Alexander Russel

Ms. Coleman

Operating Systems

21 April 2016

Git for Windows Erroneously Creates Alternate Data Streams

Git behaves strangely when we are talking about files which names contain a colon.

Except the original prefix of the disk (for example, C: \), Windows does not allow other colons in the file name or path. In Unix, there is no such restriction. The test repository has been created to check it, containing one file “foo:bar”, containing “hello”. Cloning the repository with the standard version of Git for Windows does not give any errors or warnings:

```
C:\src
> git clone https://github.com/latkin/filetest.git
Cloning into 'filetest'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

But, instead of a file named foo:bar, you get an empty file foo:

```
C:\src
> cd .\filetest\
C:\src\filetest
```

```
> dir -force


    Directory: C:\src\filetest

Mode                 LastWriteTime         Length Name

----                 -------------         ------ ----

d--h--        7/17/2016   5:53 PM                .git

-a----        7/17/2016   5:47 PM              0 foo
```

This is strange, but even more interesting is that Git has a different look at this:

```
C:\src\filetest

> git status

On branch master

Your branch is up-to-date with 'origin/master'.

Untracked files:

  (use "git add <file>..." to include in what will be committed)


        foo


nothing added to commit but untracked files present (use "git add" to track)
```

Git notices the the untracked file foo but believes that the foo:bar is also present and contains the original string. Things get even stranger when you activate the core.fscache option

(which is enabled by default in versions from 2.8.2 and higher). Then foo:bar is recorded as missing:

```
C:\src\filetest
> git config core.fscache true
C:\src\filetest
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    foo:bar

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        foo

no changes added to commit (use "git add" and/or "git commit -a")
```

Why does it happen like this? The main reason for this is quite a non-obvious feature of the NTFS, called "alternative data streams."

Briefly, the files in NTFS are not just data but the sets of one or more data streams. What we believe to be the content of the file, in fact, is the content of he main, unnamed stream. Also, the data can be stored in the other, named streams. These flows can be directly accessed adding **`:streamname`** to the usual file path.

So, although the **`foo:bar`** is not a valid file name on Windows, the Windows file API is glad to take it for read/write operations, as it is quite a normal “way” to put something in the file system, in this case – an alternative flow **`bar`** of the **`foo`** file.

So, all this is quite silly. Git has to find the wrong file name, give an error and not try to write a file to disk. The valid data is processed in Unix like this (for example, a file named \ Windows \ System32 \ crypt32.dll will be blocked). These files will be processed correctly irrespective of the core.fscache option.