Commit exists on GitHub but not in my local clone

In this article

Sometimes a commit will be viewable on GitHub Enterprise Server, but will not exist in your local clone of the repository.

When you use `git show` to view a specific commit on the command line, you may get a fatal error.

For example, you may receive a `bad object` error locally:

```
$ git show 1095ff3d0153115e75b7bca2c09e5136845b5592
> fatal: bad object 1095ff3d0153115e75b7bca2c09e5136845b5592
```

However, when you view the commit on your GitHub Enterprise Server instance, you'll be able to see it without any problems:

`github.com/$account/$repository/commit/1095ff3d0153115e75b7bca2c09e5136845b5592`

There are several possible explanations:

- The local repository is out of date.
- The branch that contains the commit was deleted, so the commit is no longer referenced.
- Someone force pushed over the commit.

The local repository is out of date

Your local repository may not have the commit yet. To get information from your remote repository to your local clone, use `git fetch`:

```
git fetch REMOTE
```

This safely copies information from the remote repository to your local clone without making any changes to the files you have checked out. You can use `git fetch upstream` to get information from a repository you've forked, or `git fetch origin` to get information from a repository you've only cloned.

Tip: For more information, read about managing remotes and fetching data in the Pro Git book.

The branch that contained the commit was deleted

If a collaborator on the repository has deleted the branch containing the commit or has force pushed over the branch, the missing commit may have been orphaned (i.e. it cannot be reached from any reference) and therefore will not be fetched into your local clone.

Fortunately, if any collaborator has a local clone of the repository with the missing commit, they can push it back to GitHub Enterprise Server. They need to make sure the commit is referenced by a local branch and then push it as a new branch to GitHub Enterprise Server.

Let's say that the person still has a local branch (call it `B`) that contains the commit. This might be tracking the branch that was force pushed or deleted and they simply haven't updated yet. To preserve the commit, they can push that local branch to a new branch (call it `recover-B`) on GitHub Enterprise Server. For this example, let's assume they have a remote named `upstream` via which they have push access to `github.com/$account/$repository`.

The other person runs:

```
$ git branch recover-B B
# Create a new local branch referencing the commit
$ git push upstream B:recover-B
# Push local B to new upstream branch, creating new reference to commit
```

Now, *you* can run:

```
$ git fetch upstream recover-B
# Fetch commit into your local repository.
```

Avoid force pushes

Avoid force pushing to a repository unless absolutely necessary. This is especially true if more than one person can push to the repository. If someone force pushes to a repository, the force push may overwrite commits that other people based their work on. Force pushing changes the repository history and can corrupt pull requests.

Further reading

- "Working with Remotes" from the *Pro Git* book
- "Data Recovery" from the *Pro Git* book