



Linux Ubuntu 26.04 Manual Pages on command 'git-for-each-ref.1'

\$ man git-for-each-ref.1

GIT-FOR-EACH-REF(1)

Git Manual

GIT-FOR-EACH-REF(1)

NAME

git-for-each-ref - Output information on each ref

SYNOPSIS

```
git for-each-ref [--count=<count>] [--shell|--perl|--python|--tcl]
                [--sort=<key>...] [--format=<format>]
                [--include-root-refs] [--points-at=<object>]
                [--merged[=<object>]] [--no-merged[=<object>]]
                [--contains[=<object>]] [--no-contains[=<object>]]
                [--exclude=<pattern>...] [--start-after=<marker>]
                [ --stdin | (<pattern>...)]
```

DESCRIPTION

Iterate over all refs that match <pattern> and show them according to the given <format>, after sorting them according to the given set of <key>. If <count> is given, stop after showing that many refs. The interpolated values in <format> can optionally be quoted as string literals in the specified host language allowing their direct evaluation in that language.

OPTIONS

<pattern>...

If one or more <pattern> parameters are given, only refs are shown that match against at least one pattern, either using fnmatch(3) or literally, in the latter case matching completely or from the beginning up to a slash.

--stdin

The list of patterns is read from standard input instead of from the argument list.

`--count=<count>`

Stop after showing <count> refs.

`--sort=<key>`

Sort on the field name <key>. Prefix - to sort in descending order of the value. When unspecified, refname is used. You may use the `--sort=<key>` option multiple times, in which case the last key becomes the primary key.

`--format[=<format>]`

A string that interpolates `%(fieldname)` from a ref being shown and the object it points at. In addition, the string literal `%%` renders as `%` and `%xx` - where `xx` are hex digits - renders as the character with hex code `xx`. For example, `%00` interpolates to `\0` (NUL), `%09` to `\t` (TAB), and `%0a` to `\n` (LF).

When unspecified, <format> defaults to `%(objectname) SPC %(objecttype) TAB %(refname)`.

`--color[=<when>]`

Respect any colors specified in the `--format` option. The <when> field must be one of `always`, `never`, or `auto` (if <when> is absent, behave as if `always` was given).

`--shell, --perl, --python, --tcl`

If given, strings that substitute `%(fieldname)` placeholders are quoted as string literals suitable for the specified host language. This is meant to produce a scriptlet that can directly be "eval"ed.

`--points-at=<object>`

Only list refs which points at the given object.

`--merged[=<object>]`

Only list refs whose tips are reachable from the specified commit (HEAD if not specified).

`--no-merged[=<object>]`

Only list refs whose tips are not reachable from <object>(HEAD if not specified).

`--contains[=<object>]`

Only list refs which contain <object>(HEAD if not specified).

`--no-contains[=<object>]`

Only list refs which don't contain <object> (HEAD if not specified).

`--ignore-case`

Sorting and filtering refs are case insensitive.

--omit-empty

Do not print a newline after formatted refs where the format expands to the empty string.

--exclude=<excluded-pattern>

If one or more --exclude options are given, only refs which do not match any <excluded-pattern> parameters are shown. Matching is done using the same rules as <pattern> above.

--include-root-refs

List root refs (HEAD and pseudorefs) apart from regular refs.

--start-after=<marker>

Allows paginating the output by skipping references up to and including the specified marker. When paging, it should be noted that references may be deleted, modified or added between invocations. Output will only yield those references which follow the marker lexicographically. Output begins from the first reference that would come after the marker alphabetically. Cannot be used with --sort=<key> or --stdin options, or the <pattern> argument(s) to limit the refs.

FIELD NAMES

Various values from structured fields in referenced objects can be used to interpolate into the resulting output, or as sort keys.

For all objects, the following names can be used:

refname

The name of the ref (the part after \$GIT_DIR/). For a non-ambiguous short name of the ref append :short. The option core.warnAmbiguousRefs is used to select the strict abbreviation mode. If lstrip=<n> (rstrip=<n>) is appended, strip <n> slash-separated path components from the front (back) of the refname (e.g. %(refname:lstrip=2) turns refs/tags/foo into foo and %(refname:rstrip=2) turns refs/tags/foo into refs). If <n> is a negative number, strip as many path components as necessary from the specified end to leave -<n> path components (e.g. %(refname:lstrip=-2) turns refs/tags/foo into tags/foo and %(refname:rstrip=-1) turns refs/tags/foo into refs). When the ref does not have enough components, the result becomes an empty string if stripping with positive <n>, or it becomes the full refname if stripping with negative <N>. Neither is an error.

strip can be used as a synonym to lstrip.

objecttype

The type of the object (blob, tree, commit, tag).

objectsize

The size of the object (the same as `git cat-file -s` reports). Append `:disk` to get the size, in bytes, that the object takes up on disk. See the note about on-disk sizes in the CAVEATS section below.

objectname

The object name (aka SHA-1). For a non-ambiguous abbreviation of the object name append `:short`. For an abbreviation of the object name with desired length append `:short=<length>`, where the minimum length is `MINIMUM_ABBREV`. The length may be exceeded to ensure unique object names.

deltabase

This expands to the object name of the delta base for the given object, if it is stored as a delta. Otherwise it expands to the null object name (all zeroes).

upstream

The name of a local ref which can be considered "upstream" from the displayed ref. Respects `:short`, `:lstrip` and `:rstrip` in the same way as `refname` above. Additionally respects `:track` to show "[ahead N, behind M]" and `:trackshort` to show the terse version: ">" (ahead), "<" (behind), "<>" (ahead and behind), or "=" (in sync). `:track` also prints "[gone]" whenever unknown upstream ref is encountered. Append `:track,nobacket` to show tracking information without brackets (i.e "ahead N, behind M").

For any remote-tracking branch `%(upstream)`, `%(upstream:remotename)` and `%(upstream:remoteref)` refer to the name of the remote and the name of the tracked remote ref, respectively. In other words, the remote-tracking branch can be updated explicitly and individually by using the refspec `%(upstream:remoteref):%(upstream)` to fetch from `%(upstream:remotename)`.

Has no effect if the ref does not have tracking information associated with it. All the options apart from `nobacket` are mutually exclusive, but if used together the last option is selected.

push

The name of a local ref which represents the `@{push}` location for the displayed ref. Respects `:short`, `:lstrip`, `:rstrip`, `:track`, `:trackshort`, `:remotename`, and `:remoteref` options as `upstream` does. Produces an empty string if no `@{push}` ref is configured.

HEAD

* if HEAD matches current ref (the checked out branch), ' ' otherwise.

color

Change output color. Followed by :<colorname>, where color names are described under Values in the "CONFIGURATION FILE" section of git-config(1). For example, %(color:bold red).

align

Left-, middle-, or right-align the content between %(align:...) and %(end). The "align:" is followed by width=<width> and position=<position> in any order separated by a comma, where the <position> is either left, right or middle, default being left and <width> is the total length of the content with alignment. For brevity, the "width=" and/or "position=" prefixes may be omitted, and bare <width> and <position> used instead. For instance, %(align:<width>,<position>). If the contents length is more than the width then no alignment is performed. If used with --quote everything in between %(align:...) and %(end) is quoted, but if nested then only the topmost level performs quoting.

if

Used as %(if)...%(then)...%(end) or %(if)...%(then)...%(else)...%(end). If there is an atom with value or string literal after the %(if) then everything after the %(then) is printed, else if the %(else) atom is used, then everything after %(else) is printed. We ignore space when evaluating the string before %(then), this is useful when we use the %(HEAD) atom which prints either "*" or " " and we want to apply the if condition only on the HEAD ref. Append ":equals=<string>" or ":notequals=<string>" to compare the value between the %(if:...) and %(then) atoms with the given string.

symref

The ref which the given symbolic ref refers to. If not a symbolic ref, nothing is printed. Respects the :short, :lstrip and :rstrip options in the same way as refname above.

signature

The GPG signature of a commit.

signature:grade

Show

G

for a good (valid) signature

B

for a bad signature

U

for a good signature with unknown validity

X

for a good signature that has expired

Y

for a good signature made by an expired key

R

for a good signature made by a revoked key

E

if the signature cannot be checked (e.g. missing key)

N

for no signature.

signature:signer

The signer of the GPG signature of a commit.

signature:key

The key of the GPG signature of a commit.

signature:fingerprint

The fingerprint of the GPG signature of a commit.

signature:primarykeyfingerprint

The primary key fingerprint of the GPG signature of a commit.

signature:trustlevel

The trust level of the GPG signature of a commit. Possible outputs are ultimate, fully, marginal, never and undefined.

worktreepath

The absolute path to the worktree in which the ref is checked out, if it is checked out in any linked worktree. Empty string otherwise.

ahead-behind:<commit-ish>

Two integers, separated by a space, demonstrating the number of commits ahead and behind, respectively, when comparing the output ref to the <committish> specified in the format.

is-base:<commit-ish>

In at most one row, (<commit-ish>) will appear to indicate the ref that is most likely

the ref used as a starting point for the branch that produced <commit-ish>. This choice is made using a heuristic: choose the ref that minimizes the number of commits in the first-parent history of <commit-ish> and not in the first-parent history of the ref.

For example, consider the following figure of first-parent histories of several refs:



Here, if A, B, and C are the filtered references, and the format string is

%(refname):%(is-base:D), then the output would be

refs/heads/A:

refs/heads/B:(D)

refs/heads/C:

This is because the first-parent history of D has its earliest intersection with the first-parent histories of the filtered refs at a common first-parent ancestor of B and C and ties are broken by the earliest ref in the sorted order.

Note that this token will not appear if the first-parent history of <commit-ish> does not intersect the first-parent histories of the filtered refs.

describe[:<option>,...]

A human-readable name, like git-describe(1); empty string for undescribable commits. The describe string may be followed by a colon and one or more comma-separated options.

tags=<bool-value>

Instead of only considering annotated tags, consider lightweight tags as well; see the corresponding option in git-describe(1) for details.

abbrev=<number>

Use at least <number> hexadecimal digits; see the corresponding option in git-describe(1) for details.

match=<pattern>

Only consider tags matching the glob(7) <pattern>, excluding the refs/tags/ prefix;
see the corresponding option in git-describe(1) for details.

exclude=<pattern>

Do not consider tags matching the glob(7) <pattern>, excluding the refs/tags/
prefix; see the corresponding option in git-describe(1) for details.

In addition to the above, for commit and tag objects, the header field names (tree, parent, object, type, and tag) can be used to specify the value in the header field. Fields tree and parent can also be used with modifier :short and :short=<length> just like objectname.

For commit and tag objects, the special creatordate and creator fields will correspond to the appropriate date or name-email-date tuple from the committer or tagger fields depending on the object type. These are intended for working on a mix of annotated and lightweight tags.

For tag objects, a fieldname prefixed with an asterisk (*) expands to the fieldname value of the peeled object, rather than that of the tag object itself.

Fields that have name-email-date tuple as its value (author, committer, and tagger) can be suffixed with name, email, and date to extract the named component. For email fields (authoremail, committeremail and taggeremail), :trim can be appended to get the email without angle brackets, and :localpart to get the part before the @ symbol out of the trimmed email. In addition to these, the :mailmap option and the corresponding :mailmap,trim and :mailmap,localpart can be used (order does not matter) to get values of the name and email according to the .mailmap file or according to the file set in the mailmap.file or mailmap.blob configuration variable (see gitmailmap(5)).

The raw data in an object is raw.

raw:size

The raw data size of the object.

Note that --format=%(raw) can not be used with --python, --shell, --tcl, because such language may not support arbitrary binary data in their string variable type.

The message in a commit or a tag object is contents, from which contents:<part> can be used to extract various parts out of:

contents:size

The size in bytes of the commit or tag message.

contents:subject

The first paragraph of the message, which typically is a single line, is taken as the "subject" of the commit or the tag message. Instead of contents:subject, field subject can also be used to obtain same results. :sanitize can be appended to subject for subject line suitable for filename.

contents:body

The remainder of the commit or the tag message that follows the "subject".

contents:signature

The optional GPG signature of the tag.

contents:lines=<n>

The first <n> lines of the message.

Additionally, the trailers as interpreted by git-interpret-trailers(1) are obtained as trailers[:<option>,...] (or by using the historical alias contents:trailers[:<option>,...]).

For valid <option> values see trailers section of git-log(1).

For sorting purposes, fields with numeric values sort in numeric order (objectsize, authordate, committerdate, creatordate, taggerdate). All other fields are used to sort in their byte-value order.

There is also an option to sort by versions, this can be done by using the fieldname version:refname or its alias v:refname.

In any case, a field name that refers to a field inapplicable to the object referred by the ref does not cause an error. It returns an empty string instead.

As a special case for the date-type fields, you may specify a format for the date by adding : followed by date format name (see the values the --date option to git-rev-list(1) takes).

If this formatting is provided in a --sort key, references will be sorted according to the byte-value of the formatted string rather than the numeric value of the underlying timestamp.

Some atoms like %(align) and %(if) always require a matching %(end). We call them "opening atoms" and sometimes denote them as %(\$open).

When a scripting language specific quoting is in effect, everything between a top-level opening atom and its matching %(end) is evaluated according to the semantics of the opening atom and only its result from the top-level is quoted.

EXAMPLES

An example directly producing formatted text. Show the most recent 3 tagged commits:

```
#!/bin/sh
```

```

git for-each-ref --count=3 --sort='-*authordate' \
`--format='From: %(*authorname) %(*authoremail)

Subject: %(*subject)

Date: %(*authordate)

Ref: %(*refname)

%(*body)

' 'refs/tags'

```

A simple example showing the use of shell eval on the output, demonstrating the use of

--shell. List the prefixes of all heads:

```

#!/bin/sh

git for-each-ref --shell --format="ref=%(refname)" refs/heads | \
while read entry
do
    eval "$entry"
    echo `dirname $ref`
done

```

A bit more elaborate report on tags, demonstrating that the format may be an entire script:

```

#!/bin/sh

fmt='

r=%(refname)

t=%(*objecttype)

T=${r#refs/tags/}

o=%(*objectname)

n=%(*authorname)

e=%(*authoremail)

s=%(*subject)

d=%(*authordate)

b=%(*body)

kind=Tag

if test "z$t" = z

then

    # could be a lightweight tag

    t=%(objecttype)

```

```

        kind="Lightweight tag"

        o=%(objectname)

        n=%(authorname)

        e=%(authoremail)

        s=%(subject)

        d=%(authordate)

        b=%(body)

    fi

    echo "$kind $T points at a $t object $o"

    if test "z$t" = zcommit

    then

        echo "The commit was authored by $n $e

at $d, and titled

        $s

Its message reads as:

"

        echo "$b" | sed -e "s/^/ /"

        echo

    fi

,

eval=`git for-each-ref --shell --format="$fmt" \

--sort='*objecttype' \

--sort=-taggerdate \

refs/tags`

eval "$eval"

```

An example to show the usage of %(if)...%(then)...%(else)...%(end). This prefixes the current branch with a star.

```
git for-each-ref --format="%(if)%(HEAD)%(then)* %(else) %(end)%(refname:short)" refs/heads/
```

An example to show the usage of %(if)...%(then)...%(end). This prints the authorname, if present.

```
git for-each-ref --format="%(refname)%(if)%(authorname)%(then) Authored by: %(authorname)%(end)"
```

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.

NOTES

When combining multiple `--contains` and `--no-contains` filters, only references that contain at least one of the `--contains` commits and contain none of the `--no-contains` commits are shown.

When combining multiple `--merged` and `--no-merged` filters, only references that are reachable from at least one of the `--merged` commits and from none of the `--no-merged` commits are shown.

SEE ALSO

`git-show-ref(1)`

GIT

Part of the `git(1)` suite

Git 2.53.0

03/02/2026

GIT-FOR-EACH-REF(1)