


2020年8月21日 投稿者: SATOXITS

# GShell 0.1.7 – 続リモートシェル化

開発：いやもう昨日はいったいどうなることかと。

社長：出だしの5ホールの時点ではものすごい結末もよぎりましたが。

開発：稀勢の里の連続優勝後のストーリーを思い出しちゃいましたよ。

基盤：4日間楽しめると良いですね。

＊　＊　＊

社長：今日はどこから行きましょうか。

開発：優先度は、リモートホストでやることの多い作業の効率化に資する事ですね。かつ、なんか実装面で面白い事。

基盤：うちでは大半が、ローカルとの間のファイルのコピー作業ですね。あとはログのtail -f (^-^)。たまにコンテンツの編集。ごくまれに設定ファイルの編集。pingやtcpdumpもたまにやりますね。

開発：tail -f とかtcpdump以外は本来、手元で管理していて、ファイルをリモートコピーすれば出来ることではあります。ファイルをコピーして、あとはなんかプログラムを叩くだけ。tail -f みたいなのは、NFS的に実現するか、ログインして tail -fコマンドを実行するかですね。あるいは、HTTPで実現する。

基盤：tcpdumpとかはどうなるんでしょう？仮想的なファイルに見せるとか？

社長：まあ他に比べると、ログインして tail -f があまりにも簡単ですね。リモートシェルのキラー機能じゃないですかねｗ

開発：思うにFTPにそういう機能があればよいのだと思いますけどね。ストーリームからの入力というか。

基盤：named pipe に tail -f の結果を流すプログラムを貼り付けるとかすればできるんじゃないですかね。

社長：というか、コマンドを実行して標準出力をダウンロード・アップロードするという機能があれば、それだけでいいんじゃないですかね。

開発：その案、乗った！ftpコマンド風には、get {tail -f file} /dev/ttyxx みたいにできると良いですね。でこれをGoルーチンでバックグラウンドにやる。

基盤：get {tar cfz – dir} {tar xfz -} なんていうのも良いですね。put file.tar.gz {tar xfz -} とかも。

社長：うーん。今日はそれをまずやりますか。

＊　＊　＊

開発：まず、昨日の作業のまとめす。

開発：単発のファイルのアップロードとダウンロードは出来ましたが、RTT 250msの遠隔とやるとオーバヘッドが1秒かかってしまいます。これは原理的にも、0.5秒を切ることはできないでしょう。リモートには100万個単位のファイルがあります。多数のファイルをサイト間でコピーするのにこれでは使い物になりません。

基盤：ns1.its-more.jp には147万個のファイルがありますね。

```
jp1$ df -i
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
devtmpfs        123468     433  123035    1% /dev
tmpfs           126141       1  126140    1% /dev/shm
/dev/xvda1     2621440 1473712 1147728   57% /
```

基盤：そのうち83万はカウンターのファイルですが。

```
jp1$ time find counts | wc
 838565  838565 137079105

real    2m8.682s
user    0m3.643s
sys     0m7.679s
```

基盤：なんでこんなに沢山あるんですかね？

社長：ああそれは、DeleGateが1つのURLごとにカウンタファイルを一つ作るからです。動的に生成するコンテンツもあるし、我社の所有する100近いドメイン名から同じコンテンツをアクセスされたのが、URLは違うのでそれぞれ違うカウンターファイルになる。80万ファイル程度で収まっているのは少ないくらいだと思います。

基盤：しかし＄5ライトセールのinodeは250万しか無いようです。結構ヤバいかと。

社長：なんでハッシュファイルにしなかったんでしたっけ。

開発：クラッシュして壊れるのが嫌だったように思いますね。dbm とか ndbm とかの過渡期で移植性の問題もあったような。あとは手作業で個々のカウンターの中身を見られるということを優先したような気も。

基盤：しかも個々のファイルは256バイトの固定長です。今日びのファイルのブロックサイズは4KBか8KBだと思いますから、ほとんどフラグメントですね。95%以上無駄。

社長：個別ファイルのカウンターを dbm 型式に圧縮というかアーカイブして、過去のカウンターセットにアクセスできると良さそうですね。カウンターファイル間の演算ができれば、色んな期間のビューで見られるし。

開発：いずれにしても今後、DeleGate自体のカウンター機能に手を入れることはありません。やるなら、GoでDeleGateをラッピングするかフックするかして、Goで作ったカウンターで置き換えたいと思います。

社長：GShellの機能としてカウンターがあっても良さそうですね。でそれを呼び出す。ヒストリ機能も汎用化して作ると良いかもしれない。

開発：ftp的な機能の範疇ではないですが、カウターファイルをカウントアップするみたいなコマンドがあると良さそうに思います。

＊　＊　＊

開発：で、昨日の作業のまとめです。

開発：まずファイル転送用のコマンドですが、rcp や scp にならって host:file 型式でファイルを指定できるようにしました。コマンド名は gcp。ポート番号も指定したいので、host:port:file というのも使えるようにしました。省略規則は [host]:[port:][path] です。アップロード先が前と同じホストの手元と同じファイル名なら：だけでOKです。

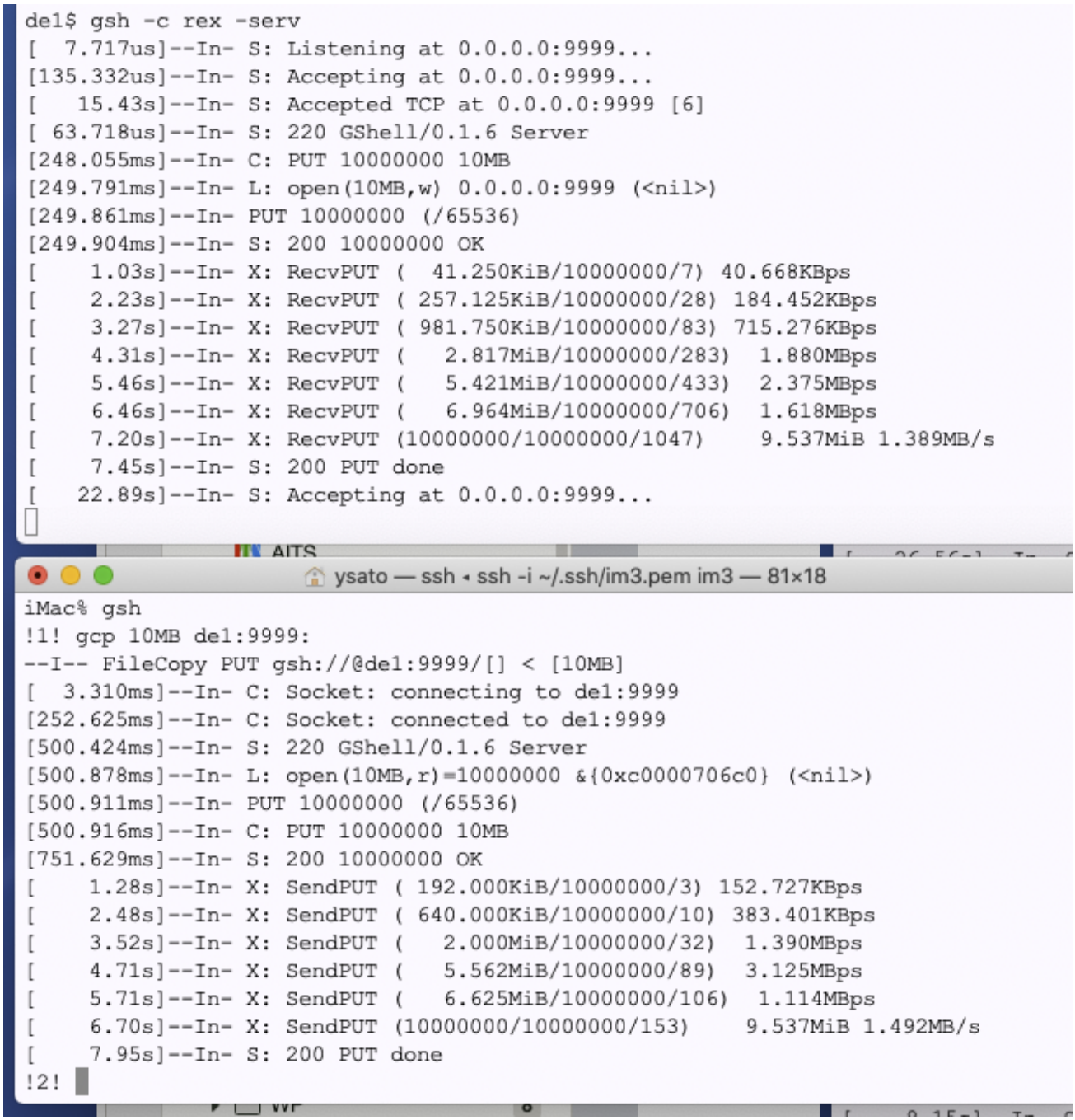
基盤：フルのURLが gsh://host:port/path だとすると、省略規則は [gsh:][//][host]:[[port]/]path とかですかね？

社長：ローカルファイルも前と同じみたいな指定ができると良いように思いますが。まあヒストリを参照すれば良いのかな。というか、ほとんどの場合、特定のひとつのサーバにログインしていて、そこにアップロードするんでしょうから、繰り返す場合にはアップロードは gput src [dst]、ダウンロードは gget dst [src] で良いように思いますね。

開発：まあそのほうが簡明ですね。

基盤：賛成。

開発：それで、最大の問題の遠隔へのアップロードですが。フランクフルト宛てではこう。




開発：一方ダウンロードではこうです。

```
!3! gcp :10MB
--I-- FileCopy GET gsh://@de1:9999/[10MB] > []
[ 38.310ms]--In- C: Socket: connecting to de1:9999
[289.123ms]--In- C: Socket: connected to de1:9999
[539.171ms]--In- S: 220 GShell/0.1.6 Server
[539.482ms]--In- C: GET 10MB
[789.777ms]--In- S: 200 10000000
[    1.00s]--In- X: RecvGET ( 218.625KiB/10000000/50) 223.570KBps
[    2.00s]--In- X: RecvGET (   1.207MiB/10000000/270)  1.036MBps
[    3.00s]--In- X: RecvGET (   3.549MiB/10000000/797)  2.453MBps
[    4.00s]--In- X: RecvGET (   6.479MiB/10000000/1376)  3.070MBps
[    4.78s]--In- X: RecvGET (10000000/10000000/1921)    9.537MiB 2.091MB/s
[    5.57s]--In- S: 200 GET done
!4! gcp :10MB
```

開発：最速で、上り 1.5MB/s、下り 2.1MB/s というところです。

基盤：今日は下りも遅いですね。下りは 5MB/s 近く出ることも多いですが。

社長：ここはぜひ、UDPを使った当社の驚速化技術を投入したいところです。

開発：一方そのころ当社よろずサーバ jp1 @ライトセール東京。

```
iMac% gsh -c gcp 10MB jp1:9999:
--I-- FileCopy PUT gsh://@jp1:9999/[] < [10MB]
[  3.068ms]--In- C: Socket: connecting to jp1:9999
[ 11.109ms]--In- C: Socket: connected to jp1:9999
[ 18.418ms]--In- S: 220 GShell/0.1.6 Server
[ 18.756ms]--In- L: open(10MB,r)=10000000 &{0xc000122660} (<nil>)
[ 18.788ms]--In- PUT 10000000 (/65536)
[ 18.790ms]--In- C: PUT 10000000 10MB
[ 26.941ms]--In- S: 200 10000000 OK
[248.974ms]--In- X: SendPUT (10000000/10000000/153)    9.537MiB 40.165MB/s
[296.382ms]--In- S: 200 PUT done
```

```
iMac% gsh -c gcp jp1:9999:10MB
--I-- FileCopy GET gsh://@jp1:9999/[10MB] > []
[  2.724ms]--In- C: Socket: connecting to jp1:9999
[ 10.838ms]--In- C: Socket: connected to jp1:9999
[ 17.893ms]--In- S: 220 GShell/0.1.6 Server
[ 18.193ms]--In- C: GET 10MB
[ 25.085ms]--In- S: 200 10000000
[424.183ms]--In- X: RecvGET (10000000/10000000/1663)    9.537MiB 23.575MB/s
[449.334ms]--In- S: 200 GET done
```

開発：こちらは最速で上り40MB/s、下り24MB/s となっています。スピードは不安定ですが、20MB/s は確実に出ますね。

基盤：macOSにインストールされているscpは性能が安定しています。例の問題で、上り1MB/sしか出ません。

```
iMac% ssh -V
OpenSSH_8.1p1, LibreSSL 2.7.3
iMac% scp -i ~/.ssh/jp1.pem 10MB jp1:
10MB                                          100% 9766KB 976.2KB/s   00:10
```

開発：この状態が放置されているというのは、信じがたいことです。Appleはこの問題を把握してないんでしょうか？あるいは我々が非常に特殊なネットワーク環境に居るのでしょうか？

社長：そもそも遠隔で scp とか使っている人少ないんでしょうかね…

GShell-0.1.6-by-SatoxITS-1 ダウンロード

＊　＊　＊

開発：ああそれで、ログイン機能というか認証機能についてですが。

社長：やはり公開鍵認証ですかね。

開発：それはパッケージを使えば簡単に出来ると思うのですが、そもそも鍵での認証にも疑問というか不安はあるわけです。秘密鍵を盗まれちゃったらどうしようとか。鍵の管理も面倒。

開発：なので、ワンタイムパスワード的なものをやりたい。たとえばすごく素朴には、gshでサーバを起動する時に、その一時的なサーバにだけ有効なパスワードを指定する。あるいは頻繁に、gsh固有のパスワードを自動生成して変更して、それをいつもgshサーバとgshクライアントとの間で共有しておく。

社長：仮想的な永続セッションですね。

基盤：生成されたパスワードを共有できない環境にあるクライントだと不便なような。

開発：まあ一番最初は、認証中のgshクライアントに発行してもらって、手入力というかコピペさせてもらう、でいいんじゃないですかね。

開発：そもそも秘密鍵の問題は、それを安全に共有というか配るのが大変だということでは無いかと思うんです。だから頻繁には変えられなくて、それがまた危険を生じる。しかも人間が手入力する前提だから簡単なものになって、それもまた危険を生じる。ですが、こういう特定ユーザの特定用途のリモートシェルでは事情というか前提が違う。そもそもオンラインで常につながっているわけですから、配布というか共有はすごく簡単。配布の時の暗号化は Diffie-Hellman で十分だと思います。

社長：たとえばサーバもクライアントも膨大な過去のヒストリを持ってるから、そう例えば１GB程度のヒストリですが、いついつに実行したコマンドと結果はなんだっけ？あなたなら覚えてますよね？みたいな認証方法があっても良さそうですね。

開発：何ギガバイトあっても、ナマのヒストリデータに依拠するのはそれも盗まれた時の危険はあるでしょうから、ひねる必要はあると思います。

基盤：gshサーバからクライアント にメールでワンタイムパスワードを送って、それをgshクライアントがPOPかIMAPで受け取って、サーバに送り返すのでも良いのでは。あるいは携帯に4桁くらいの番号をSMSしてそれを入れる。

社長：うーん、それが今風でカッコいいかな…

開発：いや、でももっとシンプルに、物量作戦で行くとするとですよ。鍵は100GBくらいにする。

社長：2048ビット256バイトの鍵が40万個詰まってるみたいな感じですかね。

開発：使い方は自在かと。何バイト目から何バイト分を鍵として使うとかその場でネゴる。

開発：いまどきのHDDにとって100GBはへのようなもので、鍵保管にかかる費用は約200円。これにHDDととネットの最高速の100MB/sでアクセスできたとしても、盗み出すのに1000秒かかるわけです。というか、すごく遅いSDメモリとかに保管すれば読み出しに1日掛かりです。

社長：相手がクラウド上のサーバだったり携帯だったりすると、現状では巨大鍵の共有はチープではないでしょうね。まあでも、現状で10GBならありですかね…

基盤：思い切り遅い大容量のランダムアクセス可能な記憶があると良いですね。

開発：ランダムアクセスの応答はそこそこ速いけど、連続読み出ししようとするとすごく遅いディスクとかファイルとか。

社長：ソフト的に、連続読み出しをチョークする仕掛けをOSというかファイルシステムに入れてあれば良いようにも思いますね。

開発：ただ、OSにしてもソフトですし、ソフトは騙せますからねえ…　ただのデータだけに。

基盤：そもそもFlashなら、すごい大容量で読み出しがすごく遅いのがありそうです。256バイトの読み出しはミリ秒でできるけど、100GB全部読むのに一週間がかりとか。

社長：それって、鍵を保存するのに1週間かかるということですか？

開発：本来は高速なFlashで、利用時のストレージとの物理的な接続をUSB-1でやるとか、SPIでやるとか… AppleTalk とかｗ

社長：10MbpsのEthernetハブとか、まだ売ってるかもですね。それ経由で鍵ファイルをNFSする。

基盤：人間でなくてユーザが使ってるマシンで認証するという手もありますね。

社長：まあ今どきのマシンならあれを積んでますよね。

開発：何にしても今の認証のインフラって、高速大容量のネットワーク、インターネット、プロセッサのセキュリティ機能が使えるって前提が無しに作られてる気はします。アプリケーションプロトコルにしたってアプリケーションにしたって、前提とか環境がまるで違う時代に作られたものが金科玉条になっちゃってるというか。昔は過去の遺産を活用することが実装のコスト減にも発展の速度にも寄与しましたけど、今は逆になっているんじゃないかって。

社長：まあ、ITの応用を一度まる洗いしたく候というのが当社の設立趣意ですから（笑）

社長：今の所うちの用途では急ぎでは無いですが、認証はぜひやりたいですね。なにか面白いのを。

＊　＊　＊

開発：あー疲れた。昨日の版の整理をしていたら、もう4時をまわってしましまいした。午睡もなく。

社長：私なんて昨日今日と飲みにもいかず液体カロリーメイトですよ。ごくごく。

開発：そうこうするうちに今日はもうそろそろスタートです。

開発：しかしこの草っ原って、ロイヤルという名前が似合わないなぁ…

＊　＊　＊

社長：それで tail -f ですが。

開発：私は、Unixのリダイレクションは良いのですが、shellレベルで入出力がファイルなら > < でプロセスだと | というのがいかがなものかと思うんです。これはプロセスだ、という特殊ファイル名的なもの、例えば < cmdfile{…} なんてすると、そのコマンドからの出力を pipe で受けるでもいいんじゃないかと。

社長：どういうメリットがあるんですかね。

開発：この cmdpath{…} の部分はOSが、というか open システムコールのユーザ空間のラッパーが解釈すると良いと思うんです。そうすると、shellによらず、動的なコンテンツを提供する仮想的なファイルが実現できる。プロセススペシャルファイル的な。全てのアプリからそれが利用できるわけです。

基盤：Windowsのショートカットとかってそういう感じのものじゃなかったでしたっけ。

開発：で、URLというのはまさにそういうものなわけですが、ファイルシステムでは無い。なので、NFSでこれをやりたいと思っているわけです。ただ、NFSだとファイルに対する機能要求が複雑すぎる。それで、FTPでやればどうかという気もするわけです。

社長：この話はごく最近やったような記憶もありますが…　ああ、アプリは拡張子でファイルの型を判定することが多いから、最後は拡張子で終わるのが良い、って話でしたね。

開発：名前は filenmame{…}.ext でも {…} filename.ext でも良いですが。

＊　＊　＊

社長：だいぶ厳しい結果に終わったようですね。

開発：どーんと落ち込んでがーんとバウンスバックすればいいんじゃないですかね。

基盤：明日は開発に集中できますねｗ

社長：で、{tail -f logfile} の件はどうなったでしょうか？

開発：できました。実際、ただpopen相当の事をsyscallでやるだけなので、そこは簡単だったのですが、{ … } をパースするところでツボってしまいました。なんと、Go言語のスライスの基本中の基本を誤解してたのが原因した。Goのスライスの部分列を取り出すには配列[start:end] なのですが、何故か配列[start:length]だと思っていたのです。

基盤：よくそれでこれまで動いてましたねｗ

開発：なんだか不思議な怪奇現象に何度かあったことはあったような記憶がありますが、適当に回避してたんでしょうね。たぶん、配列[:length] と混同してたんだと思います。この場合、end == length ですしね。

社長：その end というのは実際の最終要素ではなくて、+1 だということですね。end という表記は紛らわしいと思います。

開発：あと、shell におけるカッコの解釈についてちょっと検索したんですが、ちょっとおもしろい記事がありました。


開発：shell が使用する特殊記号の中で、カーリーブラケットは一番既定がゆるいというか、解釈が文脈依存なんだと思います。出現する文脈によってはなにも解釈されないので、ユーザプログラムで使用することもできます。

社長：そう、$N と ${N:… } は学生の頃に作ったプログラムでも使ってました。その流れで、DeleGate はデータをくくるのに { } を使ってるんですよね。

基盤：ファイル名を *.{gif,jpg} なんていう感じでマッチするのに良く使いますね。

開発：そう、一般には検索、マッチングなんですが、データの生成機能があるようなんです。

```
dg9% echo a { b } c
a { b } c
dg9% echo {0..10}
0 1 2 3 4 5 6 7 8 9 10
dg9% echo {0..3}{0..3}
00 01 02 03 10 11 12 13 20 21 22 23 30 31 32 33
```

社長：これは面白い。

開発：配列も、配列の配列、…といったこともできるし、当然配列要素の取り出しもできる。

基盤：知りませんでした。文字列を1文字ごとの配列にバラして加工したりマッチングしたり結合したりもできるんでしょうか？

開発：できるんじゃないですかね。それはそうと、zsh が { } になにやら厳しい解釈をしているようで、これで zsh のイメージがガクンと落ちてしまいました。

```
iMac% /bin/sh
sh-3.2$ echo a { b } c
a { b } c
sh-3.2$ exit
exit
iMac% echo a { b } c
zsh: parse error near `}'
```

社長：なんにしても、他のshellとの互換な構文を採用しなければならないとなったら、GShell 固有で色々できるのは { } しかないんでしょうね。

＊　＊　＊

開発：ということで今日のまとめです。

開発：今日の実装では、GShellのクライアントとサーバの間の、一つのtcpコネクション上にコマンド・レスポンスの制御と、ファイルとかの送受信データが一続きで流れるようになっています。ファイルのような静的なデータについては、最初に送信データの長さを知らせて、あとは生でドンと送っています。ギガだろうがテラだろうが。

社長：大小によらず一つのメッセージというかパケットなわけですね。

開発：最高速のデータ転送を実現する上で、CPUでの処理は最小限にしたい。ですのでこれは一つの理想形ではあるわけです。

基盤：10GB/sを実現したいならですけどね。現状では1ギガビット/sしか無いから、100MB/s で処理できれば十分だと思いますが…

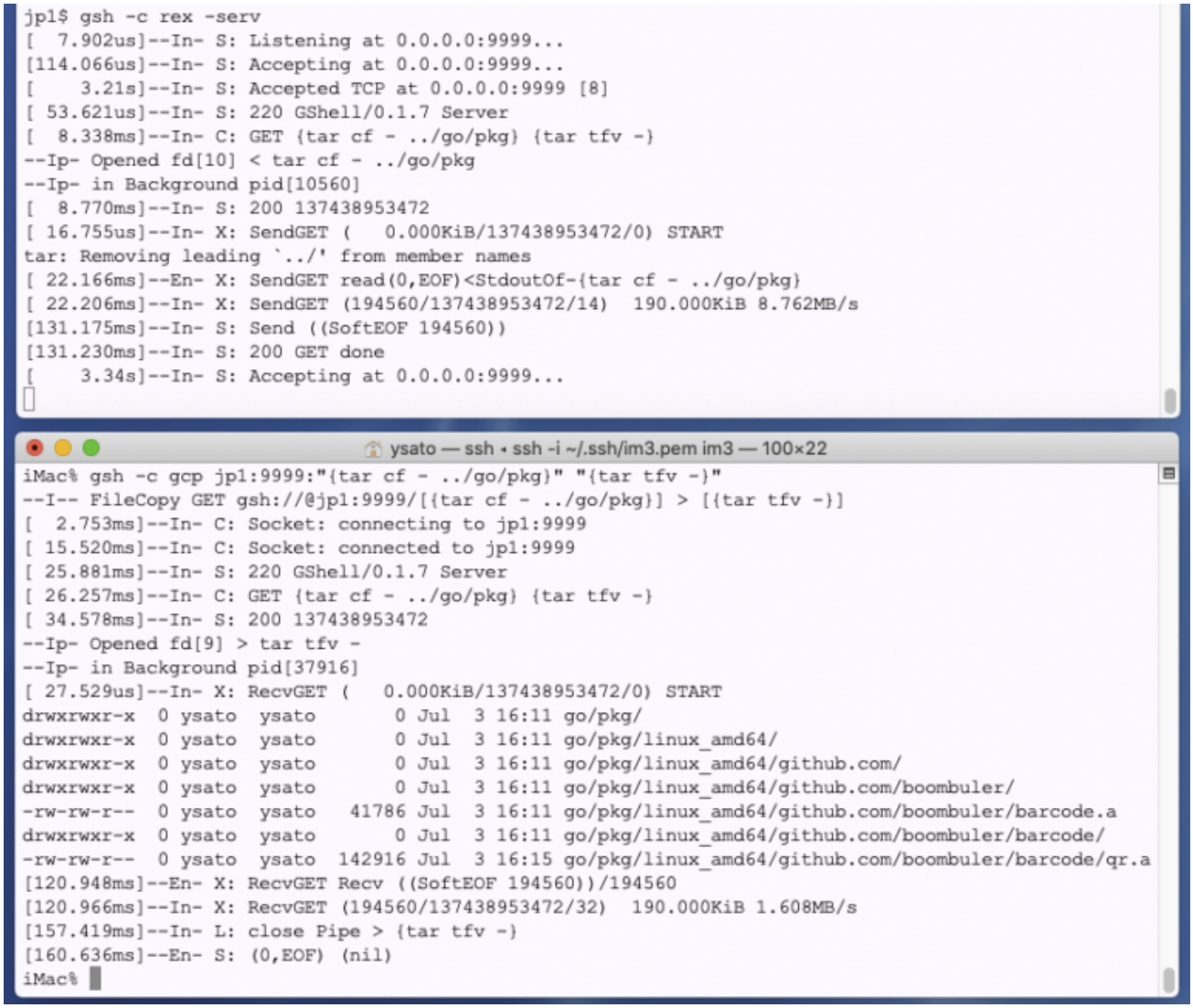
開発：それで当然問題になるのが、プログラムとかでデータを生成して送る場合、送り始める時に長さが不定であるという点です。なので、どうやって受信側がデータの終了を判定して制御のやりとりに戻るか課題。

基盤 ：普通はパケット化しますよね。といっても、単に長さ情報のヘッダだけの可変長パケットで良いと思いますが。

開発：いや、パケット化しないモードがあっていいじゃないですか。というか、プログラムでやるにしても、下の層に分けてやるのがきれいだと思うのです。

社長：下の層でSSLを使う場合には、 SSLのパケット型式に便乗するとよいかも知れませんね。Notifyみたいのを使うとか。

開発：さてそれで実例です。リモートで tar cf したのをローカルで tar tf しています。もちろん tar xf してもOKですが。




開発：ディスク上に実在するファイルのtar の場合、無限長というわけではありませんが、何十何百ギガバイトにもなるかも知れない。あるいは作成に何分も何十分もかかるかも知れない。だからこそ作っては投げのパイプライン並列・ストリーム式が必要なわけです。

社長：tar 型式そのものをパケットというか、型式を理解して中継すれば、終端も分かりそうですね。tar 型式で終端ファイルを送るというか。

開発：ああそれで、上の例の中に出てきますように、データの送信が終わったら、ちょっと間を置いて、この例では100msですが、データ終端データを送っています。ユーザ側がそれまでのデータを受け取り追えていれば、ソケットから読み込んだデータの先頭にこの終端マークがあるはずだ、というわけです。

基盤：100msでも足りないかも知れないですね。膨大な数のデータを送る場合に、1つごとに1/10秒待たされるの実用的で無いと思います。

開発：ともかく、アプリケーション自身が持っているデータのフォーマット形式を利用して長さを規定したい、ただ転送のためにぶつぶつパケットだかチャンクだかに切るのに抵抗があるのです。途中にフィルターをかませる時にすごく害になります。

社長：受信側が受信状況を送信側に知らせてくると良いかもですね。送信側からのデータが途切れたら、今何バイト受け取りましたけど、まだ残りがありますか？無いならこういうデータを送って下さいみたいな。そしたら100ms待つ必要も無いでしょう。

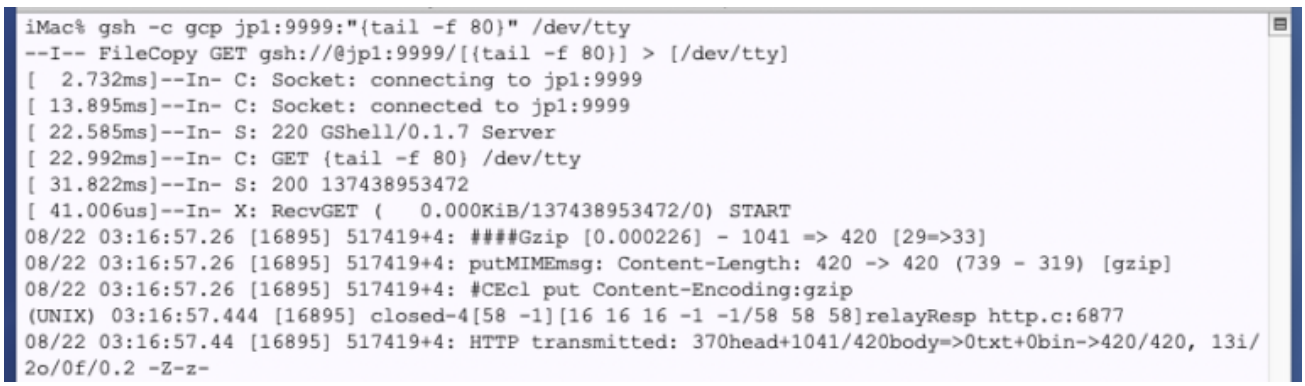
基盤：フランクフルトとだと、そのネゴに100ms以上かかりそうですね。

開発：TCP上のOOBを使うというのもやはり候補ではあると思います。

開発：ただ何にしろ、本命の実装では、データ転送用のチャネルは制御用とは分離する。複数並列転送も可能にする。そういう方針で行きたいと思います。

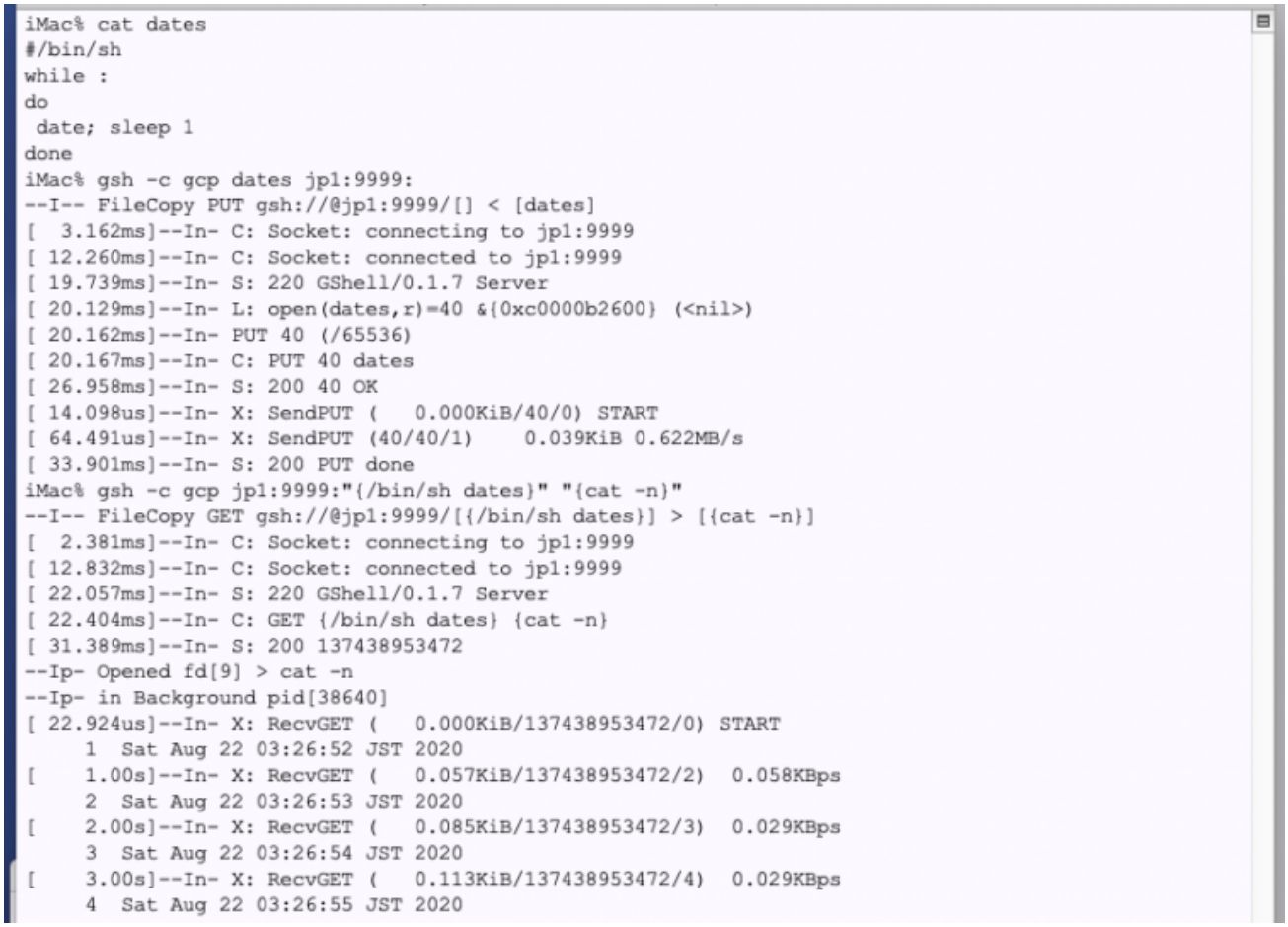
社長：ラジャー。

開発：ああそれで、tail -f で遠隔のログファイルズルズルの例も。




基盤：これは…　普通に便利ですね。リモートログインする機会が激べりしそうです。

開発：あるいはshellスクリプトを送って、実行する。



```
iMac% cat dates
#/bin/sh
while :
do
 date; sleep 1
done
iMac% gsh -c gcp dates jp1:9999:
--I-- FileCopy PUT gsh://@jp1:9999/[] < [dates]
[  3.162ms]--In- C: Socket: connecting to jp1:9999
[ 12.260ms]--In- C: Socket: connected to jp1:9999
[ 19.739ms]--In- S: 220 GShell/0.1.7 Server
[ 20.129ms]--In- L: open(dates,r)=40 &{0xc0000b2600} (<nil>)
[ 20.162ms]--In- PUT 40 (/65536)
[ 20.167ms]--In- C: PUT 40 dates
[ 26.958ms]--In- S: 200 40 OK
[ 14.098us]--In- X: SendPUT (   0.000KiB/40/0) START
[ 64.491us]--In- X: SendPUT (40/40/1)    0.039KiB 0.622MB/s
[ 33.901ms]--In- S: 200 PUT done
iMac% gsh -c gcp jp1:9999:"{/bin/sh dates}" "{cat -n}"
--I-- FileCopy GET gsh://@jp1:9999/[{/bin/sh dates}] > [{cat -n}]
[  2.381ms]--In- C: Socket: connecting to jp1:9999
[ 12.832ms]--In- C: Socket: connected to jp1:9999
[ 22.057ms]--In- S: 220 GShell/0.1.7 Server
[ 22.404ms]--In- C: GET {/bin/sh dates} {cat -n}
[ 31.389ms]--In- S: 200 137438953472
--Ip- Opened fd[9] > cat -n
--Ip- in Background pid[38640]
[ 22.924us]--In- X: RecvGET (   0.000KiB/137438953472/0) START
     1  Sat Aug 22 03:26:52 JST 2020
[    1.00s]--In- X: RecvGET (   0.057KiB/137438953472/2)  0.058KBps
     2  Sat Aug 22 03:26:53 JST 2020
[    2.00s]--In- X: RecvGET (   0.085KiB/137438953472/3)  0.029KBps
     3  Sat Aug 22 03:26:54 JST 2020
[    3.00s]--In- X: RecvGET (   0.113KiB/137438953472/4)  0.029KBps
     4  Sat Aug 22 03:26:55 JST 2020
```

基盤：便利過ぎる…

開発：あとは当面、頻繁に変更するGShell自身をアップロードしてリコンパイルして自分を新たにexecする機能ですね。これをやっとけば私もリモートログインする機会が激べりします。

社長：まあ不特定多数が安全に使うにはこのままではイケナイですけどね。でもそれは必要になったら検討しましょう。

社長：ところで、遠隔ユーザには普通のファイルのように見えて、実はアクティブなファイルだっていうのはどうやるんですかね。

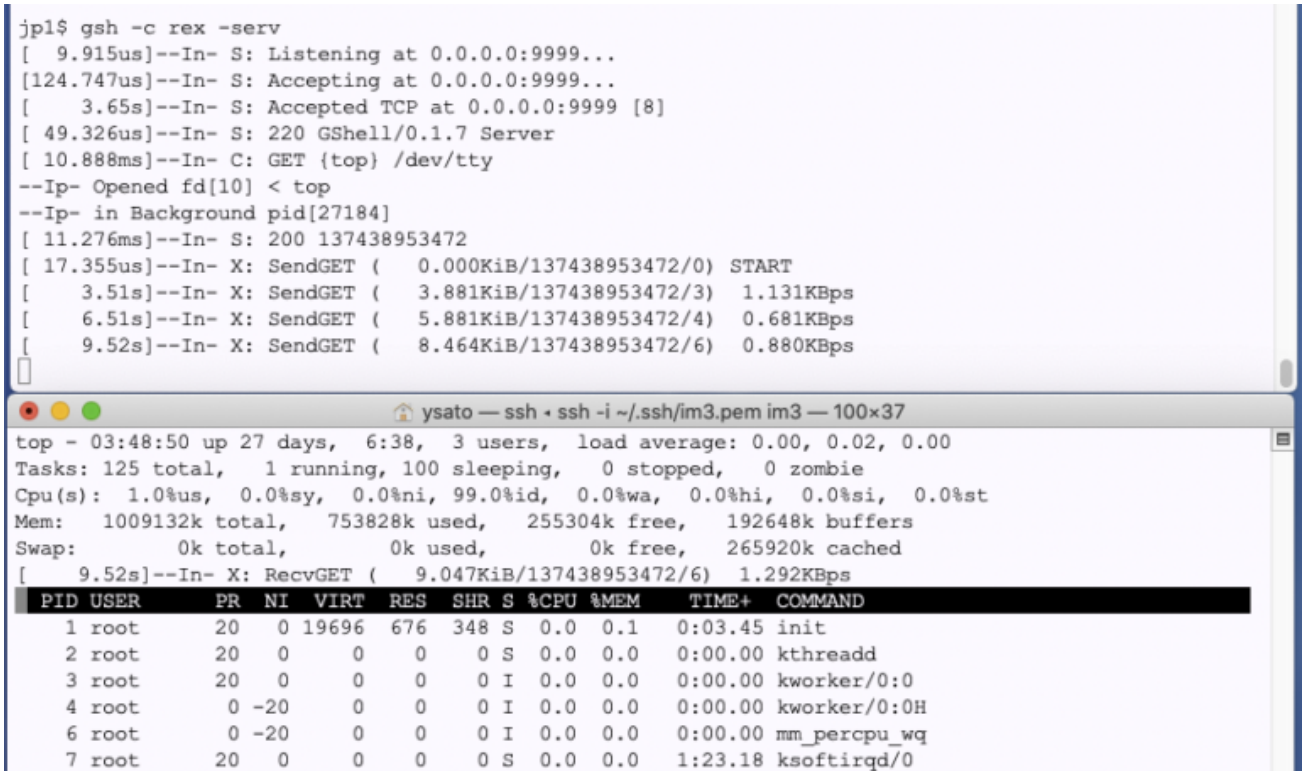
開発：特殊な型式、ファイル名の拡張子か、内容の先頭のマジックで識別される特定の型式のプログラムならGShellスクリプトとして実行する、引数はシンボリックリンクか環境変数で与える、といった形では無いかと。

基盤：CGI互換の環境変数でも良いですよね。

開発：それは当然含めるべきだと思います。ただ、FTPクライアントからは、アプリケーションレベルのオプション情報を与えるのが難しいというか、ごく限定されるとは思いますが。

社長：macOSでは OS X デビューの時からFTPがmountできます。ぜひGShellをFTPサーバにもしたいですね。

開発：おっとー。いまちょっとプチ感動したのですが、ファイル転送でtopも普通に見えますね。




基盤：sshでは怒られちゃいますね。

```
iMac% ssh -i ~/.ssh/jp1.pem jp1 "export TERM=vt100;top"
        top: failed tty get
```

開発：まあこのGShellサーバのプロセスは、端末から起動してますからね。そのtty情報が使われているのでしょう。

基盤：sshでコマンドだけ実行する時に入出力をttyだかpts経由にすることって出来ないんでうかね…

開発：ログインはしないけどtty入出力を想定したコマンドを使える、っていうモードは合って良い感じはしますね。うーん。まー、いずれかのユーザの名義にはなるんでしょうけど。guestとかanonymousでいいかなという。

社長：anonymous で GShell ログインして試用できる GShellのデモサイトがあると良いかもですね。

— 2020-0821 SatoxITS

GShell-0.1.7-by-SatoxITS ダウンロード

http-im3-gsh-gsh-0.1.7.go ダウンロード