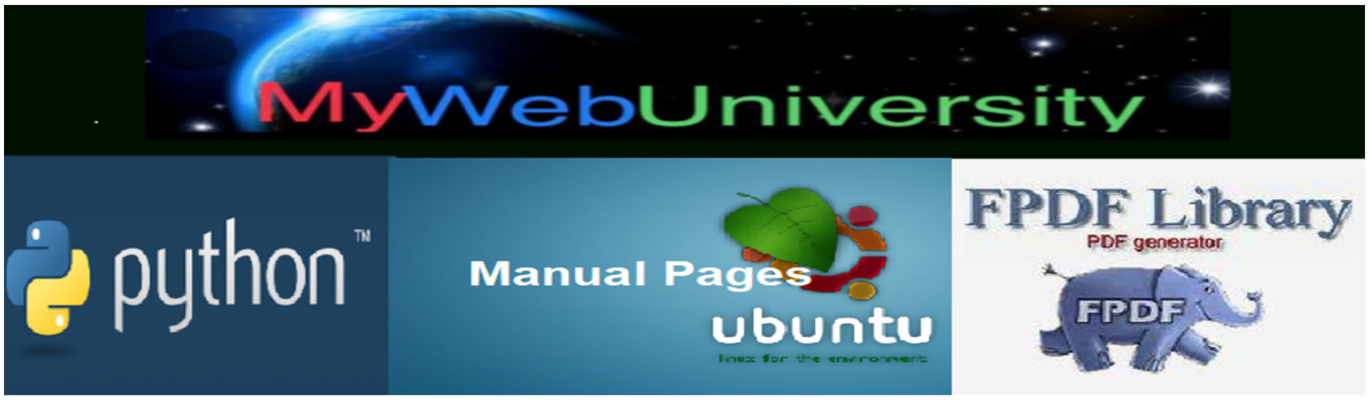

Rocky Enterprise Linux 9.2 Manual Pages on command 'patch.1'

$ man patch.1

PATCH(1) General Commands Manual PATCH(1)

NAME

patch - apply a diff file to an original

SYNOPSIS

patch [options] [originalfile [patchfile]]

but usually just

patch -pnum <patchfile

DESCRIPTION

patch takes a patch file patchfile containing a difference listing produced by the diff program and applies those differences to one or more original files, producing patched versions. Normally the patched versions are put in place of the originals. Backups can be made; see the -b or --backup option. The names of the files to be patched are usually taken from the patch file, but if there's just one file to be patched it can be specified on the command line as originalfile.

Upon startup, patch attempts to determine the type of the diff listing, unless overruled by a -c (--context), -e (--ed), -n (--normal), or -u (--unified) option. Context diffs (old-style, new-style, and unified) and normal diffs are applied by the patch program it? self, while ed diffs are simply fed to the ed(1) editor via a pipe.

patch tries to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a diff listing to patch, and it should work. If the entire diff is indented by a consistent amount, if lines end in CRLF, or if a diff is encapsulated one or more times by prepending "- " to lines start? ing with "-" as specified by Internet RFC 934, this is taken into account. After removing indenting or encapsulation, lines beginning with # are ignored, as they are considered to be comments.

With context diffs, and to a lesser extent with normal diffs, patch can detect when the line numbers mentioned in the patch are incorrect, and attempts to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, patch scans both forwards and backwards for a set of lines matching the context given in the hunk. First patch looks for a place where all lines of the context match. If no such place is found, and it's a context diff, and the maximum fuzz factor is set to 1 or more, then another scan takes place ignoring the first and last line of con? text. If that fails, and the maximum fuzz factor is set to 2 or more, the first two and last two lines of context are ignored, and another scan is made. (The default maximum fuzz factor is 2.)

Hunks with less prefix context than suffix context (after applying fuzz) must apply at the start of the file if their first line number is 1. Hunks with more prefix context than suffix context (after applying fuzz) must apply at the end of the file.

If patch cannot find a place to install that hunk of the patch, it puts the hunk out to a reject file, which normally is the name of the output file plus a .rej suffix, or # if .rej would generate a file name that is too long (if even appending the single character # makes the file name too long, then # replaces the file name's last character).

The rejected hunk comes out in unified or context diff format. If the input was a normal diff, many of the contexts are simply null. The line numbers on the hunks in the reject file may be different than in the patch file: they reflect the approximate location patch thinks the failed hunks belong in the new file rather than the old one.

As each hunk is completed, you are told if the hunk failed, and if so which line (in the new file) patch thought the hunk should go on. If the hunk is installed at a different line from the line number specified in the diff, you are told the offset. A single large offset may indicate that a hunk was installed in the wrong place. You are also told if a fuzz factor was used to make the match, in which case you should also be slightly suspi? cious. If the --verbose option is given, you are also told about hunks that match ex? actly.

If no original file origfile is specified on the command line, patch tries to figure out from the leading garbage what the name of the file to edit is, using the following rules.

First, patch takes an ordered list of candidate file names as follows:

? If the header is that of a context diff, patch takes the old and new file names in the header. A name is ignored if it does not have enough slashes to satisfy the -pnum or --strip=num option. The name /dev/null is also ignored.

? If there is an Index: line in the leading garbage and if either the old and new names are both absent or if patch is conforming to POSIX, patch takes the name in the Index: line.

? For the purpose of the following rules, the candidate file names are considered to be in the order (old, new, index), regardless of the order that they appear in the header.

Then patch selects a file name from the candidate list as follows:

? If some of the named files exist, patch selects the first name if conforming to POSIX, and the best name otherwise.

? If patch is not ignoring RCS, ClearCase, Perforce, and SCCS (see the -g num or --get=num option), and no named files exist but an RCS, ClearCase, Perforce, or SCCS master is found, patch selects the first named file with an RCS, ClearCase, Perforce, or SCCS master.

? If no named files exist, no RCS, ClearCase, Perforce, or SCCS master was found, some names are given, patch is not conforming to POSIX, and the patch appears to create a file, patch selects the best name requiring the creation of the fewest directories.

? If no file name results from the above heuristics, you are asked for the name of the file to patch, and patch selects that name.

To determine the best of a nonempty list of file names, patch first takes all the names with the fewest path name components; of those, it then takes all the names with the shortest basename; of those, it then takes all the shortest names; finally, it takes the first remaining name.

Additionally, if the leading garbage contains a Prereq: line, patch takes the first word from the prerequisites line (normally a version number) and checks the original file to see if that word can be found. If not, patch asks for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, some?
thing like the following:

```
| patch -d /usr/src/local/blurfl
```

and patch a file in the blurfl directory directly from the article containing the patch.

If the patch file contains more than one patch, patch tries to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that the name of the file to patch must be determined for each diff listing, and that the garbage before each diff listing contains interesting things such as file names and revi? sion level, as mentioned previously.

OPTIONS

-b or --backup

Make backup files. That is, when patching a file, rename or copy the original instead of removing it.   See the -V or --version-control option for details about how backup file names are determined.

--backup-if-mismatch

Back up a file if the patch does not match the file exactly and if backups are not oth? erwise requested. This is the default unless patch is conforming to POSIX.

--no-backup-if-mismatch

Do not back up a file if the patch does not match the file exactly and if backups are not otherwise requested. This is the default if patch is conforming to POSIX.

-B pref or --prefix=pref

Use the simple method to determine backup file names (see the -V method or --ver? sion-control method option), and append pref to a file name when generating its backup file name.   For example, with -B /junk/ the   simple   backup   file   name   for src/patch/util.c is /junk/src/patch/util.c.

--binary

Write all files in binary mode, except for standard output and /dev/tty. When reading, disable the heuristic for transforming CRLF line endings into LF line endings.   This option is needed on POSIX systems when applying patches generated on non-POSIX systems to non-POSIX files. (On POSIX systems, file reads and writes never transform line end? ings. On Windows, reads and writes do transform line endings by default, and patches should be generated by diff --binary when line endings are significant.)

-c or --context

Interpret the patch file as a ordinary context diff.

-d dir or --directory=dir

Change to the directory dir immediately, before doing anything else.

-D define or --ifdef=define

Use the #ifdef ... #endif construct to mark changes, with define as the differentiating symbol.

--dry-run

Print the results of applying the patches without actually changing any files.

-e or --ed

Interpret the patch file as an ed script.

-E or --remove-empty-files

Remove output files that are empty after the patches have been applied. Normally this option is unnecessary, since patch can examine the time stamps on the header to deter? mine whether a file should exist after patching. However, if the input is not a con? text diff or if patch is conforming to POSIX, patch does not remove empty patched files unless this option is given. When patch removes a file, it also attempts to remove any empty ancestor directories.

-f or --force

Assume that the user knows exactly what he or she is doing, and do not ask any ques? tions.   Skip patches whose headers do not say which file is to be patched; patch files even though they have the wrong version for the Prereq: line in the patch; and assume that patches are not reversed even if they look like they are. This option does not suppress commentary; use -s for that.

-F num or --fuzz=num

Set the maximum fuzz factor. This option only applies to diffs that have context, and causes patch to ignore up to that many lines of context in looking for places to in? stall a hunk. Note that a larger fuzz factor increases the odds of a faulty patch. The default fuzz factor is 2. A fuzz factor greater than or equal to the number of lines of context in the context diff, ordinarily 3, ignores all context.

-g num or --get=num

This option controls patch's actions when a file is under RCS or SCCS control, and does not exist or is read-only and matches the default version, or when a file is under ClearCase or Perforce control and does not exist. If num is positive, patch gets (or checks out) the file from the revision control system; if zero, patch ignores RCS, ClearCase, Perforce, and SCCS and does not get the file; and if negative, patch asks the user whether to get the file. The default value of this option is given by the value of the PATCH_GET environment variable if it is set; if not, the default value is zero.

--help

Print a summary of options and exit.

-i patchfile or --input=patchfile

Read the patch from patchfile. If patchfile is -, read from standard input, the de?

fault.

-l or --ignore-whitespace

Match patterns loosely, in case tabs or spaces have been munged in your files. Any se?

quence of one or more blanks in the patch file matches any sequence in the original

file, and sequences of blanks at the ends of lines are ignored. Normal characters must

still match exactly. Each line of the context must still match a line in the original

file.

--merge or --merge=merge or --merge=diff3

Merge a patch file into the original files similar to diff3(1) or merge(1). If a con?

flict is found, patch outputs a warning and brackets the conflict with <<<<<<< and

>>>>>>> lines. A typical conflict will look like this:

```
<<<<<<<
lines from the original file
|||||||
original lines from the patch
=======
new lines from the patch
>>>>>>>
```

The optional argument of --merge determines the output format for conflicts: the diff3

format shows the ||||||| section with the original lines from the patch; in the merge

format, this section is missing. The merge format is the default.

This option implies --forward and does not take the --fuzz=num option into account.

-n or --normal

Interpret the patch file as a normal diff.

-N or --forward

When a patch does not apply, patch usually checks if the patch looks like it has been

applied already by trying to reverse-apply the first hunk. The --forward option pre?

vents that. See also -R.

-o outfile or --output=outfile

Send output to outfile instead of patching files in place. Do not use this option if

outfile is one of the files to be patched. When outfile is -, send output to standard

output, and send any messages that would usually go to standard output to standard er?

ror.

-pnum or --strip=num

Strip the smallest prefix containing num leading slashes from each file name found in

the patch file. A sequence of one or more adjacent slashes is counted as a single

slash.   This controls how file names found in the patch file are treated, in case you

keep your files in a different directory than the person who sent out the patch.   For

example, supposing the file name in the patch file was

/u/howard/src/blurfl/blurfl.c

setting -p0 gives the entire file name unmodified, -p1 gives

u/howard/src/blurfl/blurfl.c

without the leading slash, -p4 gives

blurfl/blurfl.c

and not specifying -p at all just gives you blurfl.c. Whatever you end up with is looked

for either in the current directory, or the directory specified by the -d option.

--posix

Conform more strictly to the POSIX standard, as follows.

? Take the first existing file from the list (old, new, index) when intuiting file

names from diff headers.

? Do not remove files that are empty after patching.

? Do not ask whether to get files from RCS, ClearCase, Perforce, or SCCS.

? Require that all options precede the files in the command line.

? Do not backup files when there is a mismatch.

--quoting-style=word

Use style word to quote output names. The word should be one of the following:

literal

Output names as-is.

shell Quote names for the shell if they contain shell metacharacters or would cause

ambiguous output.

shell-always

Quote names for the shell, even if they would normally not require quoting.

c   Quote names as for a C language string.

escape Quote as with c except omit the surrounding double-quote characters.

You can specify the default value of the --quoting-style option with the environment variable QUOTING_STYLE. If that environment variable is not set, the default value is shell.

-r rejectfile or --reject-file=rejectfile

Put rejects into rejectfile instead of the default .rej file. When rejectfile is -, discard rejects.

-R or --reverse

Assume that this patch was created with the old and new files swapped. (Yes, I'm afraid that does happen occasionally, human nature being what it is.)   patch attempts to swap each hunk around before applying it. Rejects come out in the swapped format. The -R option does not work with ed diff scripts because there is too little informa? tion to reconstruct the reverse operation.

If the first hunk of a patch fails, patch reverses the hunk to see if it can be applied that way. If it can, you are asked if you want to have the -R option set.   If it can't, the patch continues to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed, due to the fact that a null context matches anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs begin with a delete, which fails, triggering the heuristic.)

--read-only=behavior

Behave as requested when trying to modify a read-only file: ignore the potential prob? lem, warn about it (the default), or fail.

--reject-format=format

Produce reject files in the specified format (either context or unified). Without this option, rejected hunks come out in unified diff format if the input patch was of that format, otherwise in ordinary context diff form.

-s or --silent or --quiet

Work silently, unless an error occurs.

--follow-symlinks

When looking for input files, follow symbolic links. Replaces the symbolic links, in?

stead of modifying the files the symbolic links point to. Git-style patches to sym?
bolic links will no longer apply. This option exists for backwards compatibility with
previous versions of patch; its use is discouraged.

-t or --batch

Suppress questions like -f, but make some different assumptions: skip patches whose
headers do not contain file names (the same as -f); skip patches for which the file has
the wrong version for the Prereq: line in the patch; and assume that patches are re?
versed if they look like they are.

-T or --set-time

Set the modification and access times of patched files from time stamps given in con?
text diff headers. Unless specified in the time stamps, assume that the context diff
headers use local time.

Use of this option with time stamps that do not include time zones is not recommended,
because patches using local time cannot easily be used by people in other time zones,
and because local time stamps are ambiguous when local clocks move backwards during
daylight-saving time adjustments. Make sure that time stamps include time zones, or
generate patches with UTC and use the -Z or --set-utc option instead.

-u or --unified

Interpret the patch file as a unified context diff.

-v or --version

Print out patch's revision header and patch level, and exit.

-V method or --version-control=method

Use method to determine backup file names.   The method can also be given by the
PATCH_VERSION_CONTROL (or, if that's not set, the VERSION_CONTROL) environment vari?
able, which is overridden by this option. The method does not affect whether backup
files are made; it affects only the names of any backup files that are made.

The value of method is like the GNU Emacs `version-control' variable; patch also recog?
nizes synonyms that are more descriptive. The valid values for method are (unique ab?
breviations are accepted):

existing or nil

Make numbered backups of files that already have them, otherwise simple backups.
This is the default.

numbered or t

Make numbered backups. The numbered backup file name for F is F.~N~ where N is the

version number.

simple or never

Make simple backups. The -B or --prefix, -Y or --basename-prefix, and -z or --suf?

fix options specify the simple backup file name. If none of these options are

given, then a simple backup suffix is used; it is the value of the SIM?

PLE_BACKUP_SUFFIX environment variable if set, and is .orig otherwise.

With numbered or simple backups, if the backup file name is too long, the backup suffix

~ is used instead; if even appending ~ would make the name too long, then ~ replaces

the last character of the file name.

--verbose

Output extra information about the work being done.

-x num or --debug=num

Set internal debugging flags of interest only to patch patchers.

-Y pref or --basename-prefix=pref

Use the simple method to determine backup file names (see the -V method or --ver?

sion-control method option), and prefix pref to the basename of a file name when gener?

ating its backup file name. For example, with -Y .del/ the simple backup file name for

src/patch/util.c is src/patch/.del/util.c.

-z suffix or --suffix=suffix

Use the simple method to determine backup file names (see the -V method or --ver?

sion-control method option), and use suffix as the suffix. For example, with -z - the

backup file name for src/patch/util.c is src/patch/util.c-.

-Z or --set-utc

Set the modification and access times of patched files from time stamps given in con?

text diff headers. Unless specified in the time stamps, assume that the context diff

headers use Coordinated Universal Time (UTC, often known as GMT). Also see the -T or

--set-time option.

The -Z or --set-utc and -T or --set-time options normally refrain from setting a file's

time if the file's original time does not match the time given in the patch header, or

if its contents do not match the patch exactly. However, if the -f or --force option

is given, the file time is set regardless.

Due to the limitations of diff output format, these options cannot update the times of

files whose contents have not changed. Also, if you use these options, you should re?
move (e.g. with make clean) all files that depend on the patched files, so that later
invocations of make do not get confused by the patched files' times.

ENVIRONMENT

PATCH_GET

This specifies whether patch gets missing or read-only files from RCS, ClearCase, Per?
force, or SCCS by default; see the -g or --get option.

POSIXLY_CORRECT

If set, patch conforms more strictly to the POSIX standard by default: see the --posix
option.

QUOTING_STYLE

Default value of the --quoting-style option.

SIMPLE_BACKUP_SUFFIX

Extension to use for simple backup file names instead of .orig.

TMPDIR, TMP, TEMP

Directory to put temporary files in; patch uses the first environment variable in this
list that is set. If none are set, the default is system-dependent; it is normally
/tmp on Unix hosts.

VERSION_CONTROL or PATCH_VERSION_CONTROL

Selects version control style; see the -v or --version-control option.

FILES

$TMPDIR/p*

temporary files

/dev/tty

controlling terminal; used to get answers to questions asked of the user

SEE ALSO

diff(1), ed(1), merge(1).

Marshall T. Rose and Einar A. Stefferud, Proposed Standard for Message Encapsulation, In?
ternet RFC 934 <URL:ftp://ftp.isi.edu/in-notes/rfc934.txt> (1985-01).

NOTES FOR PATCH SENDERS

There are several things you should bear in mind if you are going to be sending out
patches.

Create your patch systematically. A good method is the command diff -Naur old new where

old and new identify the old and new directories. The names old and new should not con?
tain any slashes.   The diff command's headers should have dates and times in Universal
Time using traditional Unix format, so that patch recipients can use the -Z or --set-utc
option. Here is an example command, using Bourne shell syntax:

```
LC_ALL=C TZ=UTC0 diff -Naur gcc-2.7 gcc-2.8
```

Tell your recipients how to apply the patch by telling them which directory to cd to, and
which patch options to use. The option string -Np1 is recommended. Test your procedure
by pretending to be a recipient and applying your patch to a copy of the original files.

You can save people a lot of grief by keeping a patchlevel.h file which is patched to in?
crement the patch level as the first diff in the patch file you send out. If you put a
Prereq: line in with the patch, it won't let them apply patches out of order without some
warning.

You can create a file by sending out a diff that compares /dev/null or an empty file dated
the Epoch (1970-01-01 00:00:00 UTC) to the file you want to create. This only works if
the file you want to create doesn't exist already in the target directory.   Conversely,
you can remove a file by sending out a context diff that compares the file to be deleted
with an empty file dated the Epoch. The file will be removed unless patch is conforming
to POSIX and the -E or --remove-empty-files option is not given. An easy way to generate
patches that create and remove files is to use GNU diff's -N or --new-file option.

If the recipient is supposed to use the -pN option, do not send output that looks like
this:

```
diff -Naur v2.0.29/prog/README prog/README
--- v2.0.29/prog/README   Mon Mar 10 15:13:12 1997
+++ prog/README   Mon Mar 17 14:58:22 1997
```

because the two file names have different numbers of slashes, and different versions of
patch interpret the file names differently. To avoid confusion, send output that looks
like this instead:

```
diff -Naur v2.0.29/prog/README v2.0.30/prog/README
--- v2.0.29/prog/README   Mon Mar 10 15:13:12 1997
+++ v2.0.30/prog/README   Mon Mar 17 14:58:22 1997
```

Avoid sending patches that compare backup file names like README.orig, since this might
confuse patch into patching a backup file instead of the real file. Instead, send patches
that compare the same base file names in different directories, e.g. old/README and

new/README.

Take care not to send out reversed patches, since it makes people wonder whether they al?
ready applied the patch.

Try not to have your patch modify derived files (e.g. the file configure where there is a
line configure: configure.in in your makefile), since the recipient should be able to re?
generate the derived files anyway. If you must send diffs of derived files, generate the
diffs using UTC, have the recipients apply the patch with the -Z or --set-utc option, and
have them remove any unpatched files that depend on patched files (e.g. with make clean).

While you may be able to get away with putting 582 diff listings into one file, it may be
wiser to group related patches into separate files in case something goes haywire.

DIAGNOSTICS

Diagnostics generally indicate that patch couldn't parse your patch file.

If the --verbose option is given, the message Hmm... indicates that there is unprocessed
text in the patch file and that patch is attempting to intuit whether there is a patch in
that text and, if so, what kind of patch it is.

patch's exit status is 0 if all hunks are applied successfully, 1 if some hunks cannot be
applied or there were merge conflicts, and 2 if there is more serious trouble. When ap?
plying a set of patches in a loop it behooves you to check this exit status so you don't
apply a later patch to a partially patched file.

CAVEATS

Context diffs cannot reliably represent the creation or deletion of empty files, empty di?
rectories, or special files such as symbolic links. Nor can they represent changes to
file metadata like ownership, permissions, or whether one file is a hard link to another.
If changes like these are also required, separate instructions (e.g. a shell script) to
accomplish them should accompany the patch.

patch cannot tell if the line numbers are off in an ed script, and can detect bad line
numbers in a normal diff only when it finds a change or deletion. A context diff using
fuzz factor 3 may have the same problem. You should probably do a context diff in these
cases to see if the changes made sense. Of course, compiling without errors is a pretty
good indication that the patch worked, but not always.

patch usually produces the correct results, even when it has to do a lot of guessing.
However, the results are guaranteed to be correct only when the patch is applied to ex?
actly the same version of the file that the patch was generated from.

COMPATIBILITY ISSUES

The POSIX standard specifies behavior that differs from patch's traditional behavior. You should be aware of these differences if you must interoperate with patch versions 2.1 and earlier, which do not conform to POSIX.

? In traditional patch, the -p option's operand was optional, and a bare -p was equiva? lent to -p0. The -p option now requires an operand, and -p 0 is now equivalent to -p0. For maximum compatibility, use options like -p0 and -p1.

Also, traditional patch simply counted slashes when stripping path prefixes; patch now counts pathname components. That is, a sequence of one or more adjacent slashes now counts as a single slash. For maximum portability, avoid sending patches containing // in file names.

? In traditional patch, backups were enabled by default. This behavior is now enabled with the -b or --backup option.

Conversely, in POSIX patch, backups are never made, even when there is a mismatch. In GNU patch, this behavior is enabled with the --no-backup-if-mismatch option, or by con? forming to POSIX with the --posix option or by setting the POSIXLY_CORRECT environment variable.

The -b suffix option of traditional patch is equivalent to the -b -z suffix options of GNU patch.

? Traditional patch used a complicated (and incompletely documented) method to intuit the name of the file to be patched from the patch header. This method did not conform to POSIX, and had a few gotchas. Now patch uses a different, equally complicated (but better documented) method that is optionally POSIX-conforming; we hope it has fewer gotchas. The two methods are compatible if the file names in the context diff header and the Index: line are all identical after prefix-stripping. Your patch is normally compatible if each header's file names all contain the same number of slashes.

? When traditional patch asked the user a question, it sent the question to standard er? ror and looked for an answer from the first file in the following list that was a ter? minal: standard error, standard output, /dev/tty, and standard input. Now patch sends questions to standard output and gets answers from /dev/tty. Defaults for some answers have been changed so that patch never goes into an infinite loop when using default an? swers.

? Traditional patch exited with a status value that counted the number of bad hunks, or with status 1 if there was real trouble. Now patch exits with status 1 if some hunks failed, or with 2 if there was real trouble.

? Limit yourself to the following options when sending instructions meant to be executed by anyone running GNU patch, traditional patch, or a patch that conforms to POSIX. Spaces are significant in the following list, and operands are required.

-c

-d dir

-D define

-e

-l

-n

-N

-o outfile

-pnum

-R

-r rejectfile

BUGS

Please report bugs via email to <bug-patch@gnu.org>.

If code has been duplicated (for instance with #ifdef OLDCODE ... #else ... #endif), patch is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you that it succeeded to boot.

If you apply a patch you've already applied, patch thinks it is a reversed patch, and of?
fers to un-apply the patch. This could be construed as a feature.

Computing how to merge a hunk is significantly harder than using the standard fuzzy algo?
rithm. Bigger hunks, more context, a bigger offset from the original location, and a worse match all slow the algorithm down.

COPYING

Copyright (C) 1984, 1985, 1986, 1988 Larry Wall.

Copyright (C) 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2009 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the

AUTHORS

Larry Wall wrote the original version of patch. Paul Eggert removed patch's arbitrary

limits; added support for binary files, setting file times, and deleting files; and made

it conform better to POSIX. Other contributors include Wayne Davison, who added unidiff

support, and David MacKenzie, who added configuration and backup support. Andreas

Gr?nbacher added support for merging.