NAME

git-cherry − Find commits yet to be applied to upstream

SYNOPSIS

git cherry [−v] [<upstream> [<head> [<limit>]]]

DESCRIPTION

Determine whether there are commits in **<head>..<upstream>** that are equivalent to those in the range **<limit>..<head>**.

The equivalence test is based on the diff, after removing whitespace and line numbers. git−cherry therefore detects when commits have been "copied" by means of **git-cherry-pick**(1), **git-am**(1) or **git-rebase**(1).

Outputs the SHA1 of every commit in **<limit>..<head>**, prefixed with **−** for commits that have an equivalent in <upstream>, and **+** for commits that do not.

OPTIONS

−v
: Show the commit subjects next to the SHA1s.

<upstream>
: Upstream branch to search for equivalent commits. Defaults to the upstream branch of HEAD.

<head>
: Working branch; defaults to HEAD.

<limit>
: Do not report commits up to (and including) limit.

EXAMPLES

Patch workflows

git−cherry is frequently used in patch−based workflows (see **gitworkflows**(7)) to determine if a series of patches has been applied by the upstream maintainer. In such a workflow you might create and send a topic branch like this:

```
$ git checkout −b topic origin/master
# work and create some commits
$ git format−patch origin/master
$ git send−email ... 00*
```

Later, you can see whether your changes have been applied by saying (still on **topic**):

```
$ git fetch  # update your notion of origin/master
$ git cherry −v
```

Concrete example

In a situation where topic consisted of three commits, and the maintainer applied two of them, the situation might look like:

```
$ git log −−graph −−oneline −−decorate −−boundary origin/master...topic
* 7654321 (origin/master) upstream tip commit
[... snip some other commits ...]
* cccc111 cherry−pick of C
* aaaa111 cherry−pick of A
[... snip a lot more that has happened ...]
| * cccc000 (topic) commit C
```

```
| * bbbb000 commit B
| * aaaa000 commit A
|/
o 1234567 branch point
```

In such cases, git−cherry shows a concise summary of what has yet to be applied:

```
$ git cherry origin/master topic
− cccc000... commit C
+ bbbb000... commit B
− aaaa000... commit A
```

Here, we see that the commits A and C (marked with **−**) can be dropped from your **topic** branch when you rebase it on top of **origin/master**, while the commit B (marked with **+**) still needs to be kept so that it will be sent to be applied to **origin/master**.

Using a limit

The optional <limit> is useful in cases where your topic is based on other work that is not in upstream. Expanding on the previous example, this might look like:

```
$ git log −−graph −−oneline −−decorate −−boundary origin/master...topic
* 7654321 (origin/master) upstream tip commit
[... snip some other commits ...]
* cccc111 cherry−pick of C
* aaaa111 cherry−pick of A
[... snip a lot more that has happened ...]
| * cccc000 (topic) commit C
| * bbbb000 commit B
| * aaaa000 commit A
| * 0000fff (base) unpublished stuff F
[... snip ...]
| * 0000aaa unpublished stuff A
|/
o 1234567 merge−base between upstream and topic
```

By specifying **base** as the limit, you can avoid listing commits between **base** and **topic**:

```
$ git cherry origin/master topic base
− cccc000... commit C
+ bbbb000... commit B
− aaaa000... commit A
```

SEE ALSO

git-patch-id(1)

GIT

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