## NAME

git-bundle - Move objects and refs by archive

## SYNOPSIS

*git bundle* create <file> <git-rev-list-args>
*git bundle*v erify <file>
*git bundle*list-heads <file> [<refname>...]
*git bundle*un bundle <file> [<refname>...]

## DESCRIPTION

Some workflows require that one or more branches of development on one machine be replicated on another machine, but the two machines cannot be directly connected, and therefore the interactive Git protocols (git, ssh, rsync, http) cannot be used. This command provides support for *git fetch* and *git pull* to operate by packaging objects and references in an archive at the originating machine, then importing those into another repository using *git fetch* and *git pull* after moving the archive by some means (e.g., by sneakernet). As no direct connection between the repositories exists, the user must specify a basis for the bundle that is held by the destination repository: the bundle assumes that all objects in the basis are already in the destination repository.

## OPTIONS

create <file>

Used to create a bundle named *file*. This requires the *git-rev-list-args* arguments to define the bundle contents.

verify <file>

Used to check that a bundle file is valid and will apply cleanly to the current repository. This includes checks on the bundle format itself as well as checking that the prerequisite commits exist and are fully linked in the current repository. *git bundle* prints a list of missing commits, if any, and exits with a non-zero status.

list-heads <file>

Lists the references defined in the bundle. If followed by a list of references, only references matching those given are printed out.

unbundle <file>

Passes the objects in the bundle to *git index-pack* for storage in the repository, then prints the names of all defined references. If a list of references is given, only references matching those in the list are printed. This command is really plumbing, intended to be called only by *git fetch*.

<git-rev-list-args>

A list of arguments, acceptable to *git rev-parse* and *git rev-list* (and containing a named ref, see SPECIFYING REFERENCES below), that specifies the specific objects and references to transport. For example, master~10..master causes the current master reference to be packaged along with all objects added since its 10th ancestor commit. There is no explicit limit to the number of references and objects that may be packaged.

[<refname>...]

A list of references used to limit the references reported as available. This is principally of use to *git fetch*, which expects to receive only those references asked for and not necessarily everything in the pack (in this case, *git bundle* acts like *git fetch-pack*).

## SPECIFYING REFERENCES

*git bundle* will only package references that are shown by *git show-ref*: this includes heads, tags, and remote heads. References such as master~1 cannot be packaged, but are perfectly suitable for defining the basis. More than one reference may be packaged, and more than one basis can be specified. The objects packaged are those not contained in the union of the given bases. Each basis can be specified explicitly (e.g. ^master~10), or implicitly (e.g. master~10..master,

--since=10.days.ago master).

It is very important that the basis used be held by the destination. It is okay to err on the side of caution, causing the bundle file to contain objects already in the destination, as these are ignored when unpacking at the destination.

## EXAMPLE

Assume you want to transfer the history from a repository R1 on machine A to another repository R2 on machine B. For whatever reason, direct connection between A and B is not allowed, but we can move data from A to B via some mechanism (CD, email, etc.). We want to update R2 with development made on the branch master in R1.

To bootstrap the process, you can first create a bundle that does not have any basis. You can use a tag to remember up to what commit you last processed, in order to make it easy to later update the other repository with an incremental bundle:

```
machineA$ cd R1
machineA$ git bundle create file.bundle master
machineA$ git tag -f lastR2bundle master
```

Then you transfer file.bundle to the target machine B. Because this bundle does not require any existing object to be extracted, you can create a new repository on machine B by cloning from it:

```
machineB$ git clone -b master /home/me/tmp/file.bundle R2
```

This will define a remote called origin in the resulting repository that lets you fetch and pull from the bundle. The $GIT_DIR/config file in R2 will have an entry like this:

```
[remote origin]
url = /home/me/tmp/file.bundle
fetch = refs/heads/*:refs/remotes/origin/*
```

To update the resulting mine.git repository, you can fetch or pull after replacing the bundle stored at /home/me/tmp/file.bundle with incremental updates.

After working some more in the original repository, you can create an incremental bundle to update the other repository:

```
machineA$ cd R1
machineA$ git bundle create file.bundle lastR2bundle..master
machineA$ git tag -f lastR2bundle master
```

You then transfer the bundle to the other machine to replace /home/me/tmp/file.bundle, and pull from it.

```
machineB$ cd R2
machineB$ git pull
```

If you know up to what commit the intended recipient repository should have the necessary objects, you can use that knowledge to specify the basis, giving a cut-off point to limit the revisions and objects that go in the resulting bundle. The previous example used the lastR2bundle tag for this purpose, but you can use any other options that you would give to the **git-log(1)** command. Here are more examples:

You can use a tag that is present in both:

```
$ git bundle create mybundle v1.0.0..master
```

You can use a basis based on time:

```
$ git bundle create mybundle --since=10.days master
```

You can use the number of commits:

```
$ git bundle create mybundle -10 master
```

You can run git-bundle verify to see if you can extract from a bundle that was created with a basis:

$ git bundle verify mybundle

This will list what commits you must have in order to extract from the bundle and will error out if you do not have them.

A bundle from a recipient repository's point of view is just like a regular repository which it fetches or pulls from. You can, for example, map references when fetching:

$ git fetch mybundle master:localRef

You can also see what references it offers:

$ git ls-remote mybundle

## GIT

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