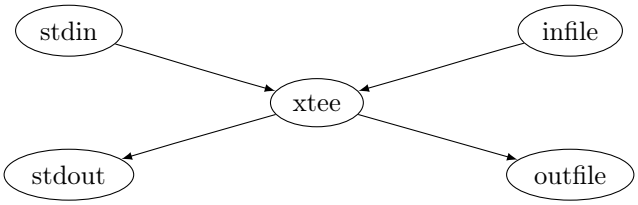
xtee 0.1

Chris Forno (jekor)

March 23, 2008

1 Basic Usage

xtee ("cross-tee"/"expanded tee") is a program for building complex pipelines. It resembles the `tee` command, except that instead of copying stdin to stdout, it copies a file to stdout.



A simple demonstration of xtee:

```
tty1$ mkfifo /tmp/fifo-in /tmp/fifo-out
tty1$ echo "hi from tty1" > /tmp/fifo-in
tty2$ cat /tmp/fifo-out
tty3$ echo "hi from tty3" | xtee -i /tmp/fifo-in -o /tmp/fifo-out
```

When you run the last command, `hi from tty1` will appear on tty3 and `hi from tty3` on tty2.

2 Example: Bidirectional HTTP Filtering

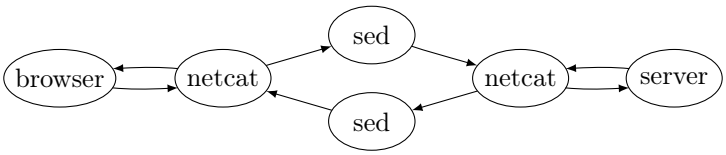
Suppose that Jim, a web developer, is working on his new client's website: xt-shirts.com, a site decidated to extra large t-shirts. Jim doesn't want to make changes to the live site without testing them locally first. So he sets up a web server that emulates the live server.

This works OK for a while, but one day Jim begins testing the shopping cart. He realizes that it's going to be difficult to keep track of which site's cookies he has loaded in which browser (both the live and test site set cookies for the domain `xt-shirts.com`). Since the domain name is hard-coded all throughout the code, it would take a lot of search and replace to have each location read from a global domain name variable. Jim also knows that even simple changes

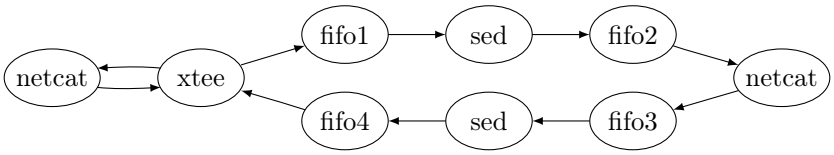
like that could have strange side-effects. And there's no guarantees that another developer won't add more hard-coded domain names to the code later.

Jim realizes that he can use a proxy to replace `xt-shirts.com` with `xt-shirts.test`. Then he won't have a problem with cookies. And as a bonus, he won't ever get confused about whether he's viewing the live site or the test version.

Jim's frustrated with all the HTTP proxies out there because they don't support filtering in both directions and/or filtering the HTTP headers (where the cookies are set, not to mention redirects!). After thinking about it for a while, Jim comes up with the idea to combine some simple programs to make the proxy himself:



The only problem is that netcat can't pass data into one program and return it from another (i.e. into the first sed and back from the second). Using just 1 sed wouldn't work either for the same reason. Luckily, he knows of xtee and refines his netcat pipeline to look like:



Jim decides to test the idea on the test web server. First, he sets the HTTP server to only listen for connections on port 80 of the loopback interface (127.0.0.1). Next, he runs the following commands on the test server.

```
tty1$ mkfifo /tmp/fifo1 /tmp/fifo2 /tmp/fifo3 /tmp/fifo4
tty1$ sed -e 's/xt-shirts\.com/xt-shirts.lan/g' < /tmp/fifo1 \
      > /tmp/fifo2
tty2$ nc 127.0.0.1 80 < /tmp/fifo2 > /tmp/fifo3
tty3$ sed -e 's/xt-shirts\.lan/xt-shirts.com/g' < /tmp/fifo3 \
      > /tmp/fifo4
tty4$ nc -l -p 80 10.0.0.3 -c "xtee -o /tmp/fifo1 -i /tmp/fifo4"
```

He then adds 10.0.0.3 (the address of the test server) to the `/etc/hosts` file on his computer (the one he uses his web browser from) and points his browser to `xt-shirts.lan`...and it works!

Note that Jim uses the `.lan` suffix instead of `.test` so that he doesn't change the `Content-Length` of the body of any HTTP transactions (which might confuse web browsers).

The only problem with the setup is that xtee and the first sed exit after each request in the HTTP session, but he sets up a script to respawn them. [1]

3 Source Code

```
import System.IO
import System.Console.GetOpt
import System.Environment
import System.Exit
import Control.Concurrent
import Control.Monad


version = 0.1
usage   = "Usage: xtee [option...] -i input-file -o output-file"
```

All of the following is just for handling commandline options.

```
data Flag = Help | Version | InFile String | OutFile String
  deriving (Show, Eq)


options :: [OptDescr Flag]
options = [
  Option ['h'] ["help"]    (NoArg Help)            "show help",
  Option ['v'] ["version"] (NoArg Version)         "show version",
  Option ['i'] ["input"]   (ReqArg InFile  "FILE") "input file",
  Option ['o'] ["output"]  (ReqArg OutFile "FILE") "output file"]


xteeOptions :: [String] → IO ([Flag])
xteeOptions argv =
  case getOpt Permute options argv of
    ([], _, [])    → ioError (userError ("\n" ++ showUsage))
    (o, _, [])     → if Help ∈ o then
                       putStr showUsage ≫ exitWith ExitSuccess
                     else
                       if Version ∈ o then
                         putStr (showVersion ++ "\n") ≫ exitWith ExitSuccess
                       else
                         return o
```

[1] The reason that xtee terminates is because it sees the `EOF` marker at the end of the HTTP request. Once xtee exits, the first sed sees the end of its input and exits as well. Each netcat will remain running for the entire HTTP session because that's how netcat was designed. (They also keep the second sed running.) Jim makes a note to request a "daemon" mode from the xtee author.

```
    (_, _, errs) → ioError (userError (concat errs ++ showUsage))
  where showVersion = "xtee version " ++ show version
        usageHeader = unlines [showVersion, usage]
        showUsage   = usageInfo usageHeader options
```

xtee copies its stdin to a file and copies another file to its stdout. We first spawn a process (a Haskell process, not an OS process) to pass stdin through to the output file and then we pass the input file though to stdout. To make sure that both complete before we exit, we use the `wait` MVar.

```
xtee      :: IO Handle → IO Handle → IO ()
xtee i o = newEmptyMVar >>= λwait →
             forkIO ((passThru (return stdin) o) >> putMVar wait ()) >>
             passThru i (return stdout) >> takeMVar wait
```

We want to copy input from one file handle to another unmodified (in text mode). We can leave the buffering up to the Haskell runtime (GHC will use block buffering by default). Since *hGetContents* reads from the handle (i') lazily, this is very simple.

```
passThru :: IO Handle → IO Handle → IO ()
passThru i o = i >>= λi' → o >>= λo' →
                 hGetContents i' >>= hPutStr o' >> hFlush o'
```

Now all we have to do is grab the input and output filenames from the command-line and pass the open file handles to *xtee*.

```
main = getArgs >>= xteeOptions >>= λopts →
  let inF     = head [f | InFile   f ← opts]
      outF    = head [f | OutFile f ← opts]
      inFile  = openFile inF  ReadMode
      outFile = openFile outF AppendMode in
  xtee inFile outFile
```