients will be added to the cover letter as extra `To:` and `Cc:` trailers. It is normal for this list to be very large if your change is touching a lot of files. You can add or remove recipients by adding or removing the recipient trailers from the cover letter using `b4 prep --edit-cover`.

For projects that are not using the MAINTAINERS file, there is usually a single list where you should send your changes. You can set that in the repository's `.git/config` file as follows:

```
[b4]
  send-series-to = some@list.name
```

This may also be already set by the project, if they have a `.b4-config` file in the root of their git repository.

3.2.4 Cleaning up old work

Once your series is accepted upstream, you can archive and clean up the prep-managed branch and all its sent tags:

```
b4 prep --cleanup
```

This will list all prep-managed branches in your repository. Pick a branch to clean up (make sure it's not currently checked out), and run the command again:

```
b4 prep --cleanup b4/my-topical-branch
```

After you confirm your action, this will create a tarball with all the patches, cover letters, and tracking information from your series, after which the branch and related tags will be deleted from your local repository.

3.2.5 Prep command flags

Please also see *Contributor-oriented settings*, which allow setting or modifying defaults for some of these flags.

`-c, --auto-to-cc`
: Automatically populate the cover letter with addresses collected from commit trailers. If a `MAINTAINERS` file is found, together with `scripts/get_maintainer.pl`, b4 will automatically perform the query to collect the maintainers and lists that should be notified of the change.

`-p OUTPUT_DIR, --format-patch OUTPUT_DIR`
: This will output your tracked series as patches similar to what `git-format-patch` would do.

`--edit-cover`
: Lets you edit the cover letter using whatever editor is defined in git-config for `core.editor`, `$EDITOR` if that is not found, or `vim` because we're pretty sure that if you don't like vim, you would have already set your `$EDITOR` to not be vim.

`--show-revision`
: Shows the current series revision.

`--force-revision N`
: Forces the revision to a different integer number. This modifies your cover letter and tracking information and makes this change permanent.

`--compare-to vN` **(v0.11+)**
: This executes a `git range-diff` command that lets you compare the previously sent version of the series to what is currently in your working branch. This is very useful right before sending off a new revision to make sure that you didn't forget to include anything into changelogs.

`--manual-reroll MSGID`
: Normally, your patch series will be automatically rerolled to the next version after a successful `b4 send` (see *send: sending in your work*). However, if you sent it in using some other mechanism, such as `git-send-email`, you can trigger a manual reroll using this command. It requires a message-id that can be retrieved from the public-inbox server, so we can properly add the reference to the previously sent series to the cover letter changelog.

`--set-prefixes PREFIX [PREFIX ...]` **(v0.11+)**
: If you want to mark your patch as `RFC`, `WIP`, or add any other subsystem identifiers, you can define them via this command. Do **not** add `PATCH` or `v1` here, as these will already be automatically added to the subject lines. To remove any extra prefixes you previously set, you can run `--set-prefixes ''`.

  Alternatively, you can add any extra prefixes to the cover letter subject line, using the usual square brackets notation, e.g.:

```
[RFC] Cover letter subject
```

  When b4 sends the message, it will be expanded with the usual `PATCH`, `vN`, etc.

`--show-info [PARAM]` **(v0.13+)**
: Dumps information about the current series that can be parsed by other tools. Starting with v0.13, he parameter can be one of the following:

  - **keyname** to show just a specific value from the current branch
  - **branchname** to show all info about a specific branch
  - **branchname:keyname** to show a specific value from a specific branch

  For example, if you have a branch called `b4/foodrv-bar` and you want to display the `series-range` value, run:

```
b4 prep --show-info b4/foodrv-bar:series-range
```

  Or, to show all values for branch `b4/foodrv-bar`:

```
b4 prep --show-info b4/foodrv-bar
```

  Or, to show `series-range` for the current branch:

```
b4 prep --show-info series-range
```

  And, to show all values for the current branch:

```
b4 prep --show-info
```

--cleanup [BRANCHNAME] (v0.13+)
Archive and delete obsolete prep-managed branches and all git objects related to them (such as sent tags). Run without parameters to list all known prep-managed branches in the repository. Rerun with the branch name to create an archival tarball with all patches, covers, and tracking information, and then delete all git objects related to that series from the local repository.

-n NEW_SERIES_NAME, --new NEW_SERIES_NAME
Creates a new branch to start work on a new patch series.

-f FORK_POINT, --fork-point FORK_POINT
When creating a new branch, use a specific fork-point instead of whatever commit happens to be at the current `HEAD`.

-F MSGID, --from-thread MSGID
After creating a new branch, populate it with patches from this pre-existing patch series. Requires a message-id that can be retrieved from the public-inbox server.

-e ENROLL_BASE, --enroll ENROLL_BASE
Enrolls your current branch to be b4-prep managed. Requires the name of the branch to use as the fork-point tracking base.

3.3 send: sending in your work

B4 supports sending your series either via your own SMTP server, or via a web submission endpoint.

Upsides of using your own SMTP server:

- it is part of decentralized infrastructure not dependent on a single point of failure
- it adds domain-level attestation to your messages via DKIM signatures
- it avoids the need to munge the From: headers in patches, which is required for email delivery that originates at a different domain

However, using your own SMTP server may not always be a valid option:

- your mail provider may not offer an SMTP compliant server for sending mail (e.g. if it only uses a web-mail/exchange client)
- there may be limits on the number of messages you can send through your SMTP server in a short period of time (which is normal for large patch series)
- your company SMTP server may modify the message bodies by adding huge legal disclaimers to all outgoing mail

The web submission endpoint helps with such cases, plus offers several other upsides:

- the messages are written to a public-inbox feed, which is then immediately available for others to follow and query
- all patches are end-to-end attested with the developer signature
- messages are less likely to get lost or delayed

Note: Even if you opt to use the web submission endpoint, you still need a valid email account for participating in decentralized development – you will need it to take part in discussions and for sending and receiving code review

feedback.

3.3.1 Authenticating with the web submission endpoint

Before you start, you will need to configure your attestation mechanism. If you already have a PGP key configured for use with git, you can just use that and skip the next section. If you don't already have a PGP key, you can create a separate ed25519 key just for web submission purposes.

Creating a new ed25519 key

Note: Creating a new ed25519 key is not required if you already have a PGP key configured with git using the `user.signingKey` git-config setting.

Installing b4 should have already pulled in the patatt patch attestation library. You can use the command line tool to create your ed25519 key:

```
$ patatt genkey
Generating a new ed25519 keypair
Wrote: /home/user/.local/share/patatt/private/20220915.key
Wrote: /home/user/.local/share/patatt/public/20220915.pub
Wrote: /home/user/.local/share/patatt/public/ed25519/example.org/alice.developer/20220915
Add the following to your .git/config (or global ~/.gitconfig):
---
[patatt]
    signingkey = ed25519:20220915
    selector = 20220915
---
Next, communicate the contents of the following file to the
repository keyring maintainers for inclusion into the project:
/home/user/.local/share/patatt/public/20220915.pub
```

Copy the `[patatt]` section and add it to your `~/.gitconfig` or to your `.git/config` in the repository that you want to enable for `b4 send`.

Configuring the web endpoint

The web endpoint you will use is going to be dependent on the project. For the Linux kernel and associated tools (like Git, B4, patatt, etc), the kernel.org endpoint can be enabled by adding the following to your `~/.gitconfig`:

```
[b4]
  send-endpoint-web = https://lkml.kernel.org/_b4_submit
```

Note: The kernel.org endpoint can only be used for kernel.org-hosted projects. If there are no recognized mailing lists in the to/cc headers, then the submission will be rejected.

Once that is added, you can request authentication, as in the example below:

```
$ b4 send --web-auth-new
Will submit a new email authorization request to:
  Endpoint: https://lkml.kernel.org/_b4_submit
      Name: Alice Developer
  Identity: alice.developer@example.org
  Selector: 20220915
    Pubkey: ed25519:ABCDE1lNXHvHOTuHV+Cf1eK9SuRNZZYrQmcJ44IkE8Q=
---
Press Enter to confirm or Ctrl-C to abort
Submitting new auth request to https://lkml.kernel.org/_b4_submit
---
Challenge generated and sent to alice.developer@example.org
Once you receive it, run b4 send --web-auth-verify [challenge-string]
```

As the instructions say, you should receive a verification email to the address you specified in your `user.email`. Once you have received it, run the verification command by copy-pasting the UUID from the confirmation message:

```
$ b4 send --web-auth-verify abcd9b34-2ecf-4d25-946a-0631c414227e
Signing challenge
Submitting verification to https://lkml.kernel.org/_b4_submit
---
Challenge successfully verified for alice.developer@example.org
You may now use this endpoint for submitting patches.
```

You should now be able to send patches via this web submission endpoint.

3.3.2 Using your own SMTP server

If there is a `sendmail` section in your git configuration, B4 will try use that by default instead of the web endpoint. Only the most common subset of options are supported. The vast majority of servers will only need the following settings:

```
[sendemail]
   smtpServer = smtp.example.org
   smtpServerPort = 465
   smtpEncryption = ssl
   smtpUser = alice.developer@example.org
   smtpPass = [omitted]
```

You can also set up msmtp or a similar tool and specify the path to the `sendmail`-compliant binary as the value for `smtpServer`. You can force B4 to use the web endpoint by using the `--use-web-endpoint` argument.

3.3.3 Sending your patches

Once your web endpoint or SMTP server are configured, you can start sending your work.

Note: At this time, only series prepared with `b4 prep` are supported, but future versions may support sending arbitrary patches generated with `git format-patch`.

Checking things over with `-o`

It is a good idea to first check that everything is looking good by running the send command with `-o somedir`, e.g.:

```
b4 send -o /tmp/presend
```

This will write out the messages just as they would be sent out, giving you a way to check that everything is looking as it should.

Checking things over with `--reflect`

One final test you can do before you submit your series is to send everything to yourself. This is especially useful when using the web endpoint, because this allows you to see what the messages will look like after being potentially post-processed on the remote end.

When `--reflect` is on:

- b4 will still populate the To:/Cc: headers with all the addresses, because this allows to check for any encoding problems
- b4 will **only send the series to the address in the From: field**
- when using the web endpoint, the messages will not be added to the public-inbox feed
- your branch will **not** be automatically rerolled to the next revision

3.3.4 What happens after you send

The following happens after you send your patches:

- b4 will automatically create a detached head containing the commits from your sent series and tag it with the contents of the cover letter; this creates a historical record of your submission, as well as adds a way to easily resend a previously sent series
- b4 will reroll your series to the next version, so that if you just sent off a `v1` of the series, the working version will be marked as `v2`
- b4 will automatically edit the cover letter to add templated changelog entries containing a pre-populated link to the just-sent series

Resending your series

If something went wrong, or if you need to resend the series because nobody paid attention to it the first time, it is easy to do this with `--resend vN`. B4 will automatically generate the series from the tagged historical version created during the previous sending attempt.

3.3.5 Command line flags

-d, --dry-run
Don't send any mail, just output the raw messages that would be sent. Normally, this is a wall of text, so you'd want to use `-o` instead.

-o OUTPUT_DIR, --output-dir OUTPUT_DIR
Prepares everything for sending, but writes out the messages into the folder specified instead. This is usually a good last check before actually sending things out and lets you verify that all patches are looking good and all recipients are correctly set.

--reflect (v0.11+)
Prepares everything for sending, but only emails yourself (the address in the `From:` header). Useful as a last check to make sure that everything is looking good, and especially useful when using the web endpoint, because it may rewrite your From: header for DMARC reasons.

--no-trailer-to-cc
Do not add any addresses found in the cover or patch trailers to To: or Cc:. This is usually handy for testing purposes, in case you want to send a set of patches to a test address (also see `--reflect`).

--to
Add any more email addresses to include into the To: header here (comma-separated). Can be set in the configuration file using the `b4.send-series-to` option (see *Contributor-oriented settings*).

--cc
Add any more email addresses to include into the Cc: header here (comma-separated). Can be set in the configuration file using the `b4.send-series-cc` option (see *Contributor-oriented settings*).

--not-me-too
Removes your own email address from the recipients.

--no-sign
Don't sign your patches with your configured attestation mechanism. Note, that patch signing is required for the web submission endpoint, so this is only a valid option to use with `-o` or when using your own SMTP server. This can be set in the configuration using the `b4.send-no-patatt-sign` (see *Contributor-oriented settings*).

--resend V
Resend a previously sent version (see above for more info).

3.4 trailers: retrieving code-review trailers

This commands allows you to easily retrieve code-review trailers sent in reply to your work and apply them to the matching commits. It should locate code-review trailers sent in response to any previously submitted versions of your series, as long as:

- either the patch-id of the commit still matches what was sent, or
- the title of the commit is exactly the same

You can always edit the trailers after they are applied by using `git rebase -i` and choosing `reword` as rebase action.

Most commonly, you just need to run:

```
b4 trailers -u
```

3.4.1 Command flags

`-u, --update`
: Update branch commits with latest received trailers.

`-S, --sloppy-trailers`
: Accept trailers where the email address of the sender differs from the email address found in the trailer itself.

`-F MSGID, --trailers-from MSGID`
: Look for trailer updates in an arbitrary tread found on the public-inbox server. Note, that this is generally only useful in the following two cases:

  - for branches not already managed by `b4 prep`
  - when a single larger series is broken up into multiple smaller series (or vice-versa)

`--since SINCE`
: Only useful with `-F`. By default, b4 will only look for your own commits as far as 1 month ago. With this flag, you can instruct it to look further back.

CHAPTER

FOUR

GETTING HELP

To report a problem or suggest a feature, please send plaintext email to tools@linux.kernel.org.