

NAME

gitdiffcore - Tweaking diff output

SYNOPSIS

git diff *

DESCRIPTION

The diff commands *git diff-index*, *git diff-files*, and *git diff-tree* can be told to manipulate differences they find in unconventional ways before showing *diff* output. The manipulation is collectively called "diffcore transformation". This short note describes what they are and how to use them to produce *diff* output that is easier to understand than the conventional kind.

THE CHAIN OF OPERATION

The *git diff*-* family works by first comparing two sets of files:

⊕

diff-index compares contents of a "tree" object and the working directory (when **--cached** flag is not used) or a "tree" object and the index file (when **--cached** flag is used);

⊕

diff-files compares contents of the index file and the working directory;

⊕

diff-tree compares contents of two "tree" objects;

In all of these cases, the commands themselves first optionally limit the two sets of files by any pathspecs given on their command-lines, and compare corresponding paths in the two resulting sets of files.

The pathspecs are used to limit the world diff operates in. They remove the filepairs outside the specified sets of pathnames. E.g. If the input set of filepairs included:

```
:100644 100644 bcd1234... 0123456... M junkfile
```

but the command invocation was **git diff-files myfile**, then the junkfile entry would be removed from the list because only "myfile" is under consideration.

The result of comparison is passed from these commands to what is internally called "diffcore", in a

format similar to what is output when the `-p` option is not used. E.g.

```
in-place edit :100644 100644 bcd1234... 0123456... M file0
create       :000000 100644 0000000... 1234567... A file4
delete      :100644 000000 1234567... 0000000... D file5
unmerged    :000000 000000 0000000... 0000000... U file6
```

The diffcore mechanism is fed a list of such comparison results (each of which is called "filepair", although at this point each of them talks about a single file), and transforms such a list into another list. There are currently 5 such transformations:

- ⊕
- ⊕
- ⊕
- ⊕
- ⊕
- ⊕

These are applied in sequence. The set of filepairs *git diff*-* commands find are used as the input to `diffcore-break`, and the output from `diffcore-break` is used as the input to the next transformation. The final result is then passed to the output routine and generates either `diff-raw` format (see Output format sections of the manual for *git diff*-* commands) or `diff-patch` format.

DIFFCORE-BREAK: FOR SPLITTING UP COMPLETE REWRITES

The second transformation in the chain is `diffcore-break`, and is controlled by the `-B` option to the *git diff*-* commands. This is used to detect a filepair that represents "complete rewrite" and break such filepair into two filepairs that represent delete and create. E.g. If the input contained this filepair:

```
:100644 100644 bcd1234... 0123456... M file0
```

and if it detects that the file "file0" is completely rewritten, it changes it to:

```
:100644 000000 bcd1234... 0000000... D file0
```

```
:000000 100644 0000000... 0123456... A file0
```

For the purpose of breaking a filepair, `diffcore-break` examines the extent of changes between the contents of the files before and after modification (i.e. the contents that have "bcd1234..." and "0123456..." as their SHA-1 content ID, in the above example). The amount of deletion of original contents and insertion of new material are added together, and if it exceeds the "break score", the filepair is broken into two. The break score defaults to 50% of the size of the smaller of the original and the result (i.e. if the edit shrinks the file, the size of the result is used; if the edit lengthens the file, the size of the original is used), and can be customized by giving a number after "-B" option (e.g. "-B75" to tell it to use 75%).

DIFFCORE-RENAME: FOR DETECTING RENAMES AND COPIES

This transformation is used to detect renames and copies, and is controlled by the `-M` option (to detect renames) and the `-C` option (to detect copies as well) to the `git diff-*` commands. If the input contained these filepairs:

```
:100644 000000 0123456... 0000000... D fileX
:000000 100644 0000000... 0123456... A file0
```

and the contents of the deleted file `fileX` is similar enough to the contents of the created file `file0`, then rename detection merges these filepairs and creates:

```
:100644 100644 0123456... 0123456... R100 fileX file0
```

When the `-C` option is used, the original contents of modified files, and deleted files (and also unmodified files, if the `--find-copies-harder` option is used) are considered as candidates of the source files in rename/copy operation. If the input were like these filepairs, that talk about a modified file `fileY` and a newly created file `file0`:

```
:100644 100644 0123456... 1234567... M fileY
:000000 100644 0000000... bcd3456... A file0
```

the original contents of `fileY` and the resulting contents of `file0` are compared, and if they are similar enough, they are changed to:

```
:100644 100644 0123456... 1234567... M fileY
```

```
:100644 100644 0123456... bcd3456... C100 fileY file0
```

In both rename and copy detection, the same "extent of changes" algorithm used in `diffcore-break` is used to determine if two files are "similar enough", and can be customized to use a similarity score different from the default of 50% by giving a number after the `"-M"` or `"-C"` option (e.g. `"-M8"` to tell it to use $8/10 = 80\%$).

Note that when rename detection is on but both copy and break detection are off, rename detection adds a preliminary step that first checks if files are moved across directories while keeping their filename the same. If there is a file added to a directory whose contents are sufficiently similar to a file with the same name that got deleted from a different directory, it will mark them as renames and exclude them from the later quadratic step (the one that pairwise compares all unmatched files to find the "best" matches, determined by the highest content similarity). So, for example, if a deleted `docs/ext.txt` and an added `docs/config/ext.txt` are similar enough, they will be marked as a rename and prevent an added `docs/ext.md` that may be even more similar to the deleted `docs/ext.txt` from being considered as the rename destination in the later step. For this reason, the preliminary "match same filename" step uses a bit higher threshold to mark a file pair as a rename and stop considering other candidates for better matches. At most, one comparison is done per file in this preliminary pass; so if there are several remaining `ext.txt` files throughout the directory hierarchy after exact rename detection, this preliminary step may be skipped for those files.

Note. When the `"-C"` option is used with `--find-copies-harder` option, `git diff-*` commands feed unmodified filepairs to `diffcore` mechanism as well as modified ones. This lets the copy detector consider unmodified files as copy source candidates at the expense of making it slower. Without `--find-copies-harder`, `git diff-*` commands can detect copies only if the file that was copied happened to have been modified in the same changeset.

DIFFCORE-MERGE-BROKEN: FOR PUTTING COMPLETE REWRITES BACK TOGETHER

This transformation is used to merge filepairs broken by `diffcore-break`, and not transformed into rename/copy by `diffcore-rename`, back into a single modification. This always runs when `diffcore-break` is used.

For the purpose of merging broken filepairs back, it uses a different "extent of changes" computation from the ones used by `diffcore-break` and `diffcore-rename`. It counts only the deletion from the original, and does not count insertion. If you removed only 10 lines from a 100-line document, even if you added 910 new lines to make a new 1000-line document, you did not do a complete rewrite. `diffcore-break` breaks such a case in order to help `diffcore-rename` to consider such filepairs as a candidate of rename/copy detection, but if filepairs broken that way were not matched with other filepairs to create rename/copy, then this transformation merges them back into the original

"modification".

The "extent of changes" parameter can be tweaked from the default 80% (that is, unless more than 80% of the original material is deleted, the broken pairs are merged back into a single modification) by giving a second number to `-B` option, like these:

⊕

(give 50% "break score" to `diffcore-break`, use 60% for `diffcore-merge-broken`).

⊕

(the same as above, since `diffcore-break` defaults to 50%).

Note that earlier implementation left a broken pair as separate creation and deletion patches. This was an unnecessary hack, and the latest implementation always merges all the broken pairs back into modifications, but the resulting patch output is formatted differently for easier review in case of such a complete rewrite by showing the entire contents of the old version prefixed with `-`, followed by the entire contents of the new version prefixed with `+`.

DIFFCORE-PICKAXE: FOR DETECTING ADDITION/DELETION OF SPECIFIED STRING

This transformation limits the set of filepairs to those that change specified strings between the preimage and the postimage in a certain way. `-S<block-of-text>` and `-G<regular-expression>` options are used to specify different ways these strings are sought.

`"-S<block-of-text>"` detects filepairs whose preimage and postimage have different number of occurrences of the specified block of text. By definition, it will not detect in-file moves. Also, when a changeset moves a file wholesale without affecting the interesting string, `diffcore-rename` kicks in as usual, and `-S` omits the filepair (since the number of occurrences of that string didn't change in that rename-detected filepair). When used with `--pickaxe-regex`, treat the `<block-of-text>` as an extended POSIX regular expression to match, instead of a literal string.

`"-G<regular-expression>"` (mnemonic: `grep`) detects filepairs whose textual diff has an added or a deleted line that matches the given regular expression. This means that it will detect in-file (or what rename-detection considers the same file) moves, which is noise. The implementation runs diff twice and greps, and this can be quite expensive. To speed things up, binary files without `textconv` filters will be ignored.

When `-S` or `-G` are used without `--pickaxe-all`, only filepairs that match their respective criterion are kept in the output. When `--pickaxe-all` is used, if even one filepair matches their respective criterion in a changeset, the entire changeset is kept. This behavior is designed to make reviewing changes in the context of the whole changeset easier.

DIFFCORE-ORDER: FOR SORTING THE OUTPUT BASED ON FILENAMES

This is used to reorder the filepairs according to the user's (or project's) taste, and is controlled by the `-O` option to the `git diff-*` commands.

This takes a text file each of whose lines is a shell glob pattern. Filepairs that match a glob pattern on an earlier line in the file are output before ones that match a later line, and filepairs that do not match any glob pattern are output last.

As an example, a typical orderfile for the core Git probably would look like this:

```
README
Makefile
Documentation
*.h
*.c
t
```

DIFFCORE-ROTATE: FOR CHANGING AT WHICH PATH OUTPUT STARTS

This transformation takes one pathname, and rotates the set of filepairs so that the filepair for the given pathname comes first, optionally discarding the paths that come before it. This is used to implement the `--skip-to` and the `--rotate-to` options. It is an error when the specified pathname is not in the set of filepairs, but it is not useful to error out when used with "git log" family of commands, because it is unreasonable to expect that a given path would be modified by each and every commit shown by the "git log" command. For this reason, when used with "git log", the filepair that sorts the same as, or the first one that sorts after, the given pathname is where the output starts.

Use of this transformation combined with `diffcore-order` will produce unexpected results, as the input to this transformation is likely not sorted when `diffcore-order` is in effect.

SEE ALSO

git-diff(1), **git-diff-files(1)**, **git-diff-index(1)**, **git-diff-tree(1)**, **git-format-patch(1)**, **git-log(1)**, **gitglossary(7)**, **The Git User's Manual[1]**

GIT

Part of the **git(1)** suite

NOTES

1. The Git User's Manual
`git-htmldocs/user-manual.html`