NAME

git-cat-file - Provide content or type and size information for repository objects

SYNOPSIS

```
git cat-file (-t [--allow-unknown-type]| -s [--allow-unknown-type]| -e | -p | <type>
 | --textconv | --filters ) [--path=<path>] <object>
git cat-file (--batch[=<format>] | --batch-check[=<format>]) [ --textconv | --filter
s ] [--follow-symlinks]
```

DESCRIPTION

In its first form, the command provides the content or the type of an object in the repository. The type is required unless -t or -p is used to find the object type, or -s is used to find the object size, or --textconv or --filters is used (which imply type "blob").

In the second form, a list of objects (separated by linefeeds) is provided on stdin, and the SHA-1, type, and size of each object is printed on stdout. The output format can be overridden using the optional <format> argument. If either --textconv or --filters was specified, the input is expected to list the object names followed by the path name, separated by a single whitespace, so that the appropriate drivers can be determined.

OPTIONS

<object>
The name of the object to show. For a more complete list of ways to spell object names, see the "SPECIFYING REVISIONS" section in gitrevisions(7).

-t
Instead of the content, show the object type identified by <object>.

-s
Instead of the content, show the object size identified by <object>.

-e
Exit with zero status if <object> exists and is a valid object. If <object> is of an invalid format exit with non-zero and emits an error on stderr.

-p
Pretty-print the contents of <object> based on its type.

<type>
Typically this matches the real type of <object> but asking for a type that can trivially be dereferenced from the given <object> is also permitted. An example is to ask for a "tree" with <object> being a commit object that contains it, or to ask for a "blob" with <object> being a tag object that points at it.

--textconv
Show the content as transformed by a textconv filter. In this case, <object> has to be of the form <tree-ish>:<path>, or :<path> in order to apply the filter to the content recorded in the index at <path>.

--filters
Show the content as converted by the filters configured in the current working tree for the given <path> (i.e. smudge filters, end-of-line conversion, etc). In this case, <object> has to be of the form <tree-ish>:<path>, or :<path>.

--path=<path>

For use with --textconv or --filters, to allow specifying an object name and a path separately, e.g. when it is difficult to figure out the revision from which the blob came.

--batch, --batch=<format>
Print object information and contents for each object provided on stdin. May not be combined with any other options or arguments except --textconv or --filters, in which case the input lines also need to specify the path, separated by whitespace. See the section BATCH OUTPUT below for details.

--batch-check, --batch-check=<format>
Print object information for each object provided on stdin. May not be combined with any other options or arguments except --textconv or --filters, in which case the input lines also need to specify the path, separated by whitespace. See the section BATCH OUTPUT below for details.

--batch-all-objects
Instead of reading a list of objects on stdin, perform the requested batch operation on all objects in the repository and any alternate object stores (not just reachable objects). Requires --batch or --batch-check be specified. By default, the objects are visited in order sorted by their hashes; see also --unordered below. Objects are presented as-is, without respecting the "replace" mechanism of git-replace(1).

--buffer
Normally batch output is flushed after each object is output, so that a process can interactively read and write from cat-file. With this option, the output uses normal stdio buffering; this is much more efficient when invoking --batch-check on a large number of objects.

--unordered
When --batch-all-objects is in use, visit objects in an order which may be more efficient for accessing the object contents than hash order. The exact details of the order are unspecified, but if you do not require a specific order, this should generally result in faster output, especially with --batch. Note that cat-file will still show each object only once, even if it is stored multiple times in the repository.

--allow-unknown-type
Allow -s or -t to query broken/corrupt objects of unknown type.

--follow-symlinks
With --batch or --batch-check, follow symlinks inside the repository when requesting objects with extended SHA-1 expressions of the form tree-ish:path-in-tree. Instead of providing output about the link itself, provide output about the linked-to object. If a symlink points outside the tree-ish (e.g. a link to /foo or a root-level link to ../foo), the portion of the link which is outside the tree will be printed.

This option does not (currently) work correctly when an object in the index is specified (e.g. :link instead of HEAD:link) rather than one in the tree.

This option cannot (currently) be used unless --batch or --batch-check is used.

For example, consider a git repository containing:

f: a file containing "hello\n"
link: a symlink to f
dir/link: a symlink to ../f
plink: a symlink to ../f

```
    alink: a symlink to /etc/passwd
```

For a regular file f, echo HEAD:f | git cat-file --batch would print

```
    ce013625030ba8dba906f756967f9e9ca394464a blob 6
```

And echo HEAD:link | git cat-file --batch --follow-symlinks would print the same thing, as would HEAD:dir/link, as they both point at HEAD:f.

Without --follow-symlinks, these would print data about the symlink itself. In the case of HEAD:link, you would see

```
    4d1ae35ba2c8ec712fa2a379db44ad639ca277bd blob 1
```

Both plink and alink point outside the tree, so they would respectively print:

```
    symlink 4
    ../f

    symlink 11
    /etc/passwd
```

OUTPUT

If -t is specified, one of the <type>.

If -s is specified, the size of the <object> in bytes.

If -e is specified, no output, unless the <object> is malformed.

If -p is specified, the contents of <object> are pretty-printed.

If <type> is specified, the raw (though uncompressed) contents of the <object> will be returned.

BATCH OUTPUT

If --batch or --batch-check is given, cat-file will read objects from stdin, one per line, and print information about them. By default, the whole line is considered as an object, as if it were fed to git-rev-parse(1).

You can specify the information shown for each object by using a custom <format>. The <format> is copied literally to stdout for each object, with placeholders of the form %(atom) expanded, followed by a newline. The available atoms are:

objectname
: The full hex representation of the object name.

objecttype
: The type of the object (the same as cat-file -t reports).

objectsize
: The size, in bytes, of the object (the same as cat-file -s reports).

objectsize:disk
: The size, in bytes, that the object takes up on disk. See the note about on-disk sizes in the CAVEATS section below.

deltabase
: If the object is stored as a delta on-disk, this expands to the full hex representation of the delta base object name. Otherwise, expands to the null OID (all zeroes). See CAVEATS below.

rest

If this atom is used in the output string, input lines are split at the first whitespace boundary. All characters before that whitespace are considered to be the object name; characters after that first run of whitespace (i.e., the "rest" of the line) are output in place of the %(rest) atom.

If no format is specified, the default format is %(objectname) %(objecttype) %(objectsize).

If --batch is specified, the object information is followed by the object contents (consisting of %(objectsize) bytes), followed by a newline.

For example, --batch without a custom format would produce:

```
<oid> SP <type> SP <size> LF
<contents> LF
```

Whereas --batch-check='%(objectname) %(objecttype)' would produce:

```
<oid> SP <type> LF
```

If a name is specified on stdin that cannot be resolved to an object in the repository, then cat-file will ignore any custom format and print:

```
<object> SP missing LF
```

If a name is specified that might refer to more than one object (an ambiguous short sha), then cat-file will ignore any custom format and print:

```
<object> SP ambiguous LF
```

If --follow-symlinks is used, and a symlink in the repository points outside the repository, then cat-file will ignore any custom format and print:

```
symlink SP <size> LF
<symlink> LF
```

The symlink will either be absolute (beginning with a /), or relative to the tree root. For instance, if dir/link points to ../../foo, then <symlink> will be ../foo. <size> is the size of the symlink in bytes.

If --follow-symlinks is used, the following error messages will be displayed:

```
<object> SP missing LF
```

is printed when the initial symlink requested does not exist.

```
dangling SP <size> LF
<object> LF
```

is printed when the initial symlink exists, but something that it (transitive-of) points to does not.

```
loop SP <size> LF
<object> LF
```

is printed for symlink loops (or any symlinks that require more than 40 link resolutions to resolve).

```
notdir SP <size> LF
<object> LF
```

is printed when, during symlink resolution, a file is used as a directory name.

CAVEATS

Note that the sizes of objects on disk are reported accurately, but care should be taken in drawing conclusions about which refs or objects are responsible for disk usage. The size of a packed non-delta object may be much larger than the size of objects which delta against it, but the choice of which object is the base and which is the delta is arbitrary and is subject to change during a repack.

Note also that multiple copies of an object may be present in the object database; in this case, it is undefined which copy's size or delta base will be reported.

GIT

Part of the git(1) suite