Rocky Enterprise Linux 9.2 Manual Pages on command 'git-format-patch.1'

C:\>man git-format-patch.1

GIT-FORMAT-PATCH(1) Git Manual GIT-FORMAT-PATCH(1)

NAME

git-format-patch - Prepare patches for e-mail submission

SYNOPSIS

git format-patch [-k] [(-o|--output-directory) <dir> | --stdout]

[--no-thread | --thread[=<style>]]

[(--attach|--inline)[=<boundary>] | --no-attach]

[-s | --signoff]

[--signature=<signature> | --no-signature]

[--signature-file=<file>]

[-n | --numbered | -N | --no-numbered]

[--start-number <n>] [--numbered-files]

[--in-reply-to=<message id>] [--suffix=.<sfx>]

[--ignore-if-in-upstream]

[--cover-from-description=<mode>]

[--rfc] [--subject-prefix=<subject prefix>]

[(--reroll-count|-v) <n>]

[--to=<email>] [--cc=<email>]

[--[no-]cover-letter] [--quiet]

[--no-notes | --notes[=<ref>]]

[--interdiff=<previous>]

[--range-diff=<previous> [--creation-factor=<percent>]]

[--progress]

[<common diff options>]

[<since> | <revision range>]

DESCRIPTION

Prepare each commit with its patch in one file per commit, formatted to resemble UNIX mailbox format. The output of this command is convenient for e-mail submission or for use with git am.

There are two ways to specify which commits to operate on.

1. A single commit, <since>, specifies that the commits leading to the tip of the current branch that are not in the history that leads to the <since> to be output.
2. Generic <revision range> expression (see "SPECIFYING REVISIONS" section in gitrevisions(7)) means the commits in the specified range.

The first rule takes precedence in the case of a single <commit>. To apply the second rule, i.e., format everything since the beginning of history up until <commit>, use the --root option: git format-patch --root <commit>. If you want to format only <commit> itself, you can do this with git format-patch -1 <commit>.

By default, each output file is numbered sequentially from 1, and uses the first line of the commit message (massaged for pathname safety) as the filename. With the --numbered-files option, the output file names will only be numbers, without the first line of the commit appended. The names of the output files are printed to standard output, unless the --stdout option is specified.

If -o is specified, output files are created in <dir>. Otherwise they are created in the current working directory. The default path can be set with the format.outputDirectory configuration option. The -o option takes precedence over format.outputDirectory. To store patches in the current working directory even when format.outputDirectory points elsewhere, use -o .. All directory components will be created.

By default, the subject of a single patch is "[PATCH] " followed by the concatenation of lines from the commit message up to the first blank line (see the DISCUSSION section of git-commit(1)).

When multiple patches are output, the subject prefix will instead be "[PATCH n/m] ". To force 1/1 to be added for a single patch, use -n. To omit patch numbers from

the subject, use -N.

If given --thread, git-format-patch will generate In-Reply-To and References headers to make the second and subsequent patch mails appear as replies to the first mail; this also generates a Message-Id header to reference.

OPTIONS

-p, --no-stat

Generate plain patches without any diffstats.

-U<n>, --unified=<n>

Generate diffs with <n> lines of context instead of the usual three. Implies --patch.

--output=<file>

Output to a specific file instead of stdout.

--output-indicator-new=<char>, --output-indicator-old=<char>, --output-indicator-context=<char>

Specify the character used to indicate new, old or context lines in the generated patch. Normally they are +, - and ' ' respectively.

--indent-heuristic

Enable the heuristic that shifts diff hunk boundaries to make patches easier to read. This is the default.

--no-indent-heuristic

Disable the indent heuristic.

--minimal

Spend extra time to make sure the smallest possible diff is produced.

--patience

Generate a diff using the "patience diff" algorithm.

--histogram

Generate a diff using the "histogram diff" algorithm.

--anchored=<text>

Generate a diff using the "anchored diff" algorithm.

This option may be specified more than once.

If a line exists in both the source and destination, exists only once, and starts with this text, this algorithm attempts to prevent it from appearing as a deletion or addition in the output. It uses the "patience diff" algorithm

internally.

--diff-algorithm={patience|minimal|histogram|myers}

Choose a diff algorithm. The variants are as follows:

default, myers

The basic greedy diff algorithm. Currently, this is the default.

minimal

Spend extra time to make sure the smallest possible diff is produced.

patience

Use "patience diff" algorithm when generating patches.

histogram

This algorithm extends the patience algorithm to "support low-occurrence common elements".

For instance, if you configured the diff.algorithm variable to a non-default value and want to use the default one, then you have to use --diff-algorithm=default option.

--stat[=<width>[,<name-width>[,<count>]]]

Generate a diffstat. By default, as much space as necessary will be used for the filename part, and the rest for the graph part. Maximum width defaults to terminal width, or 80 columns if not connected to a terminal, and can be overridden by <width>. The width of the filename part can be limited by giving another width <name-width> after a comma. The width of the graph part can be limited by using --stat-graph-width=<width> (affects all commands generating a stat graph) or by setting diff.statGraphWidth=<width> (does not affect git format-patch). By giving a third parameter <count>, you can limit the output to the first <count> lines, followed by ... if there are more.

These parameters can also be set individually with --stat-width=<width>, --stat-name-width=<name-width> and --stat-count=<count>.

--compact-summary

Output a condensed summary of extended header information such as file creations or deletions ("new" or "gone", optionally "+l" if it?s a symlink) and mode changes ("+x" or "-x" for adding or removing executable bit respectively) in diffstat. The information is put between the filename part and the graph part. Implies --stat.

--numstat

Similar to --stat, but shows number of added and deleted lines in decimal notation and pathname without abbreviation, to make it more machine friendly. For binary files, outputs two - instead of saying 0 0.

--shortstat

Output only the last line of the --stat format containing total number of modified files, as well as number of added and deleted lines.

-X[<param1,param2,...>], --dirstat[=<param1,param2,...>]

Output the distribution of relative amount of changes for each sub-directory. The behavior of --dirstat can be customized by passing it a comma separated list of parameters. The defaults are controlled by the diff.dirstat configuration variable (see git-config(1)). The following parameters are available:

changes

Compute the dirstat numbers by counting the lines that have been removed from the source, or added to the destination. This ignores the amount of pure code movements within a file. In other words, rearranging lines in a file is not counted as much as other changes. This is the default behavior when no parameter is given.

lines

Compute the dirstat numbers by doing the regular line-based diff analysis, and summing the removed/added line counts. (For binary files, count 64-byte chunks instead, since binary files have no natural concept of lines). This is a more expensive --dirstat behavior than the changes behavior, but it does count rearranged lines within a file as much as other changes. The resulting output is consistent with what you get from the other --*stat options.

files

Compute the dirstat numbers by counting the number of files changed. Each changed file counts equally in the dirstat analysis. This is the computationally cheapest --dirstat behavior, since it does not have to look at the file contents at all.

cumulative

Count changes in a child directory for the parent directory as well. Note that when using cumulative, the sum of the percentages reported may exceed 100%. The default (non-cumulative) behavior can be specified with the noncumulative parameter.

<limit>

An integer parameter specifies a cut-off percent (3% by default). Directories contributing less than this percentage of the changes are not shown in the output.

Example: The following will count changed files, while ignoring directories with less than 10% of the total amount of changed files, and accumulating child directory counts in the parent directories: --dirstat=files,10,cumulative.

--cumulative

Synonym for --dirstat=cumulative

--dirstat-by-file[=<param1,param2>...]

Synonym for --dirstat=files,param1,param2...

--summary

Output a condensed summary of extended header information such as creations, renames and mode changes.

--no-renames

Turn off rename detection, even when the configuration file gives the default to do so.

--[no-]rename-empty

Whether to use empty blobs as rename source.

--full-index

Instead of the first handful of characters, show the full pre- and post-image blob object names on the "index" line when generating patch format output.

--binary

In addition to --full-index, output a binary diff that can be applied with git-apply. Implies --patch.

--abbrev[=<n>]

Instead of showing the full 40-byte hexadecimal object name in diff-raw format output and diff-tree header lines, show only a partial prefix. This is independent of the --full-index option above, which controls the diff-patch

output format. Non default number of digits can be specified with --abbrev=<n>.

-B[<n>][/<m>], --break-rewrites[=[<n>][/<m>]]

Break complete rewrite changes into pairs of delete and create. This serves two purposes:

It affects the way a change that amounts to a total rewrite of a file not as a series of deletion and insertion mixed together with a very few lines that happen to match textually as the context, but as a single deletion of everything old followed by a single insertion of everything new, and the number m controls this aspect of the -B option (defaults to 60%). -B/70% specifies that less than 30% of the original should remain in the result for Git to consider it a total rewrite (i.e. otherwise the resulting patch will be a series of deletion and insertion mixed together with context lines).

When used with -M, a totally-rewritten file is also considered as the source of a rename (usually -M only considers a file that disappeared as the source of a rename), and the number n controls this aspect of the -B option (defaults to 50%). -B20% specifies that a change with addition and deletion compared to 20% or more of the file?s size are eligible for being picked up as a possible source of a rename to another file.

-M[<n>], --find-renames[=<n>]

Detect renames. If n is specified, it is a threshold on the similarity index (i.e. amount of addition/deletions compared to the file?s size). For example, -M90% means Git should consider a delete/add pair to be a rename if more than 90% of the file hasn?t changed. Without a % sign, the number is to be read as a fraction, with a decimal point before it. I.e., -M5 becomes 0.5, and is thus the same as -M50%. Similarly, -M05 is the same as -M5%. To limit detection to exact renames, use -M100%. The default similarity index is 50%.

-C[<n>], --find-copies[=<n>]

Detect copies as well as renames. See also --find-copies-harder. If n is specified, it has the same meaning as for -M<n>.

--find-copies-harder

For performance reasons, by default, -C option finds copies only if the original file of the copy was modified in the same changeset. This flag makes the command inspect unmodified files as candidates for the source of copy. This

is a very expensive operation for large projects, so use it with caution.

Giving more than one -C option has the same effect.

-D, --irreversible-delete

Omit the preimage for deletes, i.e. print only the header but not the diff between the preimage and /dev/null. The resulting patch is not meant to be applied with patch or git apply; this is solely for people who want to just concentrate on reviewing the text after the change. In addition, the output obviously lacks enough information to apply such a patch in reverse, even manually, hence the name of the option.

When used together with -B, omit also the preimage in the deletion part of a delete/create pair.

-l<num>

The -M and -C options require O(n^2) processing time where n is the number of potential rename/copy targets. This option prevents rename/copy detection from running if the number of rename/copy targets exceeds the specified number.

-O<orderfile>

Control the order in which files appear in the output. This overrides the diff.orderFile configuration variable (see git-config(1)). To cancel diff.orderFile, use -O/dev/null.

The output order is determined by the order of glob patterns in <orderfile>.

All files with pathnames that match the first pattern are output first, all files with pathnames that match the second pattern (but not the first) are output next, and so on. All files with pathnames that do not match any pattern are output last, as if there was an implicit match-all pattern at the end of the file. If multiple pathnames have the same rank (they match the same pattern but no earlier patterns), their output order relative to each other is the normal order.

<orderfile> is parsed as follows:

? Blank lines are ignored, so they can be used as separators for readability.

? Lines starting with a hash ("#") are ignored, so they can be used for comments. Add a backslash ("\") to the beginning of the pattern if it starts with a hash.

? Each other line contains a single pattern.

Patterns have the same syntax and semantics as patterns used for fnmatch(3) without the FNM_PATHNAME flag, except a pathname also matches a pattern if removing any number of the final pathname components matches the pattern. For example, the pattern "foo*bar" matches "fooasdfbar" and "foo/bar/baz/asdf" but not "foobarx".

-a, --text

Treat all files as text.

--ignore-cr-at-eol

Ignore carriage-return at the end of line when doing a comparison.

--ignore-space-at-eol

Ignore changes in whitespace at EOL.

-b, --ignore-space-change

Ignore changes in amount of whitespace. This ignores whitespace at line end, and considers all other sequences of one or more whitespace characters to be equivalent.

-w, --ignore-all-space

Ignore whitespace when comparing lines. This ignores differences even if one line has whitespace where the other line has none.

--ignore-blank-lines

Ignore changes whose lines are all blank.

--inter-hunk-context=<lines>

Show the context between diff hunks, up to the specified number of lines, thereby fusing hunks that are close to each other. Defaults to diff.interHunkContext or 0 if the config option is unset.

-W, --function-context

Show whole surrounding functions of changes.

--ext-diff

Allow an external diff helper to be executed. If you set an external diff driver with gitattributes(5), you need to use this option with git-log(1) and friends.

--no-ext-diff

Disallow external diff drivers.

--textconv, --no-textconv

Allow (or disallow) external text conversion filters to be run when comparing binary files. See gitattributes(5) for details. Because textconv filters are typically a one-way conversion, the resulting diff is suitable for human consumption, but cannot be applied. For this reason, textconv filters are enabled by default only for git-diff(1) and git-log(1), but not for git-format-patch(1) or diff plumbing commands.

--ignore-submodules[=<when>]

Ignore changes to submodules in the diff generation. <when> can be either "none", "untracked", "dirty" or "all", which is the default. Using "none" will consider the submodule modified when it either contains untracked or modified files or its HEAD differs from the commit recorded in the superproject and can be used to override any settings of the ignore option in git-config(1) or gitmodules(5). When "untracked" is used submodules are not considered dirty when they only contain untracked content (but they are still scanned for modified content). Using "dirty" ignores all changes to the work tree of submodules, only changes to the commits stored in the superproject are shown (this was the behavior until 1.7.0). Using "all" hides all changes to submodules.

--src-prefix=<prefix>

Show the given source prefix instead of "a/".

--dst-prefix=<prefix>

Show the given destination prefix instead of "b/".

--no-prefix

Do not show any source or destination prefix.

--line-prefix=<prefix>

Prepend an additional prefix to every line of output.

--ita-invisible-in-index

By default entries added by "git add -N" appear as an existing empty file in "git diff" and a new file in "git diff --cached". This option makes the entry appear as a new file in "git diff" and non-existent in "git diff --cached". This option could be reverted with --ita-visible-in-index. Both options are experimental and could be removed in future.

For more detailed explanation on these common options, see also gitdiffcore(7).

-<n>

Prepare patches from the topmost <n> commits.

-o <dir>, --output-directory <dir>

Use <dir> to store the resulting files, instead of the current working directory.

-n, --numbered

Name output in [PATCH n/m] format, even with a single patch.

-N, --no-numbered

Name output in [PATCH] format.

--start-number <n>

Start numbering the patches at <n> instead of 1.

--numbered-files

Output file names will be a simple number sequence without the default first line of the commit appended.

-k, --keep-subject

Do not strip/add [PATCH] from the first line of the commit log message.

-s, --signoff

Add Signed-off-by: line to the commit message, using the committer identity of yourself. See the signoff option in git-commit(1) for more information.

--stdout

Print all commits to the standard output in mbox format, instead of creating a file for each one.

--attach[=<boundary>]

Create multipart/mixed attachment, the first part of which is the commit message and the patch itself in the second part, with Content-Disposition: attachment.

--no-attach

Disable the creation of an attachment, overriding the configuration setting.

--inline[=<boundary>]

Create multipart/mixed attachment, the first part of which is the commit message and the patch itself in the second part, with Content-Disposition: inline.

--thread[=<style>], --no-thread

Controls addition of In-Reply-To and References headers to make the second and subsequent mails appear as replies to the first. Also controls generation of the Message-Id header to reference.

The optional <style> argument can be either shallow or deep. shallow threading makes every mail a reply to the head of the series, where the head is chosen from the cover letter, the --in-reply-to, and the first patch mail, in this order. deep threading makes every mail a reply to the previous one.

The default is --no-thread, unless the format.thread configuration is set. If --thread is specified without a style, it defaults to the style specified by format.thread if any, or else shallow.

Beware that the default for git send-email is to thread emails itself. If you want git format-patch to take care of threading, you will want to ensure that threading is disabled for git send-email.

--in-reply-to=<message id>

Make the first mail (or all the mails with --no-thread) appear as a reply to the given <message id>, which avoids breaking threads to provide a new patch series.

--ignore-if-in-upstream

Do not include a patch that matches a commit in <until>..<since>. This will examine all patches reachable from <since> but not from <until> and compare them with the patches being generated, and any patch that matches is ignored.

--cover-from-description=<mode>

Controls which parts of the cover letter will be automatically populated using the branch?s description.

If <mode> is message or default, the cover letter subject will be populated with placeholder text. The body of the cover letter will be populated with the branch?s description. This is the default mode when no configuration nor command line option is specified.

If <mode> is subject, the first paragraph of the branch description will populate the cover letter subject. The remainder of the description will populate the body of the cover letter.

If <mode> is auto, if the first paragraph of the branch description is greater than 100 bytes, then the mode will be message, otherwise subject will be used.

If <mode> is none, both the cover letter subject and body will be populated with placeholder text.

--subject-prefix=<subject prefix>

Instead of the standard [PATCH] prefix in the subject line, instead use [<subject prefix>]. This allows for useful naming of a patch series, and can be combined with the --numbered option.

--rfc

Alias for --subject-prefix="RFC PATCH". RFC means "Request For Comments"; use this when sending an experimental patch for discussion rather than application.

-v <n>, --reroll-count=<n>

Mark the series as the <n>-th iteration of the topic. The output filenames have v<n> prepended to them, and the subject prefix ("PATCH" by default, but configurable via the --subject-prefix option) has ` v<n>` appended to it. E.g. --reroll-count=4 may produce v4-0001-add-makefile.patch file that has "Subject: [PATCH v4 1/20] Add makefile" in it.

--to=<email>

Add a To: header to the email headers. This is in addition to any configured headers, and may be used multiple times. The negated form --no-to discards all To: headers added so far (from config or command line).

--cc=<email>

Add a Cc: header to the email headers. This is in addition to any configured headers, and may be used multiple times. The negated form --no-cc discards all Cc: headers added so far (from config or command line).

--from, --from=<ident>

Use ident in the From: header of each commit email. If the author ident of the commit is not textually identical to the provided ident, place a From: header in the body of the message with the original author. If no ident is given, use the committer ident.

Note that this option is only useful if you are actually sending the emails and want to identify yourself as the sender, but retain the original author (and git am will correctly pick up the in-body header). Note also that git send-email already handles this transformation for you, and this option should not be used if you are feeding the result to git send-email.

--add-header=<header>

Add an arbitrary header to the email headers. This is in addition to any configured headers, and may be used multiple times. For example, --add-header="Organization: git-foo". The negated form --no-add-header discards all (To:, Cc:, and custom) headers added so far from config or command line.

--[no-]cover-letter

In addition to the patches, generate a cover letter file containing the branch description, shortlog and the overall diffstat. You can fill in a description in the file before sending it out.

--interdiff=<previous>

As a reviewer aid, insert an interdiff into the cover letter, or as commentary of the lone patch of a 1-patch series, showing the differences between the previous version of the patch series and the series currently being formatted. previous is a single revision naming the tip of the previous series which shares a common base with the series being formatted (for example git format-patch --cover-letter --interdiff=feature/v1 -3 feature/v2).

--range-diff=<previous>

As a reviewer aid, insert a range-diff (see git-range-diff(1)) into the cover letter, or as commentary of the lone patch of a 1-patch series, showing the differences between the previous version of the patch series and the series currently being formatted. previous can be a single revision naming the tip of the previous series if it shares a common base with the series being formatted (for example git format-patch --cover-letter --range-diff=feature/v1 -3 feature/v2), or a revision range if the two versions of the series are disjoint (for example git format-patch --cover-letter --range-diff=feature/v1~3..feature/v1 -3 feature/v2).

Note that diff options passed to the command affect how the primary product of format-patch is generated, and they are not passed to the underlying range-diff machinery used to generate the cover-letter material (this may change in the future).

--creation-factor=<percent>

Used with --range-diff, tweak the heuristic which matches up commits between the previous and current series of patches by adjusting the creation/deletion

cost fudge factor. See git-range-diff(1)) for details.

--notes[=<ref>], --no-notes

Append the notes (see git-notes(1)) for the commit after the three-dash line.

The expected use case of this is to write supporting explanation for the commit that does not belong to the commit log message proper, and include it with the patch submission. While one can simply write these explanations after format-patch has run but before sending, keeping them as Git notes allows them to be maintained between versions of the patch series (but see the discussion of the notes.rewrite configuration options in git-notes(1) to use this workflow).

The default is --no-notes, unless the format.notes configuration is set.

--[no-]signature=<signature>

Add a signature to each message produced. Per RFC 3676 the signature is separated from the body by a line with '-- ' on it. If the signature option is omitted the signature defaults to the Git version number.

--signature-file=<file>

Works just like --signature except the signature is read from a file.

--suffix=.<sfx>

Instead of using .patch as the suffix for generated filenames, use specified suffix. A common alternative is --suffix=.txt. Leaving this empty will remove the .patch suffix.

Note that the leading character does not have to be a dot; for example, you can use --suffix=-patch to get 0001-description-of-my-change-patch.

-q, --quiet

Do not print the names of the generated files to standard output.

--no-binary

Do not output contents of changes in binary files, instead display a notice that those files changed. Patches generated using this option cannot be applied properly, but they are still useful for code review.

--zero-commit

Output an all-zero hash in each patch?s From header instead of the hash of the commit.

--[no-]base[=<commit>]

Record the base tree information to identify the state the patch series applies to. See the BASE TREE INFORMATION section below for details. If <commit> is "auto", a base commit is automatically chosen. The --no-base option overrides a format.useAutoBase configuration.

--root

Treat the revision argument as a <revision range>, even if it is just a single commit (that would normally be treated as a <since>). Note that root commits included in the specified range are always formatted as creation patches, independently of this flag.

--progress

Show progress reports on stderr as patches are generated.

CONFIGURATION

You can specify extra mail header lines to be added to each message, defaults for the subject prefix and file suffix, number patches when outputting more than one patch, add "To:" or "Cc:" headers, configure attachments, change the patch output directory, and sign off patches with configuration variables.

```
[format]
        headers = "Organization: git-foo\n"
        subjectPrefix = CHANGE
        suffix = .txt
        numbered = auto
        to = <email>
        cc = <email>
        attach [ = mime-boundary-string ]
        signOff = true
        outputDirectory = <directory>
        coverLetter = auto
        coverFromDescription = auto
```

DISCUSSION

The patch produced by git format-patch is in UNIX mailbox format, with a fixed "magic" time stamp to indicate that the file is output from format-patch rather than a real mailbox, like so:

```
From 8f72bad1baf19a53459661343e21d6491c3908d3 Mon Sep 17 00:00:00 2001
```

From: Tony Luck <tony.luck@intel.com>

Date: Tue, 13 Jul 2010 11:42:54 -0700

Subject: [PATCH] =?UTF-8?q?[IA64]=20Put=20ia64=20config=20files=20on=20the=20?=

=?UTF-8?q?Uwe=20Kleine-K=C3=B6nig=20diet?=

MIME-Version: 1.0

Content-Type: text/plain; charset=UTF-8

Content-Transfer-Encoding: 8bit

arch/arm config files were slimmed down using a python script

(See commit c2330e286f68f1c408b4aa6515ba49d57f05beae comment)

Do the same for ia64 so we can have sleek & trim looking

...

Typically it will be placed in a MUA?s drafts folder, edited to add timely commentary that should not go in the changelog after the three dashes, and then sent as a message whose body, in our example, starts with "arch/arm config files were...". On the receiving end, readers can save interesting patches in a UNIX mailbox and apply them with git-am(1).

When a patch is part of an ongoing discussion, the patch generated by git format-patch can be tweaked to take advantage of the git am --scissors feature. After your response to the discussion comes a line that consists solely of "-- >8 --" (scissors and perforation), followed by the patch with unnecessary header fields removed:

...

> So we should do such-and-such.

Makes sense to me. How about this patch?

-- >8 --

Subject: [IA64] Put ia64 config files on the Uwe Kleine-K?nig diet

arch/arm config files were slimmed down using a python script

...

When sending a patch this way, most often you are sending your own patch, so in addition to the "From $SHA1 $magic_timestamp" marker you should omit From: and Date: lines from the patch file. The patch title is likely to be different from the subject of the discussion the patch is in response to, so it is likely that you would want to keep the Subject: line, like the example above.

Checking for patch corruption

Many mailers if not set up properly will corrupt whitespace. Here are two common types of corruption:

- Empty context lines that do not have any whitespace.
- Non-empty context lines that have one extra whitespace at the beginning.

One way to test if your MUA is set up correctly is:

- Send the patch to yourself, exactly the way you would, except with To: and Cc: lines that do not contain the list and maintainer address.
- Save that patch to a file in UNIX mailbox format. Call it a.patch, say.
- Apply it:

```
$ git fetch <project> master:test-apply
$ git switch test-apply
$ git restore --source=HEAD --staged --worktree :/
$ git am a.patch
```

If it does not apply correctly, there can be various reasons.

- The patch itself does not apply cleanly. That is bad but does not have much to do with your MUA. You might want to rebase the patch with git-rebase(1) before regenerating it in this case.
- The MUA corrupted your patch; "am" would complain that the patch does not apply. Look in the .git/rebase-apply/ subdirectory and see what patch file contains and check for the common corruption patterns mentioned above.
- While at it, check the info and final-commit files as well. If what is in final-commit is not exactly what you would want to see in the commit log message, it is very likely that the receiver would end up hand editing the log message when applying your patch. Things like "Hi, this is my first patch.\n" in the patch e-mail should come after the three-dash line that signals the end of the commit message.

MUA-SPECIFIC HINTS

Here are some hints on how to successfully submit patches inline using various mailers.

GMail

GMail does not have any way to turn off line wrapping in the web interface, so it will mangle any emails that you send. You can however use "git send-email" and send

your patches through the GMail SMTP server, or use any IMAP email client to connect to the google IMAP server and forward the emails through that.

For hints on using git send-email to send your patches through the GMail SMTP server, see the EXAMPLE section of git-send-email(1).

For hints on submission using the IMAP interface, see the EXAMPLE section of git-imap-send(1).

Thunderbird

By default, Thunderbird will both wrap emails as well as flag them as being format=flowed, both of which will make the resulting email unusable by Git.

There are three different approaches: use an add-on to turn off line wraps, configure Thunderbird to not mangle patches, or use an external editor to keep Thunderbird from mangling the patches.

Approach #1 (add-on)

Install the Toggle Word Wrap add-on that is available from https://addons.mozilla.org/thunderbird/addon/toggle-word-wrap/ It adds a menu entry "Enable Word Wrap" in the composer?s "Options" menu that you can tick off. Now you can compose the message as you otherwise do (cut + paste, git format-patch | git imap-send, etc), but you have to insert line breaks manually in any text that you type.

Approach #2 (configuration)

Three steps:

1. Configure your mail server composition as plain text: Edit...Account Settings...Composition & Addressing, uncheck "Compose Messages in HTML".
2. Configure your general composition window to not wrap.

   In Thunderbird 2: Edit..Preferences..Composition, wrap plain text messages at 0

   In Thunderbird 3: Edit..Preferences..Advanced..Config Editor. Search for "mail.wrap_long_lines". Toggle it to make sure it is set to false. Also, search for "mailnews.wraplength" and set the value to 0.
3. Disable the use of format=flowed: Edit..Preferences..Advanced..Config Editor. Search for "mailnews.send_plaintext_flowed". Toggle it to make sure it is set to false.

After that is done, you should be able to compose email as you otherwise would

(cut + paste, git format-patch | git imap-send, etc), and the patches will not be mangled.

Approach #3 (external editor)

The following Thunderbird extensions are needed: AboutConfig from http://aboutconfig.mozdev.org/ and External Editor from http://globs.org/articles.php?lng=en&pg=8

1. Prepare the patch as a text file using your method of choice.
2. Before opening a compose window, use Edit?Account Settings to uncheck the "Compose messages in HTML format" setting in the "Composition & Addressing" panel of the account to be used to send the patch.
3. In the main Thunderbird window, before you open the compose window for the patch, use Tools?about:config to set the following to the indicated values:

        mailnews.send_plaintext_flowed => false
        mailnews.wraplength   => 0

4. Open a compose window and click the external editor icon.
5. In the external editor window, read in the patch file and exit the editor normally.

Side note: it may be possible to do step 2 with about:config and the following settings but no one?s tried yet.

        mail.html_compose   => false
        mail.identity.default.compose_html => false
        mail.identity.id?.compose_html   => false

There is a script in contrib/thunderbird-patch-inline which can help you include patches with Thunderbird in an easy way. To use it, do the steps above and then use the script as the external editor.

KMail

This should help you to submit patches inline using KMail.

1. Prepare the patch as a text file.
2. Click on New Mail.
3. Go under "Options" in the Composer window and be sure that "Word wrap" is not set.
4. Use Message ? Insert file... and insert the patch.
5. Back in the compose window: add whatever other text you wish to the message,

complete the addressing and subject fields, and press send.

BASE TREE INFORMATION

The base tree information block is used for maintainers or third party testers to know the exact state the patch series applies to. It consists of the base commit, which is a well-known commit that is part of the stable part of the project history everybody else works off of, and zero or more prerequisite patches, which are well-known patches in flight that is not yet part of the base commit that need to be applied on top of base commit in topological order before the patches can be applied.

The base commit is shown as "base-commit: " followed by the 40-hex of the commit object name. A prerequisite patch is shown as "prerequisite-patch-id: " followed by the 40-hex patch id, which can be obtained by passing the patch through the git patch-id --stable command.

Imagine that on top of the public commit P, you applied well-known patches X, Y and Z from somebody else, and then built your three-patch series A, B, C, the history would be like:

```
---P---X---Y---Z---A---B---C
```

With git format-patch --base=P -3 C (or variants thereof, e.g. with --cover-letter or using Z..C instead of -3 C to specify the range), the base tree information block is shown at the end of the first message the command outputs (either the first patch, or the cover letter), like this:

```
base-commit: P
prerequisite-patch-id: X
prerequisite-patch-id: Y
prerequisite-patch-id: Z
```

For non-linear topology, such as

```
---P---X---A---M---C
    \         /
     Y---Z---B
```

You can also use git format-patch --base=P -3 C to generate patches for A, B and C, and the identifiers for P, X, Y, Z are appended at the end of the first message.

If set --base=auto in cmdline, it will track base commit automatically, the base commit will be the merge base of tip commit of the remote-tracking branch and

revision-range specified in cmdline. For a local branch, you need to track a remote branch by git branch --set-upstream-to before using this option.

EXAMPLES

? Extract commits between revisions R1 and R2, and apply them on top of the current branch using git am to cherry-pick them:

$ git format-patch -k --stdout R1..R2 | git am -3 -k

? Extract all commits which are in the current branch but not in the origin branch:

$ git format-patch origin

For each commit a separate file is created in the current directory.

? Extract all commits that lead to origin since the inception of the project:

$ git format-patch --root origin

? The same as the previous one:

$ git format-patch -M -B origin

Additionally, it detects and handles renames and complete rewrites intelligently to produce a renaming patch. A renaming patch reduces the amount of text output, and generally makes it easier to review. Note that non-Git "patch" programs won?t understand renaming patches, so use it only when you know the recipient uses Git to apply your patch.

? Extract three topmost commits from the current branch and format them as e-mailable patches:

$ git format-patch -3

SEE ALSO

git-am(1), git-send-email(1)

GIT

Part of the git(1) suite