NAME

patch - apply a diff file to an original

SYNOPSIS

patch [options] [origfile [patchfile]] [+ [options] [origfile]]...

but usually just

patch <patchfile

DESCRIPTION

Patch will take a patch file containing any of the four forms of difference listing produced by the *diff* program and apply those differences to an original file, producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension ‘‘.orig’’ (‘‘~’’ on systems that do not support long filenames), or as specified by the **-b**, **-B**, or **-V** switches. The extension used for making backup files may also be specified in the **SIMPLE_BACKUP_SUFFIX** environment variable, which is overridden by above switches.

If the backup file already exists, **patch** creates a new backup file name by changing the first lowercase letter in the last component of the file’s name into uppercase. If there are no more lowercase letters in the name, it removes the first character from the name. It repeats this process until it comes up with a backup file that does not already exist.

You may also specify where you want the output to go with a **-o** switch; if that file already exists, it is backed up first.

If *patchfile* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, patch will attempt to determine the type of the diff listing, unless over-ruled by a **-c**, **-e**, **-n**, or **-u** switch. Context diffs (old-style, new-style, and unified) and normal diffs are applied by the *patch* program itself, while ed diffs are simply fed to the *ed* editor via a pipe.

Patch will try 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, this will be taken into account.

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 will attempt 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* will scan 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 context. 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.) If *patch* cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus ‘‘.rej’’ (‘‘#’’ on systems that do not support long filenames). (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be 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 will be told whether the hunk succeeded or failed, and which line (in the new file) *patch* thought the hunk should go on. If this is different from the line number specified in the diff you will be told the offset. A single large offset MAY be an indication that a hunk was installed in the wrong place. You will also be told if a fuzz factor was used to make the match, in which case you should also be slightly suspicious.

If no original file is specified on the command line, *patch* will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the filename is found from lines beginning with ‘‘***’’ or ‘‘---’’, with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an ‘‘Index:’’ line in the leading garbage, *patch* will try to use the filename from that line. The context diff header takes precedence over an Index line. If no filename can be intuited from the leading garbage, you will be asked for the name of the file to patch.

If the original file cannot be found or is read-only, but a suitable SCCS or RCS file is handy, *patch* will attempt to get or check out the file.

Additionally, if the leading garbage contains a ‘‘Prereq: ’’ line, *patch* will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, *patch* will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, 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* will try 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 will be examined for interesting things such as filenames and revision level, as mentioned previously. You can give switches (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a ‘+’. (The argument list for a second or subsequent patch may not specify a new patch file, however.)

Patch recognizes the following switches:

−b causes the next argument to be interpreted as the backup extension, to be used in place of ‘‘.orig’’ or ‘‘~’’.

−B causes the next argument to be interpreted as a prefix to the backup file name. If this argument is specified any argument from -b will be ignored.

−c forces *patch* to interpret the patch file as a context diff.

−d causes *patch* to interpret the next argument as a directory, and cd to it before doing anything else.

−D causes *patch* to use the "#ifdef...#endif" construct to mark changes. The argument following will be used as the differentiating symbol. Note that, unlike the C compiler, there must be a space between the **−D** and the argument.

−e forces *patch* to interpret the patch file as an ed script.

−E causes *patch* to remove output files that are empty after the patches have been applied.

−f forces *patch* to assume that the user knows exactly what he or she is doing, and to not ask any questions. It assumes the following: skip patches for which a file to patch can’t be found; 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.

−t similar to **−f**, in that it suppresses questions, but makes some different assumptions: skip patches for which a file to patch can’t be found (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 reversed if they look like they are.

−F<number>
sets the maximum fuzz factor. This switch only applies to context diffs, and causes *patch* to ignore up to that many lines in looking for places to install a hunk. Note that a larger fuzz factor increases the odds of a faulty patch. The default fuzz factor is 2, and it may not be set to more than the number of lines of context in the context diff, ordinarily 3.

−l causes the pattern matching to be done loosely, in case the tabs and spaces have been munged in your input file. Any sequence of whitespace in the pattern line will match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.

−n forces *patch* to interpret the patch file as a normal diff.

−N causes *patch* to ignore patches that it thinks are reversed or already applied. See also **−R .**

−o causes the next argument to be interpreted as the output file name.

−p<number>
sets the pathname strip count, which controls how pathnames found in the patch file are treated, in case the you keep your files in a different directory than the person who sent out the patch. The strip count specifies how many slashes are to be stripped from the front of the pathname. (Any intervening directory names also go away.) For example, supposing the filename in the patch file was

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

setting **−p** or **−p0** gives the entire pathname 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", unless all of the directories in the leading path (u/howard/src/blurfl) exist and that path is relative, in which case you get the entire pathname unmodified. Whatever you end up with is looked for either in the current directory, or the directory specified by the **−d** switch.

−r causes the next argument to be interpreted as the reject file name.

−R tells *patch* 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* will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **−R** switch will not work with ed diff scripts because there is too little information to reconstruct the reverse operation.

If the first hunk of a patch fails, *patch* will reverse the hunk to see if it can be applied that way. If it can, you will be asked if you want to have the **−R** switch set. If it can't, the patch will continue 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 will match anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)

−s makes *patch* do its work silently, unless an error occurs.

−S causes *patch* to ignore this patch from the patch file, but continue on looking for the next patch in the file. Thus

patch -S + -S + <patchfile

will ignore the first and second of three patches.

−u forces *patch* to interpret the patch file as a unified context diff (a unidiff).

−v causes *patch* to print out its revision header and patch level.

−V causes the next argument to be interpreted as a method for creating backup file names. The type of backups made can also be given in the **VERSION_CONTROL** environment variable, which is overridden by this option. The **-B** option overrides this option, causing the prefix to always be used for making backup file names. The value of the **VERSION_CONTROL** environment variable and the argument to the **-V** option are like the GNU Emacs 'version-control' variable; they also recognize synonyms that are more descriptive. The valid values are (unique abbreviations are accepted):

't' or 'numbered'
    Always make numbered backups.

'nil' or 'existing'
    Make numbered backups of files that already have them, simple backups of the others. This is the default.

'never' or 'simple'
    Always make simple backups.

−x<number>
sets internal debugging flags, and is of interest only to *patch* patchers.

AUTHOR

Larry Wall <lwall@netlabs.com>
with many other contributors.

ENVIRONMENT

TMPDIR
Directory to put temporary files in; default is /tmp.

SIMPLE_BACKUP_SUFFIX
Extension to use for backup file names instead of ''.orig'' or ''~''.

VERSION_CONTROL
Selects when numbered backup files are made.

FILES

$TMPDIR/patch*

SEE ALSO

diff(1)

NOTES FOR PATCH SENDERS

There are several things you should bear in mind if you are going to be sending out patches. First, you can save people a lot of grief by keeping a patchlevel.h file which is patched to increment 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. Second, make sure you've specified the filenames right, either in a context diff header, or with an Index: line. If you are patching something in a subdirectory, be sure to tell the patch user to specify a **−p** switch as needed. Third, you can create a file by sending out a diff that compares a null file to the file you want to create. This will only work if the file you want to create doesn't exist already in the target directory. Fourth, take care not to send out reversed patches, since it makes people wonder whether they already applied the patch. Fifth, while you may be able to get away with putting 582 diff listings into one file, it is probably wiser to group related patches into separate files in case something goes haywire.

DIAGNOSTICS

Too many to list here, but generally indicative that *patch* couldn't parse your patch file.

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 will exit with a non-zero status if any reject files were created. When applying 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

Patch cannot tell if the line numbers are off in an ed script, and can only detect bad line numbers in a normal diff when it finds a ''change'' or a ''delete'' command. A context diff using fuzz factor 3 may have the same problem. Until a suitable interactive interface is added, 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 exactly the same version of the file that the patch was generated from.

BUGS

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

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* will think it is a reversed patch, and offer to unapply the patch. This could be construed as a feature.