

NAME

gitnamespaces – Git namespaces

SYNOPSIS

```
GIT_NAMESPACE=<namespace> git upload-pack
GIT_NAMESPACE=<namespace> git receive-pack
```

DESCRIPTION

Git supports dividing the refs of a single repository into multiple namespaces, each of which has its own branches, tags, and HEAD. Git can expose each namespace as an independent repository to pull from and push to, while sharing the object store, and exposing all the refs to operations such as **git-gc(1)**.

Storing multiple repositories as namespaces of a single repository avoids storing duplicate copies of the same objects, such as when storing multiple branches of the same source. The alternates mechanism provides similar support for avoiding duplicates, but alternates do not prevent duplication between new objects added to the repositories without ongoing maintenance, while namespaces do.

To specify a namespace, set the **GIT_NAMESPACE** environment variable to the namespace. For each ref namespace, Git stores the corresponding refs in a directory under **refs/namespaces/**. For example, **GIT_NAMESPACE=foo** will store refs under **refs/namespaces/foo/**. You can also specify namespaces via the **--namespace** option to **git(1)**.

Note that namespaces which include a / will expand to a hierarchy of namespaces; for example, **GIT_NAMESPACE=foo/bar** will store refs under **refs/namespaces/foo/refs/namespaces/bar/**. This makes paths in **GIT_NAMESPACE** behave hierarchically, so that cloning with **GIT_NAMESPACE=foo/bar** produces the same result as cloning with **GIT_NAMESPACE=foo** and cloning from that repo with **GIT_NAMESPACE=bar**. It also avoids ambiguity with strange namespace paths such as **foo/refs/heads/**, which could otherwise generate directory/file conflicts within the **refs** directory.

git-upload-pack(1) and **git-receive-pack(1)** rewrite the names of refs as specified by **GIT_NAMESPACE**. **git-upload-pack** and **git-receive-pack** will ignore all references outside the specified namespace.

The smart HTTP server, **git-http-backend(1)**, will pass **GIT_NAMESPACE** through to the backend programs; see **git-http-backend(1)** for sample configuration to expose repository namespaces as repositories.

For a simple local test, you can use **git-remote-ext(1)**:

```
git clone ext::'git --namespace=foo %s /tmp/prefixed.git'
```

SECURITY

The fetch and push protocols are not designed to prevent one side from stealing data from the other repository that was not intended to be shared. If you have private data that you need to protect from a malicious peer, your best option is to store it in another repository. This applies to both clients and servers. In particular, namespaces on a server are not effective for read access control; you should only grant read access to a namespace to clients that you would trust with read access to the entire repository.

The known attack vectors are as follows:

1. The victim sends "have" lines advertising the IDs of objects it has that are not explicitly intended to be shared but can be used to optimize the transfer if the peer also has them. The attacker chooses an object ID X to steal and sends a ref to X, but isn't required to send the content of X because the victim already has it. Now the victim believes that the attacker has X, and it sends the content of X back to the attacker later. (This attack is most straightforward for a client to perform

on a server, by creating a ref to X in the namespace the client has access to and then fetching it. The most likely way for a server to perform it on a client is to "merge" X into a public branch and hope that the user does additional work on this branch and pushes it back to the server without noticing the merge.)

2. As in #1, the attacker chooses an object ID X to steal. The victim sends an object Y that the attacker already has, and the attacker falsely claims to have X and not Y, so the victim sends Y as a delta against X. The delta reveals regions of X that are similar to Y to the attacker.