株式会社 ITS MORE
2020年4月設立

Digitally signed by 佐藤 豊
Date: 2020.08.10 19:26:33 +09'00'

2020年8月10日 投稿者: SATOXITS

続々GShell

開発：さて、1000行あればshellぽいものになるかなと思っていたのですが、超えてしまいました。まだまだ先は遠いかなという感じです。

Gshell-by-SatoxITS ダウンロード

開発：私自身はshellをとてもシンプルにしか使わないのですが、一方でヒストリー機能とリソース使用量の記録は非常に重要だと思っています。それで、どのコマンドをどこで実行してどのくらいのリソース使用だったかというのを記録することにしました。あと、さっきいたあのディレクトリに戻りたいということが多いので、作業ディレクトリのヒストリを作ることにしました。

開発：まずディレクトリ移動のヒストリはこんな感じ。

```
MacMini% gsh
--I-- GSH_HOME=/Users/ysato/.gsh
!1! -ver
gsh/0.0.5 (2020-0810a)
--I-- (0.000061794s)
!2! cd
--I-- (0.000678226s)
!3! cd /var/log
--I-- (0.000076831s)
!4! cd /usr/lib
--I-- (0.000062419s)
!5! cd /etc
--I-- (0.000083684s)
!6! cd ?
!0 Aug 10 17:37:38 /Users/ysato/gsh
!1 Aug 10 17:38:46 /Users/ysato
!2 Aug 10 17:38:56 /private/var/log
!3 Aug 10 17:39:04 /usr/lib
!4 Aug 10 17:39:06 /private/etc
--I-- (0.002100751s)
!7! cd !2
--I-- (0.000549815s)
!8! pwd
/private/var/log
--I-- (0.004685754s)
```

開発：コマンドのヒストリはこんな感じです。

```
MacMini% gsh
--I-- GSH_HOME=/Users/ysato/.gsh
!1! date
Mon Aug 10 17:49:03 JST 2020
--I-- (0.002291318s)
!2! time -o 1GB openssl rand 1000000000
--I-- Opened [3] 1GB
-o: Usr=0.645621s Sys=1.181960s Rss=892928B Snd=0 Rcv=0 Sig=0
--I-- (3.491167903s)
!3! ls -l 1GB
-rw-------  1 ysato  staff  1000000000 Aug 10 17:49 1GB
--I-- (0.005636178s)
!4! history
!0 date
!1 time -o 1GB openssl rand 1000000000
!2 ls -l 1GB
--I-- (0.000049283s)
!5! history -l
!0 Aug 10 17:49:03 (2.334873ms) date
!1 Aug 10 17:49:18 (3.491174495s) time -o 1GB openssl rand 1000000000
!2 Aug 10 17:49:43 (5.669018ms) ls -l 1GB
!3 Aug 10 17:49:53 (53.065µs) history
--I-- (0.000539181s)
!6! !1
--I-- Opened [4] 1GB
-o: Usr=0.628608s Sys=1.089395s Rss=872448B Snd=0 Rcv=0 Sig=0
--I-- (1.906877230s)
!7! !2
-rw-------  1 ysato  staff  1000000000 Aug 10 17:51 1GB
--I-- (0.004874121s)
```

社長：「あのコマンドを実行したディレクトリに戻りたい」という事が多いので、cdの後の！番号もコマンドのヒストリのものを使えるのが良いように思いますね。

開発：私もそう思いました。ディレクトリ移動のヒストリは、コマンドのヒストリから抽出すれば良いのかなと。

基盤：どのディレクトリでどのくらいリソースを使う仕事をしたとか、そのディレクトリでのアクティブな滞在時間がわかると良いですね。まあそれも、コマンドのヒストリーのほうから集計すれば良いのかも知れませんが。

社長：あと、ヒストリの何番から何番までを再実行したいとか、一つのコマンド列として登録できるようになってると良いですね。

開発：ただ、長大なコマンドヒストリを探したり編集したりをラインエディタでやるのはどうかと思います。なので、そういう作業はこのgshellをHTTPサーバにして、HTTPクライアントというかブラウザからやればよいのかなと。

開発：あとこれは各コマンドの出力も後から読めるようにしたいです。デフォルトで、コマンド単位でsciriptコマンドが動いているみたいな。で、それを閲覧するのもやはり、ブラウザが適していそうです。

社長：リモートにログインした状態でそれをやるには、リモートのgshellサーバに繋がないといけないですが、ローカルな出向元のshellの口から連絡したいですよね。そういう意味でも、リモートシェル機能は内蔵していないといけない。

開発：そうですね。次の主題はこのgshellを簡単なHTTPサーバにすることだと思っています。

基盤：ヒストリの中からコピペっていうのもやりますよね。コマンド番号が邪魔だと思っていたのですが、fc -ln なんていうコマンド・オプションがある模様です。

基盤：!3-5 で !3;!4;!5 をできちゃうと良いかも。

社長：ところで「再実行」は伝統的に！ですが、？をヒストリとかの検索機能に割り当てると良いようにも思うのですが。

開発：？はファイル名の1文字のマッチングに使われてしまってますからねぇ…　でもまあ、行頭の？だけは別扱いでも良いかもしません。ふつうに？として使いたかったら、./?とかすれば良さどうです。

基盤：つまり？がhistoryコマンドということですね。?3とかすると、ヒストリの3番目のコマンドをラインエディタに呼び出すとか。

社長：ディレクトリがタブでコンプリーションできると良いですね。というか、このへんはかな漢字変換のシンプル版みたいなものですから、実際かな漢字変換と同じキーバインディングで検索すればよいのかなと思いますが。

基盤：いっそIMEも作っちゃいたいですね。ローマ字かな変換までなら、簡単なプログラムで出来ると思います。

開発：あと、gshell 自体が検索したファイルとか、に対しては、find の -ls 的に表示できるようにしたいと思っています。まずはwhichでやってみました。

```
MacMini% gsh
--I-- GSH_HOME=/Users/ysato/.gsh
!1! which ssh
/Users/ysato/bin/ssh
--I-- (0.000064418s)
!2! which ssh -a
/Users/ysato/bin/ssh
/usr/bin/ssh
--I-- (0.001356461s)
!3! which ssh -a -ls
-rwxr-xr-x  2157868 Aug  5 21:28:36 /Users/ysato/bin/ssh
-rwxr-xr-x  2142352 Jul 10 07:27:01 /usr/bin/ssh
--I-- (0.001008450s)
```

基盤：shared library でも同じようにできるでしょうね。

開発：ああ、あと chdir のヒストリも同様です。

```
MacMini% gsh
--I-- GSH_HOME=/Users/ysato/.gsh
!1! cd
--I-- (0.000065687s)
!2! cd /var/log
--I-- (0.000083938s)
!3! cd /etc
--I-- (0.000067484s)
!4! cd /tmp
--I-- (0.000071368s)
!5! cd ? -ls
!0 Aug 10 18:36:25 drwxr-xr-x     1216 Aug 10 17:38:22 /Users/ysato/gsh
!1 Aug 10 18:36:32 drwxr-xr-x     5248 Aug 10 17:37:23 /Users/ysato
!2 Aug 10 18:36:37 drwxr-xr-x     1504 Aug 10 10:09:11 /private/var/log
!3 Aug 10 18:36:41 drwxr-xr-x     2752 Aug  6 09:59:13 /private/etc
!4 Aug 10 18:36:44 dtrwxrwxrwx     1088 Aug 10 17:44:58 /private/tmp
--I-- (0.000237581s)
```

開発：それとか、空白を含む引数、特にファイル名ですが、を使うためにいちいちクオートでやるのがすごく嫌です。なので、¥s を空白と解釈することにしました。

```
MacMini% gsh
--I-- GSH_HOME=/Users/ysato/.gsh
!1! echo Hello\sWorld!
Hello World!
--I-- (0.000026043s)
!2! cd /Volumes/GoogleDrive/My\sDrive
--I-- (0.000155646s)
!3! pwd
/Volumes/GoogleDrive/My Drive
--I-- (0.004222191s)
```

社長：空白を¥sで表現するのはDeleGateでもやってました。なんで標準的に存在してないんですかね？

開発：さあ。何にしてもこれは個人の手元で使う表記ですから、やりたいようにやれば良いかなと思います。

社長：whichでやる指定されたPATHの検索とfindは兄弟みたいなものだし、lsは find -ls の簡略版みたいものだと思います。PATHの中でファイルをワイルドカードで検索する機能はもともと動的ライブラリのために存在しますから、それをユーザがコマンドとして、あるいは引数のワイルドカードとして使えるようにすると良いと思いますね。

開発：PATHをいちいち定義するのは面倒だったりするので、これまでに滞在したディレクトリを動的にPATHとして使えれば良いですね。あるいは、これまでプログラムに渡したファイル名と思しき引数の中から検索させる、というかオートコンプリーションさせる。

社長：コンピュータの黎明期にはヒストリ機能というのはリソースを食う贅沢機能だったわけです。個人が使えるディスク容量が1メガも無い時代。コマンドシェルって、そういう時代の錯誤を引きずってる気がしますね。

基盤：まあ、ヒストリに1ギガ使ってもなんのことはないですよね。ベタ検索したって数秒もかからない。

社長：なのでこの先はずっとそういう形でヒストリを保存して、ああ10年前この時にはこんなコマンドを打ってたなみたいな思い出にひたったりするわけです。

開発：ヒストリの検索をしやすくするためには、ユーザが明示的にラベルと言うかタグを付けたいですね。というか、ヒストリをHTTPで、HTMLで眺めるという全体なら、ログインごとにタイトルとかサブタイトルを付ける必要はあるのかなと思います。ブックマークといいいますか。

社長：スクリプトにコメントを書くことは多いですが、インタラクティブに入力している端末でコメントを書くというのは、なんだかシュールで良いですね。

開発：で今気づいたんですが、bash では # 以降をコメントとして無視してくれますが、zsh はそうなってないんですね。

社長：実装上の何か困難は？

開発：だいぶGoの書き方というか、パッケージのドキュメントの流儀とか、スライスの使い方がわかって来たのでスムーズになりました。ただ今作っているのがシステム寄りのコードなのですが、os パッケージが非力というか未熟なのがつらいですかね。syscall になることが多いです。

▼ gsh0.0.5
//

```
// gsh - Go lang based Shell
// (c) 2020 ITS more Co., Ltd.
// 2020-0807 created by SatoxITS (sato@its-more.jp)
//
package main // gsh main
// Documents: https://golang.org/pkg/
import (
        "bufio"
        "strings"
        "strconv"
        "sort"
        "fmt"
        "os"
        "time"
        "syscall"
        "go/types"
        "go/token"
        "net"
)

var VERSION = "gsh/0.0.5 (2020-0810a)"
var LINESIZE = (8*1024)
var PATHSEP = ":" // should be ";" in Windows
var DIRSEP = "/" // canbe \ in Windows
var PROMPT = "> "
var GSH_HOME = ".gsh" // under home directory

type GCommandHistory struct {
        StartAt         time.Time
```

```
        EndAt           time.Time
        ResCode         int
        OutData         *os.File
        CmdLine         string
        ResCons         int // Resource consumption, CPU time or so
}
type GChdirHistory struct {
        Dir             string
        MovedAt         time.Time
}
type GshContext struct {
        StartDir        string  // the current directory at the start
        GetLine         string  // gsh-getline command as a input line
        ChdirHistory    []GChdirHistory // the 1st entry is wd at the
        gshPA           syscall.ProcAttr
        CommandHistory  []GCommandHistory
        BackGround      bool
        BackGroundJobs  []int
        LastRusage      syscall.Rusage
        GshHomeDir      string
        TerminalId      int
}

func isin(what string, list []string) bool {
        for _, v := range list  {
                if v == what {
                        return true
                }
        }
        return false
}

func env(opts []string) {
        env := os.Environ()
        if isin("-s", opts){
                sort.Slice(env, func(i,j int) bool {
                        return env[i] < env[j]
                })
        }
        for _, v := range env {
                fmt.Printf("%v\n",v)
        }
}
```

```
func strsubst(str string) string {
        rstr := ""
        inEsc := 0 // escape characer mode
        for _, ch := range str {
                if inEsc == 0 {
                        if ch == '\\' {
                                inEsc = '\\'
                                continue;
                        }
                }
                if inEsc == '\\' {
                        if ch == 's' { ch = ' ' }
                        if ch == 'r' { ch = '\r' }
                        if ch == 'n' { ch = '\n' }
                        if ch == 't' { ch = '\t' }
                        if ch == '\\' { ch = '\\' }
                        inEsc = 0
                }
                rstr = rstr + string(ch)
        }
        return rstr
}

func showFileInfo(path string, opts []string) {
        if isin("-ls",opts) {
                fi, _ := os.Stat(path)
                mod := fi.ModTime()
                date := mod.Format(time.Stamp)
                fmt.Printf("%v %8v %s ",fi.Mode(),fi.Size(),date)
        }
        fmt.Printf("%s",path)
        if ! isin("-n",opts) {
                fmt.Printf("\n")
        }
}

func toFullpath(path string) (fullpath string) {
        pathv := strings.Split(path,DIRSEP)
        if pathv[0] == "." {
                pathv[0], _ = os.Getwd()
        }else
        if pathv[0] == ".." {
```

```
                cwd, _ := os.Getwd()
                ppathv := strings.Split(cwd,DIRSEP)
                pathv[0] = strings.Join(ppathv,DIRSEP)
        }else
        if pathv[0] == "~" {
                pathv[0],_ = os.UserHomeDir()
        }
        return strings.Join(pathv,DIRSEP)
}

func which(list string, argv []string) (fullpathv []string, itis bool)
        if len(argv) <= 1 {
                fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
                return []string{""}, false
        }
        path := argv[1]
        pathenv, efound := os.LookupEnv(list)
        if ! efound {
                fmt.Printf("which: no \"%s\" environment\n",list)
                return []string{""}, false
        }
        dirv := strings.Split(pathenv,PATHSEP)
        ffound := false
        ffullpath := path
        for _, dir := range dirv {
                fullpath := dir + DIRSEP + path
                fi, err := os.Stat(fullpath)
                if err != nil {
                        fullpath = dir + DIRSEP + path + ".go"
                        fi, err = os.Stat(fullpath)
                }
                if err == nil {
                        fm := fi.Mode()
                        if fm.IsRegular() {
                                ffullpath = fullpath
                                ffound = true
                                if ! isin("-s", argv) {
                                        showFileInfo(fullpath,argv)
                                }
                                if ! isin("-a", argv) {
                                        break;
                                }
                        }
```

```
                }
        }
        return []string{ffullpath}, ffound
}

func find(argv []string){
}

func eval(argv []string, nlend bool){
        var ai = 1
        pfmt := "%s"
        if argv[ai][0:1] == "%" {
                pfmt = argv[ai]
                ai = 2
        }
        if len(argv) <= ai {
                return
        }
        gocode := strings.Join(argv[ai:]," ");
        fset := token.NewFileSet()
        rval, _ := types.Eval(fset,nil,token.NoPos,gocode)
        fmt.Printf(pfmt,rval.Value)
        if nlend { fmt.Printf("\n") }
}

func getval(name string) (found bool, val int) {
        /* should expand the name here */
        if name == "gsh.pid" {
                return true, os.Getpid()
        }else
        if name == "gsh.ppid" {
                return true, os.Getppid()
        }
        return false, 0
}

func echo(argv []string, nlend bool){
        for ai := 1; ai < len(argv); ai++ {
                if 1 < ai {
                        fmt.Printf(" ");
                }
                arg := argv[ai]
                found, val := getval(arg)
```

```
                if found {
                        fmt.Printf("%d",val)
                }else{
                        fmt.Printf("%s",arg)
                }
        }
        if nlend {
                fmt.Printf("\n");
        }
}

func resfile() string {
        return "gsh.tmp"
}
//var resF *File
func resmap() {
        //_ , err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os
        // https://developpaper.com/solution-to-golang-bad-file-descri
        _ , err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
        if err != nil {
                fmt.Printf("refF could not open: %s\n",err)
        }else{
                fmt.Printf("refF opened\n")
        }
}

func excommand(gshCtx GshContext, exec bool, argv []string) (GshContex
        gshPA := gshCtx.gshPA
        fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"
        if itis == false {
                return gshCtx, true
        }
        fullpath := fullpathv[0]
        if 0 < strings.Index(fullpath,".go") {
                nargv := argv // []string{}
                gofullpathv, itis := which("PATH",[]string{"which","go
                if itis == false {
                        fmt.Printf("--F-- Go not found\n")
                        return gshCtx, true
                }
                gofullpath := gofullpathv[0]
                nargv = []string{ gofullpath, "run", fullpath }
                fmt.Printf("--I-- %s {%s %s %s}\n",gofullpath,
```

```
                        nargv[0],nargv[1],nargv[2])
                if exec {
                        syscall.Exec(gofullpath,nargv,os.Environ())
                }else{
                        pid, _ := syscall.ForkExec(gofullpath,nargv,&
                        if gshCtx.BackGround {
                                fmt.Printf("--I-- in Background [%d]\n
                                gshCtx.BackGroundJobs = append(gshCtx
                        }else{
                                rusage := syscall.Rusage {}
                                syscall.Wait4(pid,nil,0,&rusage)
                                gshCtx.LastRusage = rusage
                        }
                }
        }else{
                if exec {
                        syscall.Exec(fullpath,argv,os.Environ())
                }else{
                        pid, _ := syscall.ForkExec(fullpath,argv,&gshI
                        //fmt.Printf("[%d]\n",pid); // '&' to be back
                        if gshCtx.BackGround {
                                fmt.Printf("--I-- in Background [%d]\n
                                gshCtx.BackGroundJobs = append(gshCtx
                        }else{
                                rusage := syscall.Rusage {}
                                syscall.Wait4(pid,nil,0,&rusage);
                                gshCtx.LastRusage = rusage
                        }
                }
        }
        return gshCtx, false
}
func sleep(gshCtx GshContext, gshPA syscall.ProcAttr, argv []string)
        if len(argv) < 2 {
                fmt.Printf("Sleep 100ms, 100us, 100ns, ...\n")
                return
        }
        duration := argv[1];
        d, err := time.ParseDuration(duration)
        if err != nil {
                d, err = time.ParseDuration(duration+"s")
                if err != nil {
                        fmt.Printf("duration ? %s (%s)\n",duration,err
```

```
                        return
                }
        }
        fmt.Printf("Sleep %v ns\n",duration)
        time.Sleep(d)
        if 0 < len(argv[2:]) {
                gshellv(gshCtx, gshPA, argv[2:])
        }
}
func repeat(gshCtx GshContext, gshPA syscall.ProcAttr, argv []string)
        if len(argv) < 2 {
                return
        }
        start0 := time.Now()
        for ri,_ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
                if 0 < len(argv[2:]) {
                        //start := time.Now()
                        gshellv(gshCtx, gshPA, argv[2:])
                        end := time.Now()
                        elps := end.Sub(start0);
                        if( 1000000000 < elps ){
                                fmt.Printf("(repeat#%d %v)\n",ri,elps)
                        }
                }
        }
}

func gen(gshPA syscall.ProcAttr, argv []string) {
        if len(argv) < 2 {
                fmt.Printf("Usage: %s N\n",argv[0])
                return
        }
        // should br repeated by "repeat" command
        count, _ := strconv.Atoi(argv[1])
        fd := gshPA.Files[1] // Stdout
        file := os.NewFile(fd,"internalStdOut")
        fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
        //buf := []byte{}
        outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
        for gi := 0; gi < count; gi++ {
                file.WriteString(outdata)
        }
        //file.WriteString("\n")
```

```
        fmt.Printf("\n(%d B)\n",count*len(outdata));
        //file.Close()
}

// -s, -si, -so // bi-directional, source, sync (maybe socket)
func sconnect(gshCtx GshContext, gshPA syscall.ProcAttr, inTCP bool, a
        if len(argv) < 2 {
                fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
                return
        }
        remote := argv[1]
        if remote == ":" { remote = "0.0.0.0:9999" }

        if inTCP { // TCP
                dport, err := net.ResolveTCPAddr("tcp",remote);
                if err != nil {
                        fmt.Printf("Address error: %s (%s)\n",remote,e
                        return
                }
                conn, err := net.DialTCP("tcp",nil,dport)
                if err != nil {
                        fmt.Printf("Connection error: %s (%s)\n",remot
                        return
                }
                file, _ := conn.File();
                fd := file.Fd()
                fmt.Printf("Socket: connected to %s, socket[%d]\n",rer

                savfd := gshPA.Files[1]
                gshPA.Files[1] = fd;
                gshellv(gshCtx, gshPA, argv[2:])
                gshPA.Files[1] = savfd
                file.Close()
                conn.Close()
        }else{
                //dport, err := net.ResolveUDPAddr("udp4",remote);
                dport, err := net.ResolveUDPAddr("udp",remote);
                if err != nil {
                        fmt.Printf("Address error: %s (%s)\n",remote,e
                        return
                }
                //conn, err := net.DialUDP("udp4",nil,dport)
                conn, err := net.DialUDP("udp",nil,dport)
```

```
                if err != nil {
                        fmt.Printf("Connection error: %s (%s)\n",remot
                        return
                }
                file, _ := conn.File();
                fd := file.Fd()

                ar := conn.RemoteAddr()
                //al := conn.LocalAddr()
                fmt.Printf("Socket: connected to %s [%s], socket[%d]\n
                        remote,ar.String(),fd)

                savfd := gshPA.Files[1]
                gshPA.Files[1] = fd;
                gshellv(gshCtx, gshPA, argv[2:])
                gshPA.Files[1] = savfd
                file.Close()
                conn.Close()
        }
}
func saccept(gshCtx GshContext, gshPA syscall.ProcAttr, inTCP bool, ar
        if len(argv) < 2 {
                fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
                return
        }
        local := argv[1]
        if local == ":" { local = "0.0.0.0:9999" }
        if inTCP { // TCP
                port, err := net.ResolveTCPAddr("tcp",local);
                if err != nil {
                        fmt.Printf("Address error: %s (%s)\n",local,er
                        return
                }
                //fmt.Printf("Listen at %s...\n",local);
                sconn, err := net.ListenTCP("tcp", port)
                if err != nil {
                        fmt.Printf("Listen error: %s (%s)\n",local,err
                        return
                }
                //fmt.Printf("Accepting at %s...\n",local);
                aconn, err := sconn.AcceptTCP()
                if err != nil {
                        fmt.Printf("Accept error: %s (%s)\n",local,err
```

```
                return
        }
        file, _ := aconn.File()
        fd := file.Fd()
        fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)

        savfd := gshPA.Files[0]
        gshPA.Files[0] = fd;
        gshellv(gshCtx, gshPA, argv[2:])
        gshPA.Files[0] = savfd

        sconn.Close();
        aconn.Close();
        file.Close();
}else{
        //port, err := net.ResolveUDPAddr("udp4",local);
        port, err := net.ResolveUDPAddr("udp",local);
        if err != nil {
                fmt.Printf("Address error: %s (%s)\n",local,er
                return
        }
        fmt.Printf("Listen UDP at %s...\n",local);
        //uconn, err := net.ListenUDP("udp4", port)
        uconn, err := net.ListenUDP("udp", port)
        if err != nil {
                fmt.Printf("Listen error: %s (%s)\n",local,err
                return
        }
        file, _ := uconn.File()
        fd := file.Fd()
        ar := uconn.RemoteAddr()
        remote := ""
        if ar != nil { remote = ar.String() }
        if remote == "" { remote = "?" }

        // not yet received
        //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"'

        savfd := gshPA.Files[0]
        gshPA.Files[0] = fd;
        savenv := gshPA.Env
        gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
        gshellv(gshCtx, gshPA, argv[2:])
```

```
                gshPA.Env = savenv
                gshPA.Files[0] = savfd

                uconn.Close();
                file.Close();
        }
}

// empty line command
func pwd(gshPA syscall.ProcAttr){
        // execute context command, pwd + date
        // context notation, representation scheme, to be resumed at r
        cwd, _ := os.Getwd()
        t := time.Now()
        date := t.Format(time.UnixDate)
        exe, _ := os.Executable()
        host, _ := os.Hostname()
        fmt.Printf("{PWD=\"%s\"",cwd)
        fmt.Printf(" HOST=\"%s\"",host)
        fmt.Printf(" DATE=\"%s\"",date)
        fmt.Printf(" TIME=\"%s\"",t.String())
        fmt.Printf(" PID=\"%d\"",os.Getpid())
        fmt.Printf(" EXE=\"%s\"",exe)
        fmt.Printf("}\n")
}

// these should be browsed and edited by HTTP browser
// show the time of command with -t and direcotry with -ls
// openfile-history, sort by -a -m -c
// sort by elapsed time by -t -s
// search by "more" like interface
// edit history
// sort history, and wc or uniq
// CPU and other resource consumptions
// limit showing range (by time or so)
// export / import history
func xHistory(gshCtx GshContext, argv []string) (rgshCtx GshContext)
        for i, v := range gshCtx.CommandHistory {
                fmt.Printf("!%d ",i)
                if isin("-v",argv){
                        fmt.Println(v) // should be with it date
                }else{
                        if isin("-l",argv){
```

```
                                elps := v.EndAt.Sub(v.StartAt);
                                start := v.StartAt.Format(time.Stamp)
                                fmt.Printf("%s (%8v) ",start,elps)
                        }
                        fmt.Printf("%s",v.CmdLine)
                        fmt.Printf("\n")
                }
        }
        return gshCtx
}
// !n - history index
func searchHistory(gshCtx GshContext, gline string) (string, bool, boo
        if gline[0] == '!' {
                hix, err := strconv.Atoi(gline[1:])
                if err != nil {
                        fmt.Printf("--E-- (%s : range)\n",hix)
                        return "", false, true
                }
                if hix < 0 || len(gshCtx.CommandHistory) <= hix {
                        fmt.Printf("--E-- (%d : out of range)\n",hix)
                        return "", false, true
                }
                return gshCtx.CommandHistory[hix].CmdLine, false, fals
        }
        // search
        //for i, v := range gshCtx.CommandHistory {
        //}
        return gline, false, false
}

// temporary adding to PATH environment
// cd name -lib for LD_LIBRARY_PATH
// chdir with directory history (date + full-path)
// -s for sort option (by visit date or so)
func xChdirHistory(gshCtx GshContext, argv []string){
        for i, v := range gshCtx.ChdirHistory {
                fmt.Printf("!%d ",i)
                fmt.Printf("%v ",v.MovedAt.Format(time.Stamp))
                showFileInfo(v.Dir,argv)
        }
}
func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
        cdhist := gshCtx.ChdirHistory
```

```
        if isin("?",argv ) || isin("-t",argv) {
                xChdirHistory(gshCtx,argv)
                return gshCtx
        }
        pwd, _ := os.Getwd()
        dir := ""
        if len(argv) <= 1 {
                dir = toFullpath("~")
        }else{
                dir = argv[1]
        }
        if dir[0] == '!' {
                if dir == "!0" {
                        dir = gshCtx.StartDir
                }else
                if dir == "!!" {
                        index := len(cdhist) - 1
                        if 0 < index { index -= 1 }
                        dir = cdhist[index].Dir
                }else{
                        index, err := strconv.Atoi(dir[1:])
                        if err != nil {
                                fmt.Printf("--E-- xChdir(%v)\n",err)
                                dir = "?"
                        }else
                        if len(gshCtx.ChdirHistory) <= index {
                                fmt.Printf("--E-- xChdir(history range
                                dir = "?"
                        }else{
                                dir = cdhist[index].Dir
                        }
                }
        }
        if dir != "?" {
                err := os.Chdir(dir)
                if err != nil {
                        fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],er
                }else{
                        cwd, _ := os.Getwd()
                        if cwd != pwd {
                                hist1 := GChdirHistory { }
                                hist1.Dir = cwd
                                hist1.MovedAt = time.Now()
```

```
                                gshCtx.ChdirHistory = append(cdhist,h
                        }
                }
        }
        return gshCtx
}
func showRusage(what string,argv []string, ru *syscall.Rusage){
        fmt.Printf("%s: ",what);
        fmt.Printf("Usr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
        fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
        fmt.Printf(" Rss=%vB",ru.Maxrss)
        if isin("-l",argv) {
                fmt.Printf(" MinFlt=%v",ru.Minflt)
                fmt.Printf(" MajFlt=%v",ru.Majflt)
                fmt.Printf(" IxRSS=%vB",ru.Ixrss)
                fmt.Printf(" IdRSS=%vB",ru.Idrss)
                fmt.Printf(" Nswap=%vB",ru.Nswap)
        fmt.Printf(" Read=%v",ru.Inblock)
        fmt.Printf(" Write=%v",ru.Oublock)
        }
        fmt.Printf(" Snd=%v",ru.Msgsnd)
        fmt.Printf(" Rcv=%v",ru.Msgrcv)
        //if isin("-l",argv) {
                fmt.Printf(" Sig=%v",ru.Nsignals)
        //}
        fmt.Printf("\n");
}
func xTime(gshCtx GshContext, argv []string) (GshContext,bool) {
        if 2 <= len(argv){
                xgshCtx, fin := gshellv(gshCtx,gshCtx.gshPA,argv[1:])
                gshCtx = xgshCtx
                showRusage(argv[1],argv,&gshCtx.LastRusage)
                return gshCtx, fin
        }else{
                rusage:= syscall.Rusage {}
                syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
                showRusage("self",argv, &rusage)
                syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
                showRusage("child",argv, &rusage)
                return gshCtx, false
        }
}
func xJobs(gshCtx GshContext, argv []string){
```

```
        fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
        for ji, pid := range gshCtx.BackGroundJobs {
                //wstat := syscall.WaitStatus {0}
                rusage := syscall.Rusage {}
                //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHAN
                wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&ru
                if err != nil {
                        fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err
                }else{
                                fmt.Printf("%%%d[%d](%d)\n",ji,pid,wpi
                        showRusage("chld",argv,&rusage)
                }
        }
}

func gshellv(gshCtx GshContext, gshPA syscall.ProcAttr, argv []string)
        //fmt.Printf("--I-- gshellv((%d))\n",len(argv))
        if len(argv) <= 0 {
                return gshCtx, false
        }
        for ai := 0; ai < len(argv); ai++ {
                argv[ai] = strsubst(argv[ai])
        }
        if false {
                for ai := 0; ai < len(argv); ai++ {
                        fmt.Printf("[%d] %s [%d]%T\n",
                                ai,argv[ai],len(argv[ai]),argv[ai])
                }
        }
        cmd := argv[0]
        if cmd == "-ot" {
                sconnect(gshCtx, gshPA, true, argv)
                return gshCtx, false
        }
        if cmd == "-ou" {
                sconnect(gshCtx, gshPA, false, argv)
                return gshCtx, false
        }
        if cmd == "-it" {
                saccept(gshCtx, gshPA, true , argv)
                return gshCtx, false
        }
        if cmd == "-iu" {
```

```
                saccept(gshCtx, gshPA, false, argv)
                return gshCtx, false
        }
        if cmd == "-i" || cmd == "-o" || cmd == "-a" || cmd == "-s" {
                if len(argv) < 2 {
                        return gshCtx, false
                }
                fdix := 0;
                mode := os.O_RDONLY;
                if cmd == "-i" {
                }
                if cmd == "-o" {
                        fdix = 1;
                        mode = os.O_RDWR | os.O_CREATE;
                }
                if cmd == "-a" {
                        fdix = 1;
                        mode = os.O_RDWR | os.O_CREATE | os.O_APPEND;
                }
                f, err := os.OpenFile(argv[1], mode, 0600)
                if err != nil {
                        fmt.Printf("%s\n",err)
                        return gshCtx, false
                }
                savfd := gshPA.Files[fdix]
                gshPA.Files[fdix] = f.Fd()
                fmt.Printf("--I-- Opened [%d] %s\n",f.Fd(),argv[1])
                gshCtx, _ = gshellv(gshCtx, gshPA, argv[2:])
                gshPA.Files[fdix] = savfd
                return gshCtx, false
        }
        if cmd == "-bg" {
                xfin := false
                // set background option
                gshCtx.BackGround = true
                gshCtx, xfin = gshellv(gshCtx,gshPA,argv[1:])
                gshCtx.BackGround = false
                return gshCtx, xfin
        }
        if cmd == "call" {
                gshCtx, _ = excommand(gshCtx, false,argv[1:])
                return gshCtx, false
        }
```

```
        if cmd == "cd" || cmd == "chdir" {
                gshCtx = xChdir(gshCtx,argv);
                return gshCtx, false
        }
        if cmd == "#define" {
        }
        if cmd == "echo" {
                echo(argv,true)
                return gshCtx, false
        }
        if cmd == "env" {
                env(argv)
                return gshCtx, false
        }
        if cmd == "eval" {
                eval(argv,true)
                return gshCtx, false
        }
        if cmd == "exec" {
                gshCtx, _ = excommand(gshCtx, true,argv[1:])
                // should not return here
                return gshCtx, false
        }
        if cmd == "exit" || cmd == "quit" {
                // write Result code EXIT to 3>
                return gshCtx, true
        }
        if cmd == "-find" {
                find(argv)
                return gshCtx, false
        }
        if cmd == "fork" {
                // mainly for a server
                return gshCtx, false
        }
        if cmd == "-gen" {
                gen(gshPA, argv)
                return gshCtx, false
        }
        if cmd == "history" || cmd == "hi" { // hi should be alias
                gshCtx = xHistory(gshCtx, argv)
                return gshCtx, false
        }
```

```
        if cmd == "jobs" {
                xJobs(gshCtx,argv)
                return gshCtx, false
        }
        if cmd == "nop" {
                return gshCtx, false
        }
        if cmd == "pstitle" {
                // to be gsh.title
        }
        if cmd == "repeat" || cmd == "rep" { // repeat cond command
                repeat(gshCtx,gshPA,argv)
                return gshCtx, false
        }
        if cmd == "set" { // set name ...
                return gshCtx, false
        }
        if cmd == "time" {
                gshCtx, fin = xTime(gshCtx,argv)
                return gshCtx, fin
        }
        if cmd == "sleep" {
                sleep(gshCtx,gshPA,argv)
                return gshCtx, false
        }
        if cmd == "-ver" {
                fmt.Printf("%s\n",VERSION);
                return gshCtx, false
        }
        if cmd == "pwh" {
                pwd(gshPA);
                return gshCtx, false
        }
        if cmd == "where" {
                // data file or so?
        }
        if cmd == "which" {
                which("PATH",argv);
                return gshCtx, false
        }
        gshCtx, _ = excommand(gshCtx, false,argv)
        return gshCtx, false
}
```

```
func gshelll(gshCtx GshContext, gshPA syscall.ProcAttr, gline string)
        argv := strings.Split(string(gline)," ")
        gshCtx, fin := gshellv(gshCtx,gshPA,argv)
        return gshCtx, fin
}
func tgshelll(gshCtx GshContext, gshPA syscall.ProcAttr, gline string)
        start := time.Now()
        gshCtx, fin := gshelll(gshCtx,gshPA,gline)
        end := time.Now()
        elps := end.Sub(start);
        fmt.Printf("--I-- (%d.%09ds)\n",elps/1000000000,elps%10000000
        return gshCtx, fin
}
func Ttyid() (int) {
        fi, err := os.Stdin.Stat()
        if err != nil {
                return 0;
        }
        //fmt.Printf("Stdin: %v Dev=%d\n",
        //      fi.Mode(),fi.Mode()&os.ModeDevice)
        if (fi.Mode() & os.ModeDevice) != 0 {
                stat := syscall.Stat_t{};
                err := syscall.Fstat(0,&stat)
                if err != nil {
                        //fmt.Printf("--I-- Stdin: (%v)\n",err)
                }else{
                        //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
                        //      stat.Rdev&0xFF,stat.Rdev);
                        //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&
                        return int(stat.Rdev & 0xFF)
                }
        }
        return 0
}
func ttyfile(gshCtx GshContext) string {
        //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
        ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
                 strconv.Itoa(gshCtx.TerminalId)
        //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
        return ttyfile
}
func ttyline(gshCtx GshContext) (*os.File){
        file, err := os.OpenFile(ttyfile(gshCtx),
```

```
                os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
        if err != nil {
                fmt.Printf("--F-- cannot open %s (%s)\n",ttyfile(gshCt
                return file;
        }
        return file
}
func getline(gshCtx GshContext, hix int, skipping, with_exgetline bool
        if( skipping ){
                reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
                line, _, _ := reader.ReadLine()
                return string(line)
        }else
        if( with_exgetline && gshCtx.GetLine != "" ){
                //var xhix int64 = int64(hix); // cast
                newenv := os.Environ()
                newenv = append(newenv, "GSH_LINENO="+strconv.FormatIn

                tty := ttyline(gshCtx)
                tty.WriteString(prevline)
                Pa := os.ProcAttr {
                        "", // start dir
                        newenv, //os.Environ(),
                        []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
                        nil,
                }
//fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine
proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getli
                if err != nil {
                        fmt.Printf("Proc ERROR (%s)\n",nil)
                        for ; ; {
                        }
                }
                //stat, err := proc.Wait()
                proc.Wait()
                buff := make([]byte,LINESIZE)
                count, err := tty.Read(buff)
                //_, err = tty.Read(buff)
                //fmt.Printf("--D-- getline (%d)\n",count)
                if err != nil {
                        if ! (count == 0) { // && err.String() == "EOF
                                fmt.Printf("--E-- getline error (%s)\n
                        }
```

```
                }else{
                        //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
                }
                tty.Close()
                return string(buff[0:count])
        }else{
                // if isatty {
                        fmt.Printf("!%d",hix)
                        fmt.Print(PROMPT)
                // }
                reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
                line, _, _ := reader.ReadLine()
                return string(line)
        }
}
//
// $USERHOME/.gsh/
//               gsh-history.txt
//               gsh-aliases.txt // should be conditional?
//
func gshSetup(gshCtx GshContext) (GshContext, bool) {
        homedir, err := os.UserHomeDir()
        if err != nil {
                fmt.Printf("--E-- You have no UserHomeDir (%v)\n",err)
                return gshCtx, true
        }
        gshhome := homedir + "/" + GSH_HOME
        _, err2 := os.Stat(gshhome)
        if err2 != nil {
                err3 := os.Mkdir(gshhome,0700)
                if err3 != nil {
                        fmt.Printf("--E-- Could not Create %s (%s)\n",
                                gshhome,err)
                        return gshCtx, true
                }
                fmt.Printf("--I-- Created %s\n",gshhome)
        }
        gshCtx.GshHomeDir = gshhome
        return gshCtx, false
}
func script(gshCtx GshContext) (_ GshContext) {
        gshCtx, err0 := gshSetup(gshCtx)
        if err0 {
```

```
		return gshCtx
	}
	gshPA := syscall.ProcAttr {
		"", // the staring directory
		os.Environ(), // environ[]
		[]uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()
		nil, // OS specific
	}
	cwd, _ := os.Getwd()
	gshCtx = GshContext {
		cwd, // StartDir
		"", // GetLine
		[]GChdirHistory { {cwd,time.Now()} }, // ChdirHistory
		gshPA,
		[]GCommandHistory{ }, //something for invokation?
		false,
		[]int{},
		syscall.Rusage{},
		"", // GshHomeDir
		Ttyid(),
	}
	gshCtx, _ = gshSetup(gshCtx)
	fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
	//resmap()
	gsh_getlinev, with_exgetline :=
		 which("PATH",[]string{"which","gsh-getline","-s"})
	if with_exgetline {
		gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
		gshCtx.GetLine = toFullpath(gsh_getlinev[0])
	}else{
	fmt.Printf("--W-- No gsh-getline found. Using internal getline
	}

	prevline := ""
	skipping := false
	for hix := 1; ; {
		gline := getline(gshCtx,hix,skipping,with_exgetline,gs
		if skipping {
			if strings.Index(gline,"fi") == 0 {
				fmt.Printf("fi\n");
				skipping = false;
			}else{
				//fmt.Printf("%s\n",gline);
```

```
                        }
                        continue
                }
                if strings.Index(gline,"if") == 0 {
                        //fmt.Printf("--D-- if start: %s\n",gline);
                        skipping = true;
                        continue
                }
                if 0 < len(gline) && gline[0] == '!' {
                        xgline, set, err := searchHistory(gshCtx,gline
                        if err {
                                continue
                        }
                        if set {
                                // set the line in command line editor
                        }
                        gline = xgline
                }
                ghist := GCommandHistory { }
                ghist.StartAt = time.Now()
                xgshCtx, fin := tgshelll(gshCtx,gshPA,gline)
                gshCtx = xgshCtx
                ghist.EndAt = time.Now()
                ghist.CmdLine = gline
                gshCtx.CommandHistory = append(gshCtx.CommandHistory,
                if fin {
                        break;
                }
                if len(gline) == 0 {
                        pwd(gshPA);
                        continue;
                }
                prevline = gline;
                hix++;
        }
        return gshCtx
}
func main() {
        gshPA := syscall.ProcAttr {
                "", // the staring directory
                os.Environ(), // environ[]
                []uintptr{},
                nil, // OS specific
```

```
        }
        gshCtx := GshContext {
                "", // StartDir
                "", // GetLine
                []GChdirHistory{ {"",time.Now()} },
                gshPA,
                []GCommandHistory{ },
                false,
                []int{},
                syscall.Rusage{},
                "", // GshHomeDir
                Ttyid(),
        }
        gshCtx = script(gshCtx)
        fmt.Printf("%s\n",gshCtx.StartDir)
}
// TODO:
// - inter gsh communication, possibly running in remote hosts -- to k
// - merged histories of multiple parallel gsh sessions
// - alias as a function
// - instant alias end environ export to the permanent > ~/.gsh/gsh-al
// - retrieval PATH of files by its type
// - gsh as an IME
// - all commands have its subucomand after "---" symbol
// - filename expansion by "-find" command
// - history of ext code and output of each commoand
//---END--- (^-^)/
```

— 2020-0810 SatoxITS