

```

1 //<html>
2 /*<head>
3 <link rel=icon href=GShell-Logo05icon.png>
4 <meta charset=UTF-8>
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>GShell-0.1.7 by SatoxITS</title>
7 </head>
8 <span id=gsh>
9 <header id=header height=100px onclick="shiftBG();"
10 <div align=right><note>GShell version 0.1.7 // 2020-08-21 // SatoxITS</note></div>
11 </header>
12 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
13 */
14 */
15 <details id=overview><summary>Overview</summary><pre>
16 To be written
17 </pre></details>
18 */
19 */
20 <details id=index open><summary>Index</summary>
21 <pre onclick="document.getElementById('gocode').open=true;">
22
23 Implementation
24 Structures
25 <a href=#import>import</a>
26 <a href=#struct>struct</a>
27 Main functions
28 <a href=#comexpansion>str-expansion</a> // macro processor
29 <a href=#finder>finder</a> // builtin find + du
30 <a href=#grep>grep</a> // builtin grep + wc + cksum + ...
31 <a href=#plugin>plugin</a> // plugin commands
32 <a href=#ex-commands>system</a> // external commands
33 <a href=#builtin>builtin</a> // builtin commands
34 <a href=#network>network</a> // socket handler
35 <a href=#remote-sh>remote-sh</a> // remote shell
36 <a href=#redirect>redirect</a> // StdIn/Out redirection
37 <a href=#history>history</a> // command history
38 <a href=#rusage>rusage</a> // resource usage
39 <a href=#encode>encode</a> // encode / decode
40 <a href=#getline>getline</a> // line editor
41 <a href=#scanf>scanf</a> // string decomposer
42 <a href=#interpreter>interpreter</a> // command interpreter
43 <a href=#main>main</a><pre>
44 </details>
45 */
46 //<details open id=gocode><summary>Source Code</summary>
47 //<pre onclick="document.getElementById('gocode').open=false;">
48 // gsh - Go lang based Shell
49 // (c) 2020 ITS more Co., Ltd.
50 // 2020-0807 created by SatoxITS (sato@its-more.jp)
51
52 package main // gsh main
53 // <a name=import>Imported packages</a> // <a href=https://golang.org/pkg/>Packages</a>
54 import (
55     "fmt" // <a href=https://golang.org/pkg/fmt/>fmt</a>
56     "strings" // <a href=https://golang.org/pkg/strings/>strings</a>
57     "strconv" // <a href=https://golang.org/pkg/strconv/>strconv</a>
58     "sort" // <a href=https://golang.org/pkg/sort/>sort</a>
59     "time" // <a href=https://golang.org/pkg/time/>time</a>
60     "bufio" // <a href=https://golang.org/pkg/bufio/>bufio</a>
61     "io/ioutil" // <a href=https://golang.org/pkg/io/ioutil/>ioutil</a>
62     "os" // <a href=https://golang.org/pkg/os/>os</a>
63     "syscall" // <a href=https://golang.org/pkg/syscall/>syscall</a>
64     "plugin" // <a href=https://golang.org/pkg/plugin/>plugin</a>
65     "net" // <a href=https://golang.org/pkg/net/>net</a>
66     "net/http" // <a href=https://golang.org/pkg/net/http/>http</a>
67     "html" // <a href=https://golang.org/pkg/html/>html</a>
68     "path/filepath" // <a href=https://golang.org/pkg/path/filepath/>filepath</a>
69     "go/types" // <a href=https://golang.org/pkg/go/types/>types</a>
70     "go/token" // <a href=https://golang.org/pkg/go/token/>token</a>
71     "encoding/base64" // <a href=https://golang.org/pkg/encoding/base64/>base64</a>
72     //gshdata // gshell's logo and source code
73 )
74
75 var NAME = "gsh"
76 var VERSION = "0.1.7"
77 var DATE = "2020-0821"
78 var LINESIZE = (8*1024)
79 var PATHSEP = ";" // should be ";" in Windows
80 var DIRSEP = "/" // canbe \ in Windows
81 var GSH_HOME = ".gsh" // under home directory
82 var MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
83 var PROMPT = ">"
84
85 // -xX logging control
86 // --A-- all
87 // --I-- info.
88 // --D-- debug
89 // --T-- time and resource usage
90 // --W-- warning
91 // --E-- error
92 // --F-- fatal error
93 // --Xn- network
94
95 // <a name=struct>Structures</a>
96 type GCommandHistory struct {
97     StartAt time.Time // command line execution started at
98     EndAt time.Time // command line execution ended at
99     ResCode int // exit code of (external command)
100     CmdError error // error string
101     OutData *os.File // output of the command
102     FoundFile []string // output - result of ufind
103     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
104     CmdId int // maybe with identified with arguments or impact
105     // redirection commands should not be the CmdId
106     WorkDir string // working directory at start
107     WorkDirX int // index in ChdirHistory
108     CmdLine string // command line
109 }
110 type GChdirHistory struct {
111     Dir string
112     MovedAt time.Time
113     CmdIndex int
114 }
115 type CmdMode struct {
116     Background bool
117 }
118 type PluginInfo struct {
119     Spec *plugin.Plugin
120     Addr plugin.Symbol
121     Name string // maybe relative
122     Path string // this is in Plugin but hidden
123 }
124 type GServer struct {

```

```

125 host      string
126 port      string
127 }
128 type ValueStack [][]string
129 type GshContext struct {
130   StartDir    string // the current directory at the start
131   GetLine     string // gsh-getline command as a input line editor
132   ChdirHistory []GChdirHistory // the 1st entry is wd at the start
133   gshPA       syscall.ProcAttr
134   CommandHistory []GCommandHistory
135   CmdCurrent   GCommandHistory
136   Background  bool
137   BackgroundJobs []int
138   LastRusage   syscall.Rusage
139   GshHomeDir   string
140   TerminalId   int
141   CmdTrace     bool // should be [map]
142   CmdTime      bool // should be [map]
143   PluginFuncs []PluginInfo
144   iValues      []string
145   iDelimiter   string // field separator of print out
146   iFormat      string // default print format (of integer)
147   iValStack    ValueStack
148   LastServer   GServer
149 }
150
151 func strBegins(str, pat string) bool {
152   if len(pat) <= len(str) {
153     yes := str[0:len(pat)] == pat
154     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat, yes)
155     return yes
156   }
157   //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
158   return false
159 }
160 func isin(what string, list []string) bool {
161   for _, v := range list {
162     if v == what {
163       return true
164     }
165   }
166   return false
167 }
168 func isinX(what string, list []string) (int) {
169   for i, v := range list {
170     if v == what {
171       return i
172     }
173   }
174   return -1
175 }
176
177 func env(opts []string) {
178   env := os.Environ()
179   if isin("-s", opts) {
180     sort.Slice(env, func(i, j int) bool {
181       return env[i] < env[j]
182     })
183   }
184   for _, v := range env {
185     fmt.Printf("%v\n", v)
186   }
187 }
188
189 // - rewriting should be context dependent
190 // - should postpone until the real point of evaluation
191 // - should rewrite only known notation of symbol
192 func scanInt(str string)(val int, leng int){
193   leng = -1
194   for i, ch := range str {
195     if '0' <= ch && ch <= '9' {
196       leng = i + 1
197     } else {
198       break
199     }
200   }
201   if 0 < leng {
202     ival, _ := strconv.Atoi(str[0:leng])
203     return ival, leng
204   } else {
205     return 0, 0
206   }
207 }
208 func substHistory(gshCtx *GshContext, str string, i int, rstr string)(leng int, rstr string){
209   if len(str[i+1:]) == 0 {
210     return 0, rstr
211   }
212   hi := 0
213   histlen := len(gshCtx.CommandHistory)
214   if str[i+1] == '!' {
215     hi = histlen - 1
216     leng = 1
217   } else {
218     hi, leng = scanInt(str[i+1:])
219     if leng == 0 {
220       return 0, rstr
221     }
222     if hi < 0 {
223       hi = histlen + hi
224     }
225   }
226   if 0 <= hi && hi < histlen {
227     var ext byte
228     if 1 < len(str[i+leng:]) {
229       ext = str[i+leng:][1]
230     }
231     //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
232     if ext == 'f' {
233       leng += 1
234       xlist := []string{}
235       list := gshCtx.CommandHistory[hi].FoundFile
236       for _, v := range list {
237         //list[i] = escapeWhiteSP(v)
238         xlist = append(xlist, escapeWhiteSP(v))
239       }
240       //rstr += strings.Join(list, " ")
241       rstr += strings.Join(xlist, " ")
242     } else {
243       if ext == '@' || ext == 'd' {
244         // !N@ .. workdir at the start of the command
245         leng += 1
246         rstr += gshCtx.CommandHistory[hi].WorkDir
247       } else {
248         rstr += gshCtx.CommandHistory[hi].CmdLine
249       }

```

```

250     }else{
251         leng = 0
252     }
253     return leng,rstr
254 }
255 func escapeWhiteSP(str string)(string){
256     if len(str) == 0 {
257         return "\\z" // empty, to be ignored
258     }
259     rstr := ""
260     for _,ch := range str {
261         switch ch {
262             case '\\': rstr += "\\\\"
263             case ' ': rstr += "\\s"
264             case '\t': rstr += "\\t"
265             case '\r': rstr += "\\r"
266             case '\n': rstr += "\\n"
267             default: rstr += string(ch)
268         }
269     }
270     return rstr
271 }
272 func unescapeWhiteSP(str string)(string){ // strip original escapes
273     rstr := ""
274     for i := 0; i < len(str); i++ {
275         ch := str[i]
276         if ch == '\\' {
277             if i+1 < len(str) {
278                 switch str[i+1] {
279                     case 'z':
280                         continue;
281                 }
282             }
283         }
284         rstr += string(ch)
285     }
286     return rstr
287 }
288 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
289     ustrv := []string{}
290     for _,v := range strv {
291         ustrv = append(ustrv,unescapeWhiteSP(v))
292     }
293     return ustrv
294 }
295
296 // <a name=comexpansion>str-expansion</a>
297 // - this should be a macro processor
298 func strsubst(gshCtx *GshContext,str string,histonly bool) string {
299     rbuff := []byte{}
300     if false {
301         //@@@Unicode should be cared as a character
302         return str
303     }
304     //rstr := ""
305     inEsc := 0 // escape characer mode
306     for i := 0; i < len(str); i++ {
307         //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
308         ch := str[i]
309         if inEsc == 0 {
310             if ch == '|' {
311                 //leng,xrstr := substHistory(gshCtx,str,i,rstr)
312                 leng,rs := substHistory(gshCtx,str,i,"")
313                 if 0 < leng {
314                     //_,rs := substHistory(gshCtx,str,i,"")
315                     rbuff = append(rbuff,[]byte(rs)...)
316                     i += leng
317                     //rstr = xrstr
318                     continue
319                 }
320             }
321             switch ch {
322                 case '\\': inEsc = '\\'; continue
323                 //case '%': inEsc = '%'; continue
324                 case '$':
325                 }
326             }
327             switch inEsc {
328                 case '\\':
329                     switch ch {
330                         case '\\': ch = '\\'
331                         case 's': ch = ' '
332                         case 't': ch = '\t'
333                         case 'r': ch = '\r'
334                         case 'n': ch = '\n'
335                         case 'z': inEsc = 0; continue // empty, to be ignored
336                     }
337                 inEsc = 0
338                 case '%':
339                     switch {
340                         case ch == '%': ch = '%'
341                         case ch == 't':
342                             //rstr = rstr + time.Now().Format(time.Stamp)
343                             rs := time.Now().Format(time.Stamp)
344                             rbuff = append(rbuff,[]byte(rs)...)
345                             inEsc = 0
346                             continue;
347                         default:
348                             // postpone the interpretation
349                             //rstr = rstr + "%" + string(ch)
350                             rbuff = append(rbuff,ch)
351                             inEsc = 0
352                             continue;
353                     }
354                 inEsc = 0
355             }
356             //rstr = rstr + string(ch)
357             rbuff = append(rbuff,ch)
358         }
359         //fmt.Printf("--D--subst(%s)(%s)\n",str,string(rbuff))
360         return string(rbuff)
361         //return rstr
362     }
363 func showFileInfo(path string, opts []string) {
364     if isin("-l",opts) || isin("-ls",opts) {
365         fi, _ := os.Stat(path)
366         mod := fi.ModTime()
367         date := mod.Format(time.Stamp)
368         fmt.Printf("%v %8v %s ",fi.Mode(),fi.Size(),date)
369     }
370     fmt.Printf("%s",path)
371     if isin("-sp",opts) {
372         fmt.Printf(" ")
373     }else
374     if ! isin("-n",opts) {

```

```

375     fmt.Printf("\n")
376 }
377 }
378 func userHomeDir()(string,bool){
379     /*
380     homedir,_ = os.UserHomeDir() // not implemented in older Golang
381     */
382     homedir,found := os.LookupEnv("HOME")
383     //fmt.Printf("--I-- HOME=%v(%v)\n",homedir,found)
384     if !found {
385         return "/tmp",found
386     }
387     return homedir,found
388 }
389 }
390 func toFullpath(path string) (fullpath string) {
391     if path[0] == '/' {
392         return path
393     }
394     pathv := strings.Split(path,DIRSEP)
395     switch {
396     case pathv[0] == ".":
397         pathv[0],_ = os.Getwd()
398     case pathv[0] == "..": // all ones should be interpreted
399         cwd,_ = os.Getwd()
400         ppathv := strings.Split(cwd,DIRSEP)
401         pathv[0] = strings.Join(ppathv,DIRSEP)
402     case pathv[0] == "-":
403         pathv[0],_ = userHomeDir()
404     default:
405         cwd,_ = os.Getwd()
406         pathv[0] = cwd + DIRSEP + pathv[0]
407     }
408     return strings.Join(pathv,DIRSEP)
409 }
410 }
411 func IsRegFile(path string)(bool){
412     fi, err := os.Stat(path)
413     if err == nil {
414         fm := fi.Mode()
415         return fm.IsRegular();
416     }
417     return false
418 }
419 }
420 // <a name=encode>Encode / Decode</a>
421 // <a href=https://golang.org/pkg/encoding/base64/#example_NewEncoder>Encoder</a>
422 func Enc(gshCtx *GshContext,argv[]string)(*GshContext){
423     file := os.Stdin
424     buff := make([]byte,LINESIZE)
425     li := 0
426     encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
427     for li = 0; ; li++ {
428         count, err := file.Read(buff)
429         if count <= 0 {
430             break
431         }
432         if err != nil {
433             break
434         }
435         encoder.Write(buff[0:count])
436     }
437     encoder.Close()
438     return gshCtx
439 }
440 func Dec(gshCtx *GshContext,argv[]string)(*GshContext){
441     decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
442     li := 0
443     buff := make([]byte,LINESIZE)
444     for li = 0; ; li++ {
445         count, err := decoder.Read(buff)
446         if count <= 0 {
447             break
448         }
449         if err != nil {
450             break
451         }
452         os.Stdout.Write(buff[0:count])
453     }
454     return gshCtx
455 }
456 // lnspl [N] [-crlf][-C \\\]
457 func SplitLine(gshCtx *GshContext,argv[]string)(*GshContext){
458     reader := bufio.NewReaderSize(os.Stdin,64*1024)
459     ni := 0
460     toi := 0
461     for ni = 0; ; ni++ {
462         line, err := reader.ReadString('\n')
463         if len(line) <= 0 {
464             if err != nil {
465                 fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d (%v)\n",ni,toi,err)
466                 break
467             }
468         }
469         off := 0
470         ilen := len(line)
471         remlen := len(line)
472         for oi := 0; 0 < remlen; oi++ {
473             olen := remlen
474             addnl := false
475             if 72 < olen {
476                 olen = 72
477                 addnl = true
478             }
479             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
480                 toi,ni,oi,off,olen,remlen,ilen)
481             toi += 1
482             os.Stdout.Write([]byte(line[0:olen]))
483             if addnl {
484                 //os.Stdout.Write([]byte("\r\n"))
485                 os.Stdout.Write([]byte("\n"))
486                 os.Stdout.Write([]byte("\n"))
487             }
488             line = line[olen:]
489             off += olen
490             remlen -= olen
491         }
492     }
493     fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d\n",ni,toi)
494     return gshCtx
495 }
496 }
497 // <a name=grep>grep</a>
498 // "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
499 // a*,!ab,c, ... sequential combination of patterns

```

```

500 // what "LINE" is should be definable
501 // generic line-by-line processing
502 // grep [-v]
503 // cat -n -v
504 // uniq [-c]
505 // tail -f
506 // sed s/x/y/ or awk
507 // grep with line count like wc
508 // rewrite contents if specified
509 func (gsh*GshContext)xGrep(path string, rexpv[]string)(int){
510     file, err := os.OpenFile(path,os.O_RDONLY,0)
511     if err != nil {
512         fmt.Printf("--E-- grep %v (%v)\n",path,err)
513         return -1
514     }
515     defer file.Close()
516     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path, rexpv) }
517     //reader := bufio.NewReaderSize(file,LINESIZE)
518     reader := bufio.NewReaderSize(file,80)
519     li := 0
520     found := 0
521     for li = 0; ; li++ {
522         line, err := reader.ReadString('\n')
523         if len(line) <= 0 {
524             break
525         }
526         if 150 < len(line) {
527             // maybe binary
528             break;
529         }
530         if err != nil {
531             break
532         }
533         if 0 <= strings.Index(string(line),rexpv[0]) {
534             found += 1
535             fmt.Printf("%s:%d: %s",path,li,line)
536         }
537     }
538     //fmt.Printf("total %d lines %s\n",li,path)
539     //if( 0 < found ){ fmt.Printf("((found %d lines %s))\n",found,path); }
540     return found
541 }
542
543 // <a name=finder>Finder</a>
544 // finding files with it name and contents
545 // file names are ORED
546 // show the content with %x fmt list
547 // ls -R
548 // tar command by adding output
549 type fileSum struct {
550     Err int64 // access error or so
551     Size int64 // content size
552     DupSize int64 // content size from hard links
553     Blocks int64 // number of blocks (of 512 bytes)
554     DupBlocks int64 // Blocks pointed from hard links
555     HLinks int64 // hard links
556     Words int64
557     Lines int64
558     Files int64
559     Dirs int64 // the num. of directories
560     SymLink int64
561     Flats int64 // the num. of flat files
562     MaxDepth int64
563     MaxNamen int64 // max. name length
564     nextRepo time.Time
565 }
566 func showFusage(dir string,fusage *fileSum){
567     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
568     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
569
570     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
571         dir,
572         fusage.Files,
573         fusage.Dirs,
574         fusage.SymLink,
575         fusage.HLinks,
576         float64(fusage.Size)/1000000.0,bsume);
577 }
578 const (
579     S_IFMT = 0170000
580     S_IFCHR = 0020000
581     S_IFDIR = 0040000
582     S_IFREG = 0100000
583     S_IFLNK = 0120000
584     S_IFSOCK = 0140000
585 )
586 func cumPinfo(fsum *fileSum, path string, stater error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
587     now := time.Now()
588     if time.Second <= now.Sub(fsum.nextRepo) {
589         if !fsum.nextRepo.IsZero(){
590             tstamp := now.Format(time.Stamp)
591             showFusage(tstamp,fsum)
592         }
593         fsum.nextRepo = now.Add(time.Second)
594     }
595     if stater != nil {
596         fsum.Err += 1
597         return fsum
598     }
599     fsum.Files += 1
600     if 1 < fstat.Nlink {
601         // must count only once...
602         // at least ignore ones in the same directory
603         //if finfo.Mode().IsRegular() {
604         if (fstat.Mode & S_IFMT) == S_IFREG {
605             fsum.HLinks += 1
606             fsum.DupBlocks += int64(fstat.Blocks)
607             //fmt.Printf("---Dup HardLink %v %s\n",fstat.Nlink,path)
608         }
609     }
610     //fsum.Size += finfo.Size()
611     fsum.Size += fstat.Size
612     fsum.Blocks += int64(fstat.Blocks)
613     //if verb { fmt.Printf("(%8dBlk) %s",fstat.Blocks/2,path) }
614     if isin("-ls",argv){
615         //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }
616     // fmt.Printf("%d\t",fstat.Blocks/2)
617     }
618     //if finfo.IsDir()
619     if (fstat.Mode & S_IFMT) == S_IFDIR {
620         fsum.Dirs += 1
621     }
622     //if (finfo.Mode() & os.ModeSymlink) != 0
623     if (fstat.Mode & S_IFMT) == S_IFLNK {
624         //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }

```

```

625     //{ fmt.Printf("symlink(%o,%s)\n",fstat.Mode,finfo.Name()) }
626     fsum.SymLink += 1
627 }
628 return fsum
629 }
630 func (gsh*GshContext)xxFindEntv(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(*fileSum){
631     nols := isin("-grep",argv)
632     // sort entv
633     /*
634     if isin("-t",argv){
635         sort.Slice(filev, func(i,j int) bool {
636             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
637         })
638     }
639     */
640     /*
641     if isin("-u",argv){
642         sort.Slice(filev, func(i,j int) bool {
643             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
644         })
645     }
646     if isin("-U",argv){
647         sort.Slice(filev, func(i,j int) bool {
648             return 0 < filev[i].CreatTime().Sub(filev[j].CreatTime())
649         })
650     }
651     */
652     /*
653     if isin("-S",argv){
654         sort.Slice(filev, func(i,j int) bool {
655             return filev[j].Size() < filev[i].Size()
656         })
657     }
658     */
659     for _,filename := range entv {
660         for _,npat := range npatv {
661             match := true
662             if npat == "*" {
663                 match = true
664             }else{
665                 match, _ = filepath.Match(npat,filename)
666             }
667             path := dir + DIRSEP + filename
668             if !match {
669                 continue
670             }
671             var fstat syscall.Stat_t
672             staterr := syscall.Lstat(path,&fstat)
673             if staterr != nil {
674                 if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
675                 continue;
676             }
677             if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
678                 // should not show size of directory in "-du" mode ...
679             }else
680             if !nols && !isin("-s",argv) && ( !isin("-du",argv) || isin("-a",argv) ) {
681                 if isin("-du",argv) {
682                     fmt.Printf("%d\t",fstat.Blocks/2)
683                 }
684                 showFileInfo(path,argv)
685             }
686             if true { // && isin("-du",argv)
687                 total = cumFinfo(total,path,staterr,fstat,argv,false)
688             }
689             /*
690             if isin("-wc",argv) {
691             }
692             */
693             x := isinX("-grep",argv); // -grep will be convenient like -ls
694             if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls
695                 if IsRegFile(path){
696                     found := gsh.xGrep(path,argv[x+1:])
697                     if 0 < found {
698                         foundv := gsh.CmdCurrent.FoundFile
699                         if len(foundv) < 10 {
700                             gsh.CmdCurrent.FoundFile =
701                                 append(gsh.CmdCurrent.FoundFile,path)
702                         }
703                     }
704                 }
705             }
706             if !isin("-r0",argv) { // -d 0 in du, -depth n in find
707                 //total.Depth += 1
708                 if (fstat.Mode & S_IFMT) == S_IFLNK {
709                     continue
710                 }
711                 if dstat.Rdev != fstat.Rdev {
712                     fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
713                         dir,dstat.Rdev,path,fstat.Rdev)
714                 }
715                 if (fstat.Mode & S_IFMT) == S_IFDIR {
716                     total = gsh.xxFind(depth+1,total,path,npatv,argv)
717                 }
718             }
719         }
720     }
721     return total
722 }
723 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
724     nols := isin("-grep",argv)
725     dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
726     if oerr == nil {
727         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
728         defer dirfile.Close()
729     }else{
730     }
731 }
732 prev := *total
733 var dstat syscall.Stat_t
734 staterr := syscall.Lstat(dir,&dstat) // should be flstat
735 }
736 if staterr != nil {
737     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
738     return total
739 }
740 //filev,err := ioutil.ReadDir(dir)
741 //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
742 /*
743 if err != nil {
744     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
745     return total
746 }
747 */
748 if depth == 0 {
749     total = cumFinfo(total,dir,staterr,dstat,argv,true)

```

```

750     if !inols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
751         showFileInfo(dir,argv)
752     }
753 }
754 // it it is not a directory, just scan it and finish
755
756 for ei := 0; ; ei++ {
757     entv,x derr := dirfile.Readdirnames(8*1024)
758     if len(entv) == 0 || derr != nil {
759         //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
760         break
761     }
762     if 0 < ei {
763         fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
764     }
765     total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npats,argv)
766 }
767 if isin("-du",argv) {
768     // if in "du" mode
769     fmt.Printf("%d\t%s\n", (total.Blocks-prev.Blocks)/2,dir)
770 }
771 return total
772 }
773
774 // {ufind|fu|ls} [Files] [// Names] [-- Expressions]
775 // Files is "." by default
776 // Names is "*" by default
777 // Expressions is "-print" by default for "ufind", or -du for "fu" command
778 func (gsh*GshContext)xFind(argv[]string){
779     if 0 < len(argv) && strBegins(argv[0],"?"){
780         showFound(gsh,argv)
781         return
782     }
783     var total = fileSum{}
784     npats := []string{}
785     for _,v := range argv {
786         if 0 < len(v) && v[0] != '-' {
787             npats = append(npats,v)
788         }
789         if v == "/" { break }
790         if v == "--" { break }
791         if v == "-grep" { break }
792         if v == "-ls" { break }
793     }
794     if len(npats) == 0 {
795         npats = []string{"*"}
796     }
797     cwd := "."
798     // if to be fullpath ::: cwd, _ := os.Getwd()
799     if len(npats) == 0 { npats = []string{"*"} }
800     fusage := gsh.xxFind(0,total,cwd,npats,argv)
801     if !isin("-grep",argv) {
802         showFusage("total",fusage)
803     }
804     if !isin("-s",argv){
805         hits := len(gsh.CmdCurrent.FoundFile)
806         if 0 < hits {
807             fmt.Printf("--I-- %d files hits // can be refered with !%df\n",
808                 hits,len(gsh.CommandHistory))
809         }
810     }
811     return
812 }
813
814 func showFiles(files[]string){
815     sp := ""
816     for i,file := range files {
817         if 0 < i { sp = " " } else { sp = "" }
818         fmt.Printf(sp+"%s",escapeWhiteSP(file))
819     }
820 }
821
822 func showFound(gshCtx *GshContext, argv[]string){
823     for i,v := range gshCtx.CommandHistory {
824         if 0 < len(v.FoundFile) {
825             fmt.Printf("%d (%d) ",i,len(v.FoundFile))
826             if isin("-ls",argv){
827                 fmt.Printf("\n")
828                 for _,file := range v.FoundFile {
829                     fmt.Printf(" ") //sub number?
830                     showFileInfo(file,argv)
831                 }
832             }else{
833                 showFiles(v.FoundFile)
834                 fmt.Printf("\n")
835             }
836         }
837     }
838 }
839
840 func showMatchFile(filev []os.FileInfo, npat,dir string, argv[]string)(string,bool){
841     fname := ""
842     found := false
843     for _,v := range filev {
844         match, _ := filepath.Match(npat,(v.Name()))
845         if match {
846             fname = v.Name()
847             found = true
848             //fmt.Printf("[%d] %s\n",i,v.Name())
849             showIfExecutable(fname,dir,argv)
850         }
851     }
852     return fname,found
853 }
854
855 func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
856     var fullpath string
857     if strBegins(name,DIRSEP){
858         fullpath = name
859     }else{
860         fullpath = dir + DIRSEP + name
861     }
862     fi, err := os.Stat(fullpath)
863     if err != nil {
864         fullpath = dir + DIRSEP + name + ".go"
865         fi, err = os.Stat(fullpath)
866     }
867     if err == nil {
868         fm := fi.Mode()
869         if fm.IsRegular() {
870             // R_OK=4, W_OK=2, X_OK=1, F_OK=0
871             if syscall.Access(fullpath,5) == nil {
872                 ffullpath = fullpath
873                 ffound = true
874                 if ! isin("-s", argv) {
875                     showFileInfo(fullpath,argv)
876                 }
877             }
878         }
879     }
880 }

```

```

875     }
876     }
877 }
878 return ffullpath, ffound
879 }
880 func which(list string, argv []string) (fullpathv []string, itis bool){
881     if len(argv) <= 1 {
882         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
883         return []string{"", false
884     }
885     path := argv[1]
886     if strBegins(path, "/") {
887         // should check if executable?
888         _, exOK := showIfExecutable(path, "/", argv)
889         fmt.Printf("-D-- %v exOK=%v\n", path, exOK)
890         return []string{path}, exOK
891     }
892     pathenv, efound := os.LookupEnv(list)
893     if ! efound {
894         fmt.Printf("--E-- which: no \"%s\" environment\n", list)
895         return []string{"", false
896     }
897     showall := isin("-a", argv) || 0 < strings.Index(path, "*")
898     dirv := strings.Split(pathenv, PATHSEP)
899     ffound := false
900     ffullpath := path
901     for _, dir := range dirv {
902         if 0 < strings.Index(path, "*") { // by wild-card
903             list, _ := ioutil.ReadDir(dir)
904             ffullpath, ffound = showMatchFile(list, path, dir, argv)
905         }else{
906             ffullpath, ffound = showIfExecutable(path, dir, argv)
907         }
908         //if ffound && !isin("-a", argv) {
909         if ffound && !showall {
910             break;
911         }
912     }
913     return []string{ffullpath}, ffound
914 }
915 }
916 func stripLeadingWSParg(argv []string) ([]string){
917     for ; 0 < len(argv); {
918         if len(argv[0]) == 0 {
919             argv = argv[1:]
920         }else{
921             break
922         }
923     }
924     return argv
925 }
926 func xEval(argv []string, nlend bool){
927     argv = stripLeadingWSParg(argv)
928     if len(argv) == 0 {
929         fmt.Printf("eval [%%format] [Go-expression]\n")
930         return
931     }
932     pfmt := "%v"
933     if argv[0][0] == '$' {
934         pfmt = argv[0]
935         argv = argv[1:]
936     }
937     if len(argv) == 0 {
938         return
939     }
940     gocode := strings.Join(argv, " ");
941     //fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
942     fset := token.NewFileSet()
943     rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
944     fmt.Printf(pfmt, rval.Value)
945     if nlend { fmt.Printf("\n") }
946 }
947 }
948 func getval(name string) (found bool, val int) {
949     /* should expand the name here */
950     if name == "gsh.pid" {
951         return true, os.Getpid()
952     }else{
953         if name == "gsh.ppid" {
954             return true, os.Getppid()
955         }
956     }
957     return false, 0
958 }
959 }
960 func echo(argv []string, nlend bool){
961     for ai := 1; ai < len(argv); ai++ {
962         if 1 < ai {
963             fmt.Printf(" ");
964         }
965         arg := argv[ai]
966         found, val := getval(arg)
967         if found {
968             fmt.Printf("%d", val)
969         }else{
970             fmt.Printf("%s", arg)
971         }
972     }
973     if nlend {
974         fmt.Printf("\n");
975     }
976 }
977 }
978 func resfile() string {
979     return "gsh.tmp"
980 }
981 }
982 //var resF *File
983 func resmap() {
984     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
985     // https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/
986     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
987     if err != nil {
988         fmt.Printf("refF could not open: %s\n", err)
989     }else{
990         fmt.Printf("refF opened\n")
991     }
992 }
993 }
994 // @@2020-0821
995 func gshScanArg(str string, strip int)(argv []string){
996     var si = 0
997     var sb = 0
998     var inBracket = 0
999     var arg1 = make([]byte, LINESIZE)
1000    var ax = 0
1001    debug := false

```



```

1000
1001     for ; si < len(str); si++ {
1002         if str[si] != ' ' {
1003             break
1004         }
1005     }
1006     sb = si
1007     for ; si < len(str); si++ {
1008         if sb <= si {
1009             if debug {
1010                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1011                     inBracket,sb,si,argv[0:ax],str[si:])
1012             }
1013         }
1014         ch := str[si]
1015         if ch == '{' {
1016             inBracket += 1
1017             if 0 < strip && inBracket <= strip {
1018                 //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1019                 continue
1020             }
1021         }
1022         if 0 < inBracket {
1023             if ch == '}' {
1024                 inBracket -= 1
1025                 if 0 < strip && inBracket < strip {
1026                     //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1027                     continue
1028                 }
1029             }
1030             argl[ax] = ch
1031             ax += 1
1032             continue
1033         }
1034         if str[si] == ' ' {
1035             argv = append(argv,string(argl[0:ax]))
1036             if debug {
1037                 fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1038                     -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1039             }
1040             sb = si+1
1041             ax = 0
1042             continue
1043         }
1044         argl[ax] = ch
1045         ax += 1
1046     }
1047     if sb < si {
1048         argv = append(argv,string(argl[0:ax]))
1049         if debug {
1050             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1051                 -1+len(argv),sb,si,string(argl[0:ax]),string(str[si:]))
1052         }
1053     }
1054     if debug {
1055         fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,str,len(argv),argv)
1056     }
1057     return argv
1058 }
1059
1060 // should get stderr (into tmpfile ?) and return
1061 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1062     var pv = []int{-1,-1}
1063     syscall.Pipe(pv)
1064
1065     xarg := gshScanArg(name,1)
1066     name = strings.Join(xarg," ")
1067
1068     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
1069     pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
1070     fdix := 0
1071     dir := "?"
1072     if mode == "r" {
1073         dir = "<"
1074         fdix = 1 // read from the stdout of the process
1075     }else{
1076         dir = ">"
1077         fdix = 0 // write to the stdin of the process
1078     }
1079     gshPA := gsh.gshPA
1080     savfd := gshPA.Files[fdix]
1081
1082     var fd uintptr = 0
1083     if mode == "r" {
1084         fd = pout.Fd()
1085         gshPA.Files[fdix] = pout.Fd()
1086     }else{
1087         fd = pin.Fd()
1088         gshPA.Files[fdix] = pin.Fd()
1089     }
1090     fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1091     // should do this by Goroutine?
1092     gsh.BackGround = true
1093     gshell1(*gsh,name)
1094     gsh.BackGround = false
1095
1096     gshPA.Files[fdix] = savfd
1097     return pin,pout,false
1098 }
1099
1100 // <a name=ex-commands>External commands</a>
1101 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {
1102     if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1103
1104     gshPA := gsh.gshPA
1105     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1106     if itis == false {
1107         return true,false
1108     }
1109     fullpath := fullpathv[0]
1110     argv = unescapeWhiteSPV(argv)
1111     if 0 < strings.Index(fullpath,".go") {
1112         nargv := argv // []string{}
1113         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1114         if itis == false {
1115             fmt.Printf("--F-- Go not found\n")
1116             return false,true
1117         }
1118         gofullpath := gofullpathv[0]
1119         nargv = []string{ gofullpath, "run", fullpath }
1120         fmt.Printf("--I-- %s { %s %s %s }\n",gofullpath,
1121             nargv[0],nargv[1],nargv[2])
1122         if exec {
1123             syscall.Exec(gofullpath,nargv,os.Environ())
1124         }else{

```

```

1125     pid, _ := syscall.ForkExec(gofullpath,argv,&gshPA)
1126     if gsh.BackGround {
1127         fmt.Printf("--Ip- in Background pid[%d]\n",pid)
1128         gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1129     }else{
1130         rusage := syscall.Rusage {}
1131         syscall.Wait4(pid,nil,0,&rusage)
1132         gsh.LastRusage = rusage
1133         gsh.CmdCurrent.Rusage[1] = rusage
1134     }
1135 }
1136 }else{
1137     if exec {
1138         syscall.Exec(fullpath,argv,os.Environ())
1139     }else{
1140         pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1141         //fmt.Printf("[%d]\n",pid); // '*' to be background
1142         if gsh.BackGround {
1143             fmt.Printf("--Ip- in Background pid[%d]\n",pid)
1144             gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1145         }else{
1146             rusage := syscall.Rusage {}
1147             syscall.Wait4(pid,nil,0,&rusage);
1148             gsh.LastRusage = rusage
1149             gsh.CmdCurrent.Rusage[1] = rusage
1150         }
1151     }
1152 }
1153 return false,false
1154 }
1155 }
1156 // <a name=builtin>Builtin Commands</a>
1157 func sleep(gshCtx GshContext, argv []string) {
1158     if len(argv) < 2 {
1159         fmt.Printf("Sleep 100ms, 100us, 100ns, ...)\n")
1160         return
1161     }
1162     duration := argv[1];
1163     d, err := time.ParseDuration(duration)
1164     if err != nil {
1165         d, err = time.ParseDuration(duration+"s")
1166     }
1167     if err != nil {
1168         fmt.Printf("duration ? %s (%s)\n",duration,err)
1169         return
1170     }
1171     //fmt.Printf("Sleep %v\n",duration)
1172     time.Sleep(d)
1173     if 0 < len(argv[2:]) {
1174         gshellv(gshCtx, argv[2:])
1175     }
1176 }
1177 func repeat(gshCtx GshContext, argv []string) {
1178     if len(argv) < 2 {
1179         return
1180     }
1181     start0 := time.Now()
1182     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1183         if 0 < len(argv[2:]) {
1184             //start := time.Now()
1185             gshellv(gshCtx, argv[2:])
1186             end := time.Now()
1187             elps := end.Sub(start0);
1188             if( 1000000000 < elps ){
1189                 fmt.Printf("repeat#%d %v)\n",ri,elps);
1190             }
1191         }
1192     }
1193 }
1194 }
1195 func gen(gshCtx GshContext, argv []string) {
1196     gshPA := gshCtx.gshPA
1197     if len(argv) < 2 {
1198         fmt.Printf("Usage: %s N\n",argv[0])
1199         return
1200     }
1201     // should br repeated by "repeat" command
1202     count, _ := strconv.Atoi(argv[1])
1203     fd := gshPA.Files[1] // Stdout
1204     file := os.NewFile(fd,"internalStdOut")
1205     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1206     //buf := []byte{}
1207     outdata := "0123 5678 0123 5678 0123 5678 0123 5678"
1208     for gi := 0; gi < count; gi++ {
1209         file.WriteString(outdata)
1210     }
1211     //file.WriteString("\n")
1212     fmt.Printf("\n(%d B)\n",count*len(outdata));
1213     //file.Close()
1214 }
1215 }
1216 // <a anme=rexec>Remote Execution</a> // 2020-0820
1217 func Elapsed(from time.Time)(string){
1218     elps := time.Now().Sub(from)
1219     if 1000000000 < elps {
1220         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1221     }else
1222     if 1000000 < elps {
1223         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1224     }else{
1225         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1226     }
1227 }
1228 func absize(size int64)(string){
1229     fsize := float64(size)
1230     if 1024*1024*1024 < size {
1231         return fmt.Sprintf("%8.2fGiB",fsize/(1024*1024*1024))
1232     }else
1233     if 1024*1024 < size {
1234         return fmt.Sprintf("%8.3fMiB",fsize/(1024*1024))
1235     }else{
1236         return fmt.Sprintf("%8.3fKiB",fsize/1024)
1237     }
1238 }
1239 func abspeed(totalB int64,ns time.Duration)(string){
1240     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1241     if 1000 <= MBs {
1242         return fmt.Sprintf("%6.3fGBps",MBs/1000)
1243     }
1244     if 1 <= MBs {
1245         return fmt.Sprintf("%6.3fMBps",MBs)
1246     }else{
1247         return fmt.Sprintf("%6.3fKBps",MBs*1000)
1248     }
1249 }

```

```

1250 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1251     Start := time.Now()
1252     buff := make([]byte,bsiz)
1253     var total int64 = 0
1254     var rem int64 = size
1255     nio := 0
1256     Prev := time.Now()
1257     var PrevSize int64 = 0
1258
1259     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1260         what,absize(total),size,nio)
1261
1262     for i:= 0; ; i++ {
1263         var len = bsiz
1264         if int(rem) < len {
1265             len = int(rem)
1266         }
1267         Now := time.Now()
1268         Elps := Now.Sub(Prev);
1269         if 1000000000 < Now.Sub(Prev) {
1270             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1271                 what,absize(total),size,nio,
1272                 abspeed((total-PrevSize),Elps))
1273             Prev = Now;
1274             PrevSize = total
1275         }
1276         rlen := len
1277         if in != nil {
1278             // should watch the disconnection of out
1279             rcc,err := in.Read(buff[0:rlen])
1280             if err != nil {
1281                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1282                     what,rcc,err,in.Name())
1283                 break
1284             }
1285             rlen = rcc
1286             if string(buff[0:10]) == "(SoftEOF " {
1287                 var ecc int64 = 0
1288                 fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1289                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/&v\n",
1290                     what,ecc,total)
1291                 if ecc == total {
1292                     break
1293                 }
1294             }
1295         }
1296         wlen := rlen
1297         if out != nil {
1298             wcc,err := out.Write(buff[0:rlen])
1299             if err != nil {
1300                 fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1301                     what,wcc,err,out.Name())
1302                 break
1303             }
1304             wlen = wcc
1305         }
1306         if wlen < rlen {
1307             fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1308                 what,wlen,rlen)
1309             break;
1310         }
1311
1312         nio += 1
1313         total += int64(rlen)
1314         rem -= int64(rlen)
1315         if rem <= 0 {
1316             break
1317         }
1318     }
1319     Done := time.Now()
1320     Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1321     TotalMB := float64(total)/1000000 //MB
1322     MBps := TotalMB / Elps
1323     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %v.3fMB/s\n",
1324         what,total,size,nio,absize(total),MBps)
1325     return total
1326 }
1327 func (gsh*GshContext)RexecServer(argv[]string){
1328     debug := true
1329     Start0 := time.Now()
1330     Start := Start0
1331     // if local == ""; { local = "0.0.0.0:9999" }
1332     local := "0.0.0.0:9999"
1333
1334     if 0 < len(argv) {
1335         if argv[0] == "-s" {
1336             debug = false
1337             argv = argv[1:]
1338         }
1339     }
1340     if 0 < len(argv) {
1341         argv = argv[1:]
1342     }
1343     port, err := net.ResolveTCPAddr("tcp",local);
1344     if err != nil {
1345         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1346         return
1347     }
1348     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1349     sconn, err := net.ListenTCP("tcp", port)
1350     if err != nil {
1351         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1352         return
1353     }
1354
1355     reqbuf := make([]byte,LINESIZE)
1356     res := ""
1357     for {
1358         fmt.Printf(Elapsed(Start0)+"--In- S: Accepting at %s...\n",local);
1359         aconn, err := sconn.AcceptTCP()
1360         Start = time.Now()
1361         if err != nil {
1362             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1363             return
1364         }
1365         cInt, _ := aconn.File()
1366         fd := CInt.Fd()
1367         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d]\n",local,fd) }
1368         res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1369         fmt.Fprint(cInt,"%s",res)
1370         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1371         count, err := cInt.Read(reqbuf)
1372         if err != nil {
1373             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1374                 count,err,string(reqbuf))

```

```

1375     }
1376     req := string(reqbuf[:count])
1377     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v", string(req)) }
1378     reqv := strings.Split(string(req), "\r")
1379     cmdv := gshScanArg(reqv[0], 0)
1380     //cmdv := strings.Split(reqv[0], " ")
1381     switch cmdv[0] {
1382     case "HELLO":
1383         res = fmt.Sprintf("250 %v", req)
1384     case "GET":
1385         // download {remotefile|-zN} [localfile]
1386         var dsize int64 = 32*1024*1024
1387         var bsize int = 64*1024
1388         var fname string = ""
1389         var in *os.File = nil
1390         var pseudoEOF = false
1391         if 1 < len(cmdv) {
1392             fname = cmdv[1]
1393             if strBegins(fname, "-z") {
1394                 fmt.Sscanf(fname[2:], "%d", &dsize)
1395             } else {
1396                 if strBegins(fname, "{") {
1397                     xin, xout, err := gsh.Popen(fname, "r")
1398                     if err {
1399                         } else {
1400                             xout.Close()
1401                             defer xin.Close()
1402                             in = xin
1403                             dsize = MaxStreamSize
1404                             pseudoEOF = true
1405                         }
1406                     } else {
1407                         xin, err := os.Open(fname)
1408                         if err != nil {
1409                             fmt.Printf("--En- GET (%v)\n", err)
1410                         } else {
1411                             defer xin.Close()
1412                             in = xin
1413                             fi, _ := xin.Stat()
1414                             dsize = fi.Size()
1415                         }
1416                     }
1417                 }
1418                 //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n", dsize, bsize)
1419                 res = fmt.Sprintf("200 %v\r\n", dsize)
1420                 fmt.Fprintf(clnt, "%v", res)
1421                 fmt.Printf(Elapsed(Start)+"--In- S: %v", res)
1422                 wcount := fileRelay("SendGET", in, clnt, dsize, bsize)
1423                 if pseudoEOF {
1424                     // show end of stream data (its size) by OOB?
1425                     time.Sleep(100*1000*1000)
1426                     SoftEOF := fmt.Sprintf("((SoftEOF %v))", wcount)
1427                     fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n", SoftEOF)
1428                     fmt.Fprintf(clnt, "%v\r\n", SoftEOF)
1429                     // with client generated random?
1430                 }
1431                 res = fmt.Sprintf("200 GET done\r\n")
1432             case "PUT":
1433                 // upload {srcfile|-zN} [dstfile]
1434                 var dsize int64 = 32*1024*1024
1435                 var bsize int = 64*1024
1436                 var fname string = ""
1437                 var out *os.File = nil
1438                 if 1 < len(cmdv) { // localfile
1439                     fmt.Sscanf(cmdv[1], "%d", &dsize)
1440                 }
1441                 if 2 < len(cmdv) {
1442                     fname = cmdv[2]
1443                     if fname == "-" {
1444                         // nul dev
1445                     } else {
1446                         if strBegins(fname, "{") {
1447                             xin, xout, err := gsh.Popen(fname, "w")
1448                             if err {
1449                                 } else {
1450                                     xin.Close()
1451                                     defer xout.Close()
1452                                     out = xout
1453                                 }
1454                             } else {
1455                                 // should write to temporary file
1456                                 // should suppress ^C on tty
1457                                 xout, err := os.OpenFile(fname, os.O_CREATE|os.O_RDWR|os.O_TRUNC, 0600)
1458                                 //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n", fname, xout, err)
1459                                 if err != nil {
1460                                     fmt.Printf("--En- PUT (%v)\n", err)
1461                                 } else {
1462                                     out = xout
1463                                 }
1464                             }
1465                             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1466                                 fname, local, err)
1467                         }
1468                         fmt.Printf(Elapsed(Start)+"--In- PUT %v (/v)\n", dsize, bsize)
1469                         fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n", dsize)
1470                         fmt.Fprintf(clnt, "200 %v OK\r\n", dsize)
1471                         fileRelay("RecvPUT", clnt, out, dsize, bsize)
1472                         res = fmt.Sprintf("200 PUT done\r\n")
1473                     default:
1474                         res = fmt.Sprintf("400 What? %v", req)
1475                     }
1476                     clnt.Write([]byte(res))
1477                     fmt.Printf(Elapsed(Start)+"--In- S: %v", res)
1478                     aconn.Close();
1479                     clnt.Close();
1480                 }
1481                 sconn.Close();
1482             }
1483         func (gsh *GshContext) RexecClient(argv []string) {
1484             debug := true
1485             Start := time.Now()
1486             if len(argv) == 1 {
1487                 return
1488             }
1489             argv = argv[1:]
1490             if argv[0] == "-serv" {
1491                 gsh.RexecServer(argv[1:])
1492                 return
1493             }
1494             remote := "0.0.0.0:9999"
1495             if argv[0][0] == 'e' {
1496                 remote = argv[0][1:]
1497                 argv = argv[1:]
1498             }
1499             if argv[0] == "-s" {

```

```

1500     debug = false
1501     argv = argv[1:]
1502 }
1503 dport, err := net.ResolveTCPAddr("tcp",remote);
1504 if err != nil {
1505     fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1506     return
1507 }
1508 fmt.Printf(Elapsed(Start)+"--In- C: Socket: connecting to %s\n",remote)
1509 serv, err := net.DialTCP("tcp",nil,dport)
1510 if err != nil {
1511     fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1512     return
1513 }
1514 if debug { fmt.Printf(Elapsed(Start)+"--In- C: Socket: connected to %s\n",remote) }
1515
1516 req := ""
1517 res := make([]byte,LINESIZE)
1518 count,err := serv.Read(res)
1519 if err != nil {
1520     fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1521 }
1522 if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1523
1524 if argv[0] == "GET" {
1525     savPA := gsh.gshPA
1526     var bsize int = 64*1024
1527     req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1528     fmt.Printf(Elapsed(Start)+"--In- C: %v",req)
1529     fmt.Fprintf(serv,req)
1530     count,err = serv.Read(res)
1531     if err != nil {
1532     }else{
1533         var dsize int64 = 0
1534         var out *os.File = nil
1535         var out_tobeopened *os.File = nil
1536         var fname string = ""
1537         var rcode int = 0
1538         var pid int = -1
1539         fmt.Sscanf(string(res),"%d %d",&rcode,&dsize)
1540         fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count]))
1541         if 3 <= len(argv) {
1542             fname = argv[2]
1543             if strBegins(fname,"{") {
1544                 xin,xout,err := gsh.Popen(fname,"w")
1545                 if err {
1546                 }else{
1547                     xin.Close()
1548                     defer xout.Close()
1549                     out = xout
1550                     out_tobeopened = xout
1551                     pid = 0 // should be its pid
1552                 }
1553             }else{
1554                 // should write to temporary file
1555                 // should suppress ^C on tty
1556                 xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1557                 if err != nil {
1558                     fmt.Print("--En- %v\n",err)
1559                 }
1560                 out = xout
1561             }
1562         }
1563         in, _ := serv.File()
1564         fileRelay("RecvGET",in,out,dsize,bsize)
1565         if 0 <= pid {
1566             gsh.gshPA = savPA // recovery of Fd(), and more?
1567             fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
1568             out_tobeopened.Close()
1569             //syscall.Wait4(pid,nil,0,nil) //@@
1570         }
1571     }
1572 }else
1573 if argv[0] == "PUT" {
1574     remote, _ := serv.File()
1575     var local *os.File = nil
1576     var dsize int64 = 32*1024*1024
1577     var bsize int = 64*1024
1578     var ofile string = "-"
1579     //fmt.Printf("--I-- Rex %v\n",argv)
1580     if 1 < len(argv) {
1581         fname := argv[1]
1582         if strBegins(fname,"-z") {
1583             fmt.Sscanf(fname[2:],"%d",&dsize)
1584         }else
1585         if strBegins(fname,"{") {
1586             xin,xout,err := gsh.Popen(fname,"r")
1587             if err {
1588             }else{
1589                 xout.Close()
1590                 defer xin.Close()
1591                 //in = xin
1592                 local = xin
1593                 fmt.Printf("--In- [%d] < Upload output of %v\n",
1594                     local.Fd(),fname)
1595                 ofile = "-from."+fname
1596                 dsize = MaxStreamSize
1597             }
1598         }else{
1599             xlocal,err := os.Open(fname)
1600             if err != nil {
1601                 fmt.Printf("--En- (%s)\n",err)
1602                 local = nil
1603             }else{
1604                 local = xlocal
1605                 fi, _ := local.Stat()
1606                 dsize = fi.Size()
1607                 defer local.Close()
1608                 //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsize)
1609             }
1610             ofile = fname
1611             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
1612                 fname,dsize,local,err)
1613         }
1614     }
1615     if 2 < len(argv) && argv[2] != "" {
1616         ofile = argv[2]
1617         //fmt.Printf("(%d)%v B.ofile=%v\n",len(argv),argv,ofile)
1618     }
1619     //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
1620     fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1621     req = fmt.Sprintf("PUT %v %v \r\n",dsize,ofile)
1622     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
1623     fmt.Fprintf(serv,"%v",req)
1624     count,err = serv.Read(res)

```

```

1625     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v", string(res[0:count])) }
1626     fileRelay("SendPUT", local, remote, dsize, bsize)
1627 }else{
1628     req = fmt.Sprintf("%v\r\n", strings.Join(argv, " "))
1629     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v", req) }
1630     fmt.Fprintf(serv, "%v", req)
1631     //fmt.Printf("--In- sending RexRequest(%v)\n", len(req))
1632 }
1633 //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
1634 count,err = serv.Read(res)
1635 res := ""
1636 if count == 0 {
1637     res = "(nil)\r\n"
1638 }else{
1639     res = string(res[:count])
1640 }
1641 if err != nil {
1642     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v", count,err,res)
1643 }else{
1644     fmt.Printf(Elapsed(Start)+"--In- S: %v", res)
1645 }
1646 serv.Close()
1647 //conn.Close()
1648 }
1649
1650 // <a name=remote-sh>Remote Shell</a>
1651 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
1652 func (gsh*GshContext)FileCopy(argv []string){
1653     var host = ""
1654     var port = ""
1655     var upload = false
1656     var download = false
1657     var xargv = []string{"rex-gcp"}
1658     var srcv = []string{}
1659     var dstv = []string{}
1660     argv = argv[1:]
1661
1662     for _,v := range argv {
1663         /*
1664         if v[0] == '-' { // might be a pseudo file (generated date)
1665             continue
1666         }
1667         */
1668         obj := strings.Split(v,":")
1669         //fmt.Printf("%d %v %v\n",len(obj),v,obj)
1670         if 1 < len(obj) {
1671             host = obj[0]
1672             file := ""
1673             if 0 < len(host) {
1674                 gsh.LastServer.host = host
1675             }else{
1676                 host = gsh.LastServer.host
1677                 port = gsh.LastServer.port
1678             }
1679             if 2 < len(obj) {
1680                 port = obj[1]
1681                 if 0 < len(port) {
1682                     gsh.LastServer.port = port
1683                 }else{
1684                     port = gsh.LastServer.port
1685                 }
1686                 file = obj[2]
1687             }else{
1688                 file = obj[1]
1689             }
1690             if len(srcv) == 0 {
1691                 download = true
1692                 srcv = append(srcv,file)
1693                 continue
1694             }
1695             upload = true
1696             dstv = append(dstv,file)
1697             continue
1698         }
1699         /*
1700         idx := strings.Index(v,":")
1701         if 0 <= idx {
1702             remote = v[0:idx]
1703             if len(srcv) == 0 {
1704                 download = true
1705                 srcv = append(srcv,v[idx+1:])
1706                 continue
1707             }
1708             upload = true
1709             dstv = append(dstv,v[idx+1:])
1710             continue
1711         }
1712         */
1713         if download {
1714             dstv = append(dstv,v)
1715         }else{
1716             srcv = append(srcv,v)
1717         }
1718     }
1719     hostport := "@" + host + ":" + port
1720     if upload {
1721         if host != "" { xargv = append(xargv,hostport) }
1722         xargv = append(xargv,"PUT")
1723         xargv = append(xargv,srcv[0:]...)
1724         xargv = append(xargv,dstv[0:]...)
1725         //fmt.Printf("--I-- FileCopy PUT gsh://%s/%v < %v // %v\n",hostport,dstv,srcv,xargv)
1726         fmt.Printf("--I-- FileCopy PUT gsh://%s/%v < %v\n",hostport,dstv,srcv)
1727         gsh.RexecClient(xargv)
1728     }else{
1729         if download {
1730             if host != "" { xargv = append(xargv,hostport) }
1731             xargv = append(xargv,"GET")
1732             xargv = append(xargv,srcv[0:]...)
1733             xargv = append(xargv,dstv[0:]...)
1734             //fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v // %v\n",hostport,srcv,dstv,xargv)
1735             fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v\n",hostport,srcv,dstv)
1736             gsh.RexecClient(xargv)
1737         }else{
1738         }
1739     }
1740 }
1741 // <a name=network>network</a>
1742 // -s, -si, -so // bi-directional, source, sync (maybe socket)
1743 func sconnect(gshCtx GshContext, inTCP bool, argv []string) {
1744     gshPA := gshCtx.gshPA
1745     if len(argv) < 2 {
1746         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
1747         return
1748     }
1749     remote := argv[1]

```

```

1750 if remote == ":" { remote = "0.0.0.0:9999" }
1751
1752 if inTCP { // TCP
1753     dport, err := net.ResolveTCPAddr("tcp",remote);
1754     if err != nil {
1755         fmt.Printf("Address error: %s (%s)\n",remote,err)
1756         return
1757     }
1758     conn, err := net.DialTCP("tcp",nil,dport)
1759     if err != nil {
1760         fmt.Printf("Connection error: %s (%s)\n",remote,err)
1761         return
1762     }
1763     file, _ := conn.File();
1764     fd := file.Fd()
1765     fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
1766
1767     savfd := gshPA.Files[1]
1768     gshPA.Files[1] = fd;
1769     gshelly(gshCtx, argv[2:])
1770     gshPA.Files[1] = savfd
1771     file.Close()
1772     conn.Close()
1773 }else{
1774     //dport, err := net.ResolveUDPAddr("udp4",remote);
1775     dport, err := net.ResolveUDPAddr("udp",remote);
1776     if err != nil {
1777         fmt.Printf("Address error: %s (%s)\n",remote,err)
1778         return
1779     }
1780     //conn, err := net.DialUDP("udp4",nil,dport)
1781     conn, err := net.DialUDP("udp",nil,dport)
1782     if err != nil {
1783         fmt.Printf("Connection error: %s (%s)\n",remote,err)
1784         return
1785     }
1786     file, _ := conn.File();
1787     fd := file.Fd()
1788
1789     ar := conn.RemoteAddr()
1790     //al := conn.LocalAddr()
1791     fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
1792         remote,ar.String(),fd)
1793
1794     savfd := gshPA.Files[1]
1795     gshPA.Files[1] = fd;
1796     gshelly(gshCtx, argv[2:])
1797     gshPA.Files[1] = savfd
1798     file.Close()
1799     conn.Close()
1800 }
1801 }
1802 func saccept(gshCtx GshContext, inTCP bool, argv []string) {
1803     gshPA := gshCtx.gshPA
1804     if len(argv) < 2 {
1805         fmt.Printf("Usage: -ac [host]:[port].udp]\n")
1806         return
1807     }
1808     local := argv[1]
1809     if local == ":" { local = "0.0.0.0:9999" }
1810     if inTCP { // TCP
1811         port, err := net.ResolveTCPAddr("tcp",local);
1812         if err != nil {
1813             fmt.Printf("Address error: %s (%s)\n",local,err)
1814             return
1815         }
1816         //fmt.Printf("Listen at %s...\n",local);
1817         sconn, err := net.ListenTCP("tcp", port)
1818         if err != nil {
1819             fmt.Printf("Listen error: %s (%s)\n",local,err)
1820             return
1821         }
1822         //fmt.Printf("Accepting at %s...\n",local);
1823         aconn, err := sconn.AcceptTCP()
1824         if err != nil {
1825             fmt.Printf("Accept error: %s (%s)\n",local,err)
1826             return
1827         }
1828         file, _ := aconn.File()
1829         fd := file.Fd()
1830         fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
1831
1832         savfd := gshPA.Files[0]
1833         gshPA.Files[0] = fd;
1834         gshelly(gshCtx, argv[2:])
1835         gshPA.Files[0] = savfd
1836
1837         sconn.Close();
1838         aconn.Close();
1839         file.Close();
1840     }else{
1841         //port, err := net.ResolveUDPAddr("udp4",local);
1842         port, err := net.ResolveUDPAddr("udp",local);
1843         if err != nil {
1844             fmt.Printf("Address error: %s (%s)\n",local,err)
1845             return
1846         }
1847         fmt.Printf("Listen UDP at %s...\n",local);
1848         //uconn, err := net.ListenUDP("udp4", port)
1849         uconn, err := net.ListenUDP("udp", port)
1850         if err != nil {
1851             fmt.Printf("Listen error: %s (%s)\n",local,err)
1852             return
1853         }
1854         file, _ := uconn.File()
1855         fd := file.Fd()
1856         ar := uconn.RemoteAddr()
1857         remote := ""
1858         if ar != nil { remote = ar.String() }
1859         if remote == "" { remote = "?" }
1860
1861         // not yet received
1862         //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")
1863
1864         savfd := gshPA.Files[0]
1865         gshPA.Files[0] = fd;
1866         savenv := gshPA.Env
1867         gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
1868         gshelly(gshCtx, argv[2:])
1869         gshPA.Env = savenv
1870         gshPA.Files[0] = savfd
1871
1872         uconn.Close();
1873         file.Close();
1874     }

```

```

1875 }
1876
1877 // empty line command
1878 func xPwd(gshCtx GshContext, argv []string){
1879 // execute context command, pwd + date
1880 // context notation, representation scheme, to be resumed at re-login
1881 cwd, _ := os.Getwd()
1882 switch {
1883 case isin("-a",argv):
1884 gshCtx.ShowChdirHistory(argv)
1885 case isin("-ls",argv):
1886 showFileInfo(cwd,argv)
1887 default:
1888 fmt.Printf("%s\n",cwd)
1889 case isin("-v",argv): // obsolete empty command
1890 t := time.Now()
1891 date := t.Format(time.UnixDate)
1892 exe, _ := os.Executable()
1893 host, _ := os.Hostname()
1894 fmt.Printf("PWD-\`%s\`",cwd)
1895 fmt.Printf(" HOST-\`%s\`",host)
1896 fmt.Printf(" DATE-\`%s\`",date)
1897 fmt.Printf(" TIME-\`%s\`",t.String())
1898 fmt.Printf(" PID-\`%d\`",os.Getpid())
1899 fmt.Printf(" EXE-\`%s\`",exe)
1900 fmt.Printf("\n")
1901 }
1902 }
1903
1904 // <a name=history>History</a>
1905 // these should be browsed and edited by HTTP browser
1906 // show the time of command with -t and direcotry with -ls
1907 // openfile-history, sort by -a -m -c
1908 // sort by elapsed time by -t -s
1909 // search by "more" like interface
1910 // edit history
1911 // sort history, and wc or uniq
1912 // CPU and other resource consumptions
1913 // limit showing range (by time or so)
1914 // export / import history
1915 func xHistory(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
1916 atWorkDirX := -1
1917 if 1 < len(argv) && strBegins(argv[1],"@") {
1918 atWorkDirX, _ = strconv.Atoi(argv[1][1:])
1919 }
1920 //fmt.Printf("--D-- showHistory(%v)\n",argv)
1921 for i, v := range gshCtx.CommandHistory {
1922 // exclude commands not to be listed by default
1923 // internal commands may be suppressed by default
1924 if v.CmdLine == "" && !isin("-a",argv) {
1925 continue;
1926 }
1927 if 0 <= atWorkDirX {
1928 if v.WorkDirX != atWorkDirX {
1929 continue
1930 }
1931 }
1932 if !isin("-n",argv){ // like "fc"
1933 fmt.Printf("%!%-2d ",i)
1934 }
1935 if isin("-v",argv){
1936 fmt.Println(v) // should be with it date
1937 }else{
1938 if isin("-l",argv) || isin("-l0",argv) {
1939 elps := v.EndAt.Sub(v.StartAt);
1940 start := v.StartAt.Format(time.Stamp)
1941 fmt.Printf("@%d ",v.WorkDirX)
1942 fmt.Printf("[%v] %11v/t ",start,elps)
1943 }
1944 if isin("-l",argv) && !isin("-l0",argv){
1945 fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
1946 }
1947 if isin("-at",argv) { // isin("-ls",argv){
1948 dhi := v.WorkDirX // workdir history index
1949 fmt.Printf("@%d %s\t",dhi,v.WorkDir)
1950 // show the FileInfo of the output command??
1951 }
1952 fmt.Printf("%s",v.CmdLine)
1953 fmt.Printf("\n")
1954 }
1955 }
1956 return gshCtx
1957 }
1958 // !n - history index
1959 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
1960 if gline[0] == '!' {
1961 hix, err := strconv.Atoi(gline[1:])
1962 if err != nil {
1963 fmt.Printf("--E-- (%s : range)\n",hix)
1964 return "", false, true
1965 }
1966 if hix < 0 || len(gshCtx.CommandHistory) <= hix {
1967 fmt.Printf("--E-- (%d : out of range)\n",hix)
1968 return "", false, true
1969 }
1970 return gshCtx.CommandHistory[hix].CmdLine, false, false
1971 }
1972 // search
1973 //for i, v := range gshCtx.CommandHistory {
1974 //}
1975 return gline, false, false
1976 }
1977
1978 // temporary adding to PATH environment
1979 // cd name -lib for LD_LIBRARY_PATH
1980 // chdir with directory history (date + full-path)
1981 // -s for sort option (by visit date or so)
1982 func (gsh*GshContext)ShowChdirHistory1(i int,v GChdirHistory, argv []string){
1983 fmt.Printf("%!%-2d ",v.CmdIndex) // the first command at this WorkDir
1984 fmt.Printf("@%d ",i)
1985 fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
1986 showFileInfo(v.Dir,argv)
1987 }
1988 func (gsh*GshContext)ShowChdirHistory(argv []string){
1989 for i, v := range gsh.ChdirHistory {
1990 gsh.ShowChdirHistory1(i,v,argv)
1991 }
1992 }
1993 func skipOpts(argv []string)(int){
1994 for i,v := range argv {
1995 if strBegins(v,"-") {
1996 }else{
1997 return i
1998 }
1999 }

```



```

2000     return -1
2001 }
2002 func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
2003     cdhist := gshCtx.CkdirHistory
2004     if isin("?",argv) || isin("-t",argv) || isin("-a",argv) {
2005         gshCtx.ShowChdirHistory(argv)
2006         return gshCtx
2007     }
2008     pwd, _ := os.Getwd()
2009     dir := ""
2010     if len(argv) <= 1 {
2011         dir = toFullpath("-")
2012     }else{
2013         i := skipOpts(argv[1:])
2014         if i < 0 {
2015             dir = toFullpath("-")
2016         }else{
2017             dir = argv[1+i]
2018         }
2019     }
2020     if strBegins(dir,"@") {
2021         if dir == "@0" { // obsolete
2022             dir = gshCtx.StartDir
2023         }else
2024         if dir == "@!" {
2025             index := len(cdhist) - 1
2026             if 0 < index { index -- 1 }
2027             dir = cdhist[index].Dir
2028         }else{
2029             index, err := strconv.Atoi(dir[1:])
2030             if err != nil {
2031                 fmt.Printf("--E-- xChdir(%v)\n",err)
2032                 dir = "?"
2033             }else
2034             if len(gshCtx.CkdirHistory) <= index {
2035                 fmt.Printf("--E-- xChdir(history range error)\n")
2036                 dir = "?"
2037             }else{
2038                 dir = cdhist[index].Dir
2039             }
2040         }
2041     }
2042     if dir != "?" {
2043         err := os.Chdir(dir)
2044         if err != nil {
2045             fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)
2046         }else{
2047             cwd, _ := os.Getwd()
2048             if cwd != pwd {
2049                 hist1 := GChdirHistory { }
2050                 hist1.Dir = cwd
2051                 hist1.MovedAt = time.Now()
2052                 hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2053                 gshCtx.CkdirHistory = append(cdhist,hist1)
2054                 if !isin("-s",argv){
2055                     //cwd, _ := os.Getwd()
2056                     //fmt.Printf("%s\n",cwd)
2057                     ix := len(gshCtx.CkdirHistory)-1
2058                     gshCtx.ShowChdirHistory1(ix,hist1,argv)
2059                 }
2060             }
2061         }
2062     }
2063     if isin("-ls",argv){
2064         cwd, _ := os.Getwd()
2065         showFileInfo(cwd,argv);
2066     }
2067     return gshCtx
2068 }
2069 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2070     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2071 }
2072 func RusageSubv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2073     TimeValSub(&ru1[0].Utime,&ru2[0].Utime)
2074     TimeValSub(&ru1[0].Stime,&ru2[0].Stime)
2075     TimeValSub(&ru1[1].Utime,&ru2[1].Utime)
2076     TimeValSub(&ru1[1].Stime,&ru2[1].Stime)
2077     return ru1
2078 }
2079 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2080     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2081     return tvs
2082 }
2083 /*
2084 func RusageAddv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2085     TimeValAdd(ru1[0].Utime,ru2[0].Utime)
2086     TimeValAdd(ru1[0].Stime,ru2[0].Stime)
2087     TimeValAdd(ru1[1].Utime,ru2[1].Utime)
2088     TimeValAdd(ru1[1].Stime,ru2[1].Stime)
2089     return ru1
2090 }
2091 */
2092
2093 // <a name=rusage>Resource Usage</a>
2094 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2095     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2096     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2097     fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2098     fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2099     return ""
2100 }
2101 func GetRusagev(){[2]syscall.Rusage){
2102     var ruv = [2]syscall.Rusage{}
2103     syscall.GetRusage(syscall.RUSAGE_SELF,&ruv[0])
2104     syscall.GetRusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2105     return ruv
2106 }
2107 func showRusage(what string,argv []string, ru *syscall.Rusage){
2108     fmt.Printf("%s: ",what);
2109     fmt.Printf(" Utr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2110     fmt.Printf(" Sst=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2111     fmt.Printf(" Rss=%vB",ru.Maxrss)
2112     if isin("-l",argv) {
2113         fmt.Printf(" MinFlt=%v",ru.Minflt)
2114         fmt.Printf(" MajFlt=%v",ru.Majflt)
2115         fmt.Printf(" Ixrss=%vB",ru.Ixrss)
2116         fmt.Printf(" IdRSS=%vB",ru.Idrss)
2117         fmt.Printf(" Nswap=%vB",ru.Nswap)
2118         fmt.Printf(" Read=%v",ru.Inblock)
2119         fmt.Printf(" Write=%v",ru.Oublock)
2120     }
2121     fmt.Printf(" Snd=%v",ru.Msgsnd)
2122     fmt.Printf(" Rcv=%v",ru.Msgrcv)
2123     //if isin("-l",argv) {
2124         fmt.Printf(" Sig=%v",ru.Nsignals)

```

```

2125 //}
2126 fmt.Printf("\n");
2127 }
2128 func xTime(gshCtx GshContext, argv[]string)(GshContext,bool){
2129     if 2 <= len(argv){
2130         gshCtx.LastRusage = syscall.Rusage{
2131             rusage1 := Getrusagev()
2132             XgshCtx, fin := gshellv(gshCtx,argv[1:])
2133             rusage2 := Getrusagev()
2134             gshCtx = XgshCtx
2135             showRusage(argv[1],argv,&gshCtx.LastRusage)
2136             rusage := RusageSubv(rusage2,rusage1)
2137             showRusage("self",argv,&rusage[0])
2138             showRusage("chld",argv,&rusage[1])
2139             return gshCtx, fin
2140         }else{
2141             rusage:= syscall.Rusage {}
2142             syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2143             showRusage("self",argv, &rusage)
2144             syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2145             showRusage("chld",argv, &rusage)
2146             return gshCtx, false
2147         }
2148     }
2149     func xJobs(gshCtx GshContext, argv[]string){
2150         fmt.Printf("%d Jobs\n",len(gshCtx.BackgroundJobs))
2151         for ji, pid := range gshCtx.BackgroundJobs {
2152             //wstat := syscall.WaitStatus {0}
2153             rusage := syscall.Rusage {}
2154             //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2155             wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2156             if err != nil {
2157                 fmt.Printf("--E-- %d [%d] (%v)\n",ji,pid,err)
2158             }else{
2159                 fmt.Printf("%d [%d] (%d)\n",ji,pid,wpid)
2160                 showRusage("chld",argv,&rusage)
2161             }
2162         }
2163     }
2164     func inBackground(gshCtx GshContext, argv[]string)(GshContext,bool){
2165         if gshCtx.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2166         gshCtx.Background = true // set background option
2167         xfin := false
2168         gshCtx, xfin = gshellv(gshCtx,argv)
2169         gshCtx.Background = false
2170         return gshCtx,xfin
2171     }
2172     // -o file without command means just opening it and refer by #N
2173     // should be listed by "files" command
2174     func xOpen(gshCtx GshContext, argv[]string)(GshContext){
2175         var pv = [int{-1,-1}]
2176         err := syscall.Pipe(pv)
2177         fmt.Printf("--I-- pipe()=[%d,%d](%v)\n",pv[0],pv[1],err)
2178         return gshCtx
2179     }
2180     func fromPipe(gshCtx GshContext, argv[]string)(GshContext){
2181         return gshCtx
2182     }
2183     func xClose(gshCtx GshContext, argv[]string)(GshContext){
2184         return gshCtx
2185     }
2186
2187     // <a name=redirect>redirect</a>
2188     func redirect(gshCtx GshContext, argv[]string)(GshContext,bool){
2189         if len(argv) < 2 {
2190             return gshCtx, false
2191         }
2192
2193         cmd := argv[0]
2194         fname := argv[1]
2195         var file *os.File = nil
2196
2197         fdix := 0
2198         mode := os.O_RDONLY
2199
2200         switch {
2201             case cmd == "-i" || cmd == "<":
2202                 fdix = 0
2203                 mode = os.O_RDONLY
2204             case cmd == "-o" || cmd == ">":
2205                 fdix = 1
2206                 mode = os.O_RDWR | os.O_CREATE
2207             case cmd == "-a" || cmd == ">>":
2208                 fdix = 1
2209                 mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2210         }
2211         if fname[0] == '#' {
2212             fd, err := strconv.Atoi(fname[1:])
2213             if err != nil {
2214                 fmt.Printf("--E-- (%v)\n",err)
2215                 return gshCtx, false
2216             }
2217             file = os.NewFile(uintptr(fd),"MaybePipe")
2218         }else{
2219             xfile, err := os.OpenFile(argv[1], mode, 0600)
2220             if err != nil {
2221                 fmt.Printf("--E-- (%s)\n",err)
2222                 return gshCtx, false
2223             }
2224             file = xfile
2225         }
2226         gshPA := gshCtx.gshPA
2227         savfd := gshPA.Files[fdix]
2228         gshPA.Files[fdix] = file.Fd()
2229         fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2230         gshCtx, _ = gshellv(gshCtx, argv[2:])
2231         gshPA.Files[fdix] = savfd
2232
2233         return gshCtx, false
2234     }
2235
2236     //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2237     func httpHandler(res http.ResponseWriter, req *http.Request){
2238         path := req.URL.Path
2239         fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2240         {
2241             gshCtx, _ := setupGshContext()
2242             fmt.Printf("--I-- %s\n",path[1:])
2243             gshCtx, _ = tgshell1(gshCtx,path[1:])
2244         }
2245         fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
2246     }
2247     func httpServer(gshCtx GshContext, argv []string){
2248         http.HandleFunc("/", httpHandler)
2249         accport := "localhost:9999"

```

```

2250     fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2251     http.ListenAndServe(accport,nil)
2252 }
2253 func xGo(gshCtx GshContext, argv[]string){
2254     go gshellv(gshCtx,argv[1:]);
2255 }
2256 func xPs(gshCtx GshContext, argv[]string)(GshContext){
2257     return gshCtx
2258 }
2259 }
2260 // <a name=plugin>Plugin</a>
2261 // plugin [-ls [names]] to list plugins
2262 // Reference: <a href=https://golang.org/src/plugin/>plugin</a> source code
2263 func whichPlugin(gshCtx GshContext,name string,argv[]string)(pi *PluginInfo){
2264     pi = nil
2265     for _,p := range gshCtx.PluginFuncs {
2266         if p.Name == name && pi == nil {
2267             pi = *p
2268         }
2269         if !isin("-s",argv){
2270             //fmt.Printf("%v %v ",i,p)
2271             if isin("-ls",argv){
2272                 showFileInfo(p.Path,argv)
2273             }else{
2274                 fmt.Printf("%s\n",p.Name)
2275             }
2276         }
2277     }
2278     return pi
2279 }
2280 func xPlugin(gshCtx GshContext, argv[]string)(GshContext,error){
2281     if len(argv) == 0 || argv[0] == "-ls" {
2282         whichPlugin(gshCtx,"",argv)
2283         return gshCtx, nil
2284     }
2285     name := argv[0]
2286     pin := whichPlugin(gshCtx,name,[]string{"-s"})
2287     if pin != nil {
2288         os.Args = argv // should be recovered?
2289         pin.Addr.(func())()
2290         return gshCtx,nil
2291     }
2292     sofile := toFullpath(argv[0] + ".so") // or find it by which($PATH)
2293 }
2294 p, err := plugin.Open(sofile)
2295 if err != nil {
2296     fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2297     return gshCtx, err
2298 }
2299 fname := "Main"
2300 f, err := p.Lookup(fname)
2301 if( err != nil ){
2302     fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2303     return gshCtx, err
2304 }
2305 pin := PluginInfo {p,f,name,sofile}
2306 gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2307 fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2308 }
2309 //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2310 os.Args = argv
2311 f.(func())()
2312 return gshCtx, err
2313 }
2314 func Args(gshCtx *GshContext, argv[]string){
2315     for i,v := range os.Args {
2316         fmt.Printf("[%v] %v\n",i,v)
2317     }
2318 }
2319 func Version(gshCtx *GshContext, argv[]string){
2320     if isin("-l",argv) {
2321         fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2322     }else{
2323         fmt.Printf("%v",VERSION);
2324     }
2325     if !isin("-n",argv) {
2326         fmt.Printf("\n")
2327     }
2328 }
2329 }
2330 // <a name=scanf>Scanf</a> // string decomposer
2331 // scanf [format] [input]
2332 func scanf(sstr string)(strv[]string){
2333     strv = strings.Split(sstr, " ")
2334     return strv
2335 }
2336 func scanUntil(src,end string)(rstr string,leng int){
2337     idx := strings.Index(src,end)
2338     if 0 <= idx {
2339         rstr = src[0:idx]
2340         return rstr,idx+len(end)
2341     }
2342     return src,0
2343 }
2344 }
2345 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting
2346 func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
2347     //vint,err := strconv.Atoi(vstr)
2348     var ival int64 = 0
2349     n := 0
2350     err := error(nil)
2351     if strBegins(vstr, "-") {
2352         vx, _ := strconv.Atoi(vstr[1:])
2353         if vx < len(gsh.iValues) {
2354             vstr = gsh.iValues[vx]
2355         }else{
2356         }
2357     }
2358     // should use Eval()
2359     if strBegins(vstr,"0x") {
2360         n,err = fmt.Sscanf(vstr[2:], "%x",&ival)
2361     }else{
2362         n,err = fmt.Sscanf(vstr,"%d",&ival)
2363     }//fmt.Printf("--D-- n=%d err=(%v) {%s}=%v\n",n,err,vstr, ival)
2364 }
2365 if n == 1 && err == nil {
2366     //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
2367     fmt.Printf("%"+fmts,ival)
2368 }else{
2369     if isin("-bn",optv){
2370         fmt.Printf("%"+fmts,filepath.Base(vstr))
2371     }else{
2372         fmt.Printf("%"+fmts,vstr)
2373     }
2374 }

```

```

2375 }
2376 func (gsh*GshContext)printfv(fmts,div string,argv[]string,optv[]string,list[]string){
2377 //fmt.Printf("%d",len(list))
2378 //curfmt := "v"
2379 outlen := 0
2380 curfmt := gsh.iFormat
2381
2382 if 0 < len(fmts) {
2383 for xi := 0; xi < len(fmts); xi++ {
2384 fch := fmts[xi]
2385 if fch == '%' {
2386 if xi+1 < len(fmts) {
2387 curfmt = string(fmts[xi+1])
2388 gsh.iFormat = curfmt
2389 xi += 1
2390 if xi+1 < len(fmts) && fmts[xi+1] == '(' {
2391 vals, leng := scanUntil(fmts[xi+2:],")")
2392 //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n",curfmt,vals,leng)
2393 gsh.printVal(curfmt,vals,optv)
2394 xi += 2+leng-1
2395 outlen += 1
2396 }
2397 continue
2398 }
2399 if fch == '.' {
2400 hi, leng := scanInt(fmts[xi+1:])
2401 if 0 < leng {
2402 if hi < len(gsh.iValues) {
2403 gsh.printVal(curfmt,gsh.iValues[hi],optv)
2404 outlen += 1 // should be the real length
2405 }else{
2406 fmt.Printf("((out-range))")
2407 }
2408 xi += leng
2409 continue;
2410 }
2411 }
2412 }
2413 fmt.Printf("%c",fch)
2414 outlen += 1
2415 }
2416 }else{
2417 //fmt.Printf("--D-- print (%s)\n")
2418 for i,v := range list {
2419 if 0 < i {
2420 fmt.Printf(div)
2421 }
2422 gsh.printVal(curfmt,v,optv)
2423 outlen += 1
2424 }
2425 }
2426 if 0 < outlen {
2427 fmt.Printf("\n")
2428 }
2429 }
2430 func (gsh*GshContext)Scanv(argv[]string){
2431 //fmt.Printf("--D-- Scnav(%v)\n",argv)
2432 if len(argv) == 1 {
2433 return
2434 }
2435 argv = argv[1:]
2436 fmts := ""
2437 if strBegins(argv[0],"-F") {
2438 fmts = argv[0]
2439 gsh.iDelimiter = fmts
2440 argv = argv[1:]
2441 }
2442 input := strings.Join(argv," ")
2443 if fmts == "" { // simple decomposition
2444 v := scanv(input)
2445 gsh.iValues = v
2446 //fmt.Printf("%v\n",strings.Join(v," "))
2447 }else{
2448 v := make([]string,8)
2449 n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
2450 fmt.Printf("--D-- Sscanf ->(%v) n=%d err=(%v)\n",v,n,err)
2451 gsh.iValues = v
2452 }
2453 }
2454 func (gsh*GshContext)Printv(argv[]string){
2455 if false { //@@U
2456 fmt.Printf("%v\n",strings.Join(argv[1:], " "))
2457 return
2458 }
2459 //fmt.Printf("--D-- Printv(%v)\n",argv)
2460 //fmt.Printf("%v\n",strings.Join(gsh.iValues," "))
2461 div := gsh.iDelimiter
2462 fmts := ""
2463 argv = argv[1:]
2464 if 0 < len(argv) {
2465 if strBegins(argv[0],"-F") {
2466 div = argv[0][2:]
2467 argv = argv[1:]
2468 }
2469 }
2470
2471 optv := []string{}
2472 for _,v := range argv {
2473 if strBegins(v,"-"){
2474 optv = append(optv,v)
2475 argv = argv[1:]
2476 }else{
2477 break;
2478 }
2479 }
2480 if 0 < len(argv) {
2481 fmts = strings.Join(argv," ")
2482 }
2483 gsh.printfv(fmts,div,argv,optv,gsh.iValues)
2484 }
2485 func (gsh*GshContext)Basename(argv[]string){
2486 for i,v := range gsh.iValues {
2487 gsh.iValues[i] = filepath.Base(v)
2488 }
2489 }
2490 func (gsh*GshContext)Sortv(argv[]string){
2491 sv := gsh.iValues
2492 sort.Slice(sv, func(i,j int) bool {
2493 return sv[i] < sv[j]
2494 })
2495 }
2496 func (gsh*GshContext)Shiftv(argv[]string){
2497 vi := len(gsh.iValues)
2498 if 0 < vi {
2499 if isin("-r",argv) {

```

```

2500     top := gsh.iValues[0]
2501     gsh.iValues = append(gsh.iValues[1:],top)
2502 }else{
2503     gsh.iValues = gsh.iValues[1:]
2504 }
2505 }
2506 }
2507
2508 func (gsh*GshContext)Eng(argv[]string){
2509 }
2510 func (gsh*GshContext)Deq(argv[]string){
2511 }
2512 func (gsh*GshContext)Push(argv[]string){
2513     gsh.iValStack = append(gsh.iValStack,argv[1:])
2514     fmt.Printf("depth=%d\n",len(gsh.iValStack))
2515 }
2516 func (gsh*GshContext)Dump(argv[]string){
2517     for i,v := range gsh.iValStack {
2518         fmt.Printf("%d %v\n",i,v)
2519     }
2520 }
2521 func (gsh*GshContext)Pop(argv[]string){
2522     depth := len(gsh.iValStack)
2523     if 0 < depth {
2524         v := gsh.iValStack[depth-1]
2525         if isin("-cat",argv){
2526             gsh.iValues = append(gsh.iValues,v...)
2527         }else{
2528             gsh.iValues = v
2529         }
2530         gsh.iValStack = gsh.iValStack[0:depth-1]
2531         fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
2532     }else{
2533         fmt.Printf("depth=%d\n",depth)
2534     }
2535 }
2536
2537 // <a name=interpreter>Command Interpreter</a>
2538 func gshellv(gshCtx GshContext, argv []string) (_ GshContext, fin bool) {
2539     fin = false
2540
2541     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
2542     if len(argv) <= 0 {
2543         return gshCtx, false
2544     }
2545     xargv := []string{}
2546     for ai := 0; ai < len(argv); ai++ {
2547         xargv = append(xargv,subst(&gshCtx,argv[ai],false))
2548     }
2549     argv = xargv
2550     if false {
2551         for ai := 0; ai < len(argv); ai++ {
2552             fmt.Printf("[%d] %s [%d]T\n",
2553                 ai,argv[ai],len(argv[ai]),argv[ai])
2554         }
2555     }
2556     cmd := argv[0]
2557     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv),argv) }
2558     switch { // https://tour.golang.org/flowcontrol/11
2559     case cmd == "":
2560     case cmd == "-x":
2561         gshCtx.CmdTrace = ! gshCtx.CmdTrace
2562     case cmd == "-xt":
2563         gshCtx.CmdTime = ! gshCtx.CmdTime
2564     case cmd == "-ot":
2565         sconnect(gshCtx, true, argv)
2566     case cmd == "-ou":
2567         sconnect(gshCtx, false, argv)
2568     case cmd == "-it":
2569         saccept(gshCtx, true , argv)
2570     case cmd == "-iu":
2571         saccept(gshCtx, false, argv)
2572     case cmd == "i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
2573         redirect(gshCtx, argv)
2574     case cmd == "|":
2575         gshCtx = fromPipe(gshCtx, argv)
2576     case cmd == "args":
2577         Args(&gshCtx,argv)
2578     case cmd == "bg" || cmd == "-bg":
2579         rgshCtx, rfin := inBackground(gshCtx,argv[1:])
2580         return rgshCtx, rfin
2581     case cmd == "-bn":
2582         gshCtx.Basename(argv)
2583     case cmd == "call":
2584         _ = gshCtx.excommand(false,argv[1:])
2585     case cmd == "cd" || cmd == "chdir":
2586         gshCtx = xChdir(gshCtx,argv);
2587     case cmd == "close":
2588         gshCtx = xClose(gshCtx,argv)
2589     case cmd == "gcp":
2590         gshCtx.FileCopy(argv)
2591     case cmd == "dec" || cmd == "decode":
2592         Dec(&gshCtx,argv)
2593     case cmd == "#define":
2594     case cmd == "dump":
2595         gshCtx.Dump(argv)
2596     case cmd == "echo":
2597         echo(argv,true)
2598     case cmd == "enc" || cmd == "encode":
2599         Enc(&gshCtx,argv)
2600     case cmd == "env":
2601         env(argv)
2602     case cmd == "eval":
2603         xEval(argv[1:],true)
2604     case cmd == "exec":
2605         _ = gshCtx.excommand(true,argv[1:])
2606         // should not return here
2607     case cmd == "exit" || cmd == "quit":
2608         // write Result code EXIT to 3>
2609         return gshCtx, true
2610     case cmd == "fdls":
2611         // dump the attributes of fds (of other process)
2612     case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
2613         gshCtx.xFind(argv[1:])
2614     case cmd == "fu":
2615         gshCtx.xFind(argv[1:])
2616     case cmd == "fork":
2617         // mainly for a server
2618     case cmd == "-gen":
2619         gen(gshCtx, argv)
2620     case cmd == "-go":
2621         xGo(gshCtx, argv)
2622     case cmd == "-grep":
2623         gshCtx.xFind(argv)
2624

```

```

2625 case cmd == "gdeg":
2626     gshCtx.Deg(argv)
2627 case cmd == "geng":
2628     gshCtx.Eng(argv)
2629 case cmd == "gpop":
2630     gshCtx.Pop(argv)
2631 case cmd == "gpush":
2632     gshCtx.Push(argv)
2633 case cmd == "history" || cmd == "hi": // hi should be alias
2634     gshCtx = xHistory(gshCtx, argv)
2635 case cmd == "jobs":
2636     xJobs(gshCtx,argv)
2637 case cmd == "lnx":
2638     SplitLine(&gshCtx,argv)
2639 case cmd == "ls":
2640     gshCtx.xFind(argv)
2641 case cmd == "nop":
2642     // do nothing
2643 case cmd == "pipe":
2644     gshCtx = xOpen(gshCtx,argv)
2645 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
2646     gshCtx, _ = xPlugin(gshCtx,argv[1:])
2647 case cmd == "print" || cmd == "-pr":
2648     // output internal slice // also sprintf should be
2649     gshCtx.Printv(argv)
2650 case cmd == "ps":
2651     xPs(gshCtx,argv)
2652 case cmd == "pstitle":
2653     // to be gsh.title
2654 case cmd == "rexecl" || cmd == "rexd":
2655     gshCtx.RexecServer(argv)
2656 case cmd == "rexecl" || cmd == "rex":
2657     gshCtx.RexecClient(argv)
2658 case cmd == "repeat" || cmd == "rep": // repeat cond command
2659     repeat(gshCtx,argv)
2660 case cmd == "scan":
2661     // scan input (or so in fscanf) to internal slice (like Files or map)
2662     gshCtx.Scanv(argv)
2663 case cmd == "set":
2664     // set name ...
2665 case cmd == "serv":
2666     httpServer(gshCtx,argv)
2667 case cmd == "shift":
2668     gshCtx.Shiftv(argv)
2669 case cmd == "sleep":
2670     sleep(gshCtx,argv)
2671 case cmd == "-sort":
2672     gshCtx.Sortv(argv)
2673 case cmd == "time":
2674     gshCtx, fin = xTime(gshCtx,argv)
2675 case cmd == "pwd":
2676     xPwd(gshCtx,argv);
2677 case cmd == "ver" || cmd == "-ver" || cmd == "version":
2678     Version(&gshCtx,argv)
2679 case cmd == "where":
2680     // data file or so?
2681 case cmd == "which":
2682     which("PATH",argv);
2683 default:
2684     if whichPlugin(gshCtx,cmd,[jstring{"-s"}]) != nil {
2685         gshCtx, _ = xPlugin(gshCtx,argv)
2686     }else{
2687         notfound, _ := gshCtx.excommand(false,argv)
2688         if notfound {
2689             fmt.Printf("--E-- command not found (%v)\n",cmd)
2690         }
2691     }
2692 }
2693 return gshCtx, fin
2694 }
2695
2696 func gshelll(gshCtx GshContext, gline string) (gx GshContext, rfin bool) {
2697     argv := strings.Split(string(gline)," ")
2698     gshCtx, fin := gshellv(gshCtx,argv)
2699     return gshCtx, fin
2700 }
2701 func tgshelll(gshCtx GshContext, gline string) (gx GshContext, xfin bool) {
2702     start := time.Now()
2703     gshCtx, fin := gshelll(gshCtx,gline)
2704     end := time.Now()
2705     elps := end.Sub(start);
2706     if gshCtx.CmdTime {
2707         fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + "(&d.%09ds)\n",
2708             elps/1000000000,elps%1000000000)
2709     }
2710     return gshCtx, fin
2711 }
2712 func Ttyid() (int) {
2713     fi, err := os.Stdin.Stat()
2714     if err != nil {
2715         return 0;
2716     }
2717     //fmt.Printf("Stdin: %v Dev=%d\n",
2718     // fi.Mode(),fi.Mode(&os.ModeDevice)
2719     if (fi.Mode() & os.ModeDevice) != 0 {
2720         stat := syscall.Stat_t{};
2721         err := syscall.Fstat(0,&stat)
2722         if err != nil {
2723             //fmt.Printf("--I-- Stdin: (%v)\n",err)
2724         }else{
2725             //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
2726             // stat.Rdev&0xFF,stat.Rdev);
2727             //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
2728             return int(stat.Rdev & 0xFF)
2729         }
2730     }
2731     return 0
2732 }
2733 func ttyfile(gshCtx GshContext) string {
2734     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
2735     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
2736         fmt.Sprintf("%02d",gshCtx.TerminalId)
2737     //strconv.Itoa(gshCtx.TerminalId)
2738     //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
2739     return ttyfile
2740 }
2741 func ttyline(gshCtx GshContext) (*os.File){
2742     file, err := os.OpenFile(ttyfile(gshCtx),
2743         os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
2744     if err != nil {
2745         fmt.Printf("--F-- cannot open %s (%s)\n",ttyfile(gshCtx),err)
2746         return file;
2747     }
2748     return file
2749 }

```

```

2750 // <a name=getline>Command Line Editor</a>
2751 func getline(gshCtx GshContext, hix int, skipping, with_exgetline bool, gsh_getlinev[]string, prevline string) (string) {
2752     if( skipping {
2753         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
2754         line, _, _ := reader.ReadLine()
2755         return string(line)
2756     }else
2757     if( with_exgetline && gshCtx.GetLine != "" ){
2758         //var xhix int64 = int64(hix); // cast
2759         newenv := os.Environ()
2760         newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
2761
2762         tty := ttyline(gshCtx)
2763         tty.WriteString(prevline)
2764         Pa := os.ProcAttr {
2765             "", // start dir
2766             newenv, //os.Environ(),
2767             []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
2768             nil,
2769         }
2770 //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
2771 proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
2772 if err != nil {
2773     fmt.Printf("--F-- getline process error (%v)\n",err)
2774     // for ; { }
2775     return "exit (getline program failed)"
2776 }
2777 //stat, err := proc.Wait()
2778 proc.Wait()
2779 buff := make([]byte,LINESIZE)
2780 count, err := tty.Read(buff)
2781 //err = tty.Read(buff)
2782 //fmt.Printf("--D-- getline (%d)\n",count)
2783 if err != nil {
2784     if ! (count == 0) { // && err.String() == "EOF" ) {
2785         fmt.Printf("--E-- getline error (%s)\n",err)
2786     }
2787 }else{
2788     //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
2789 }
2790 tty.Close()
2791 gline := string(buff[0:count])
2792 return gline
2793 }else{
2794     // if isatty {
2795     fmt.Printf("%d",hix)
2796     fmt.Print(PROMPT)
2797     // }
2798     reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
2799     line, _, _ := reader.ReadLine()
2800     return string(line)
2801 }
2802 }
2803 //
2804 // $USERHOME/.gsh/
2805 // gsh-rc.txt, or gsh-configure.txt
2806 // gsh-history.txt
2807 // gsh-aliases.txt // should be conditional?
2808 //
2809 func gshSetupHomedir(gshCtx GshContext) (GshContext, bool) {
2810     homedir,found := userHomeDir()
2811     if !found {
2812         fmt.Printf("--E-- You have no UserHomeDir\n")
2813         return gshCtx, true
2814     }
2815     gshhome := homedir + "/" + GSH_HOME
2816     _, err2 := os.Stat(gshhome)
2817     if err2 != nil {
2818         err3 := os.Mkdir(gshhome,0700)
2819         if err3 != nil {
2820             fmt.Printf("--E-- Could not Create %s (%s)\n",
2821                 gshhome,err3)
2822             return gshCtx, true
2823         }
2824         fmt.Printf("--I-- Created %s\n",gshhome)
2825     }
2826     gshCtx.GshHomeDir = gshhome
2827     return gshCtx, false
2828 }
2829 func setupGshContext()(GshContext,bool){
2830     gshPA := syscall.ProcAttr {
2831         "", // the staring directory
2832         os.Environ(), // environ[]
2833         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
2834         nil, // OS specific
2835     }
2836     cwd, _ := os.Getwd()
2837     gshCtx := GshContext {
2838         cwd, // StartDir
2839         "", // GetLine
2840         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
2841         gshPA,
2842         []GCommandHistory{}, //something for invokation?
2843         GCommandHistory{}, // CmdCurrent
2844         false,
2845         []int{},
2846         syscall.Rusage{},
2847         "", // GshHomeDir
2848         Ttyid(),
2849         false,
2850         false,
2851         []PluginInfo{},
2852         []string{},
2853         "",
2854         "v",
2855         ValueStack{},
2856         GServer{"",""}, // LastServer
2857     }
2858     err := false
2859     gshCtx, err = gshSetupHomedir(gshCtx)
2860     return gshCtx, err
2861 }
2862 // <a name=main>Main loop</a>
2863 func script(gshCtxGiven *GshContext) (_ GshContext) {
2864     gshCtx,err0 := setupGshContext()
2865     if err0 {
2866         return gshCtx;
2867     }
2868     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
2869     //resmap()
2870     gsh_getlinev, with_exgetline :=
2871     which("PATH",[]string{"which","gsh-getline","-s"})
2872     if with_exgetline {
2873         gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
2874         gshCtx.GetLine = toFullpath(gsh_getlinev[0])

```

```

2875 }else{
2876 fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
2877 }
2878
2879 ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
2880 gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
2881
2882 prevline := ""
2883 skipping := false
2884 for hix := len(gshCtx.CommandHistory); ; {
2885     gline := getline(gshCtx,hix,skipping,with_exgetline,gsh_getlinev,prevline)
2886     if skipping {
2887         if strings.Index(gline,"fi") == 0 {
2888             fmt.Printf("fi\n");
2889             skipping = false;
2890         }else{
2891             //fmt.Printf("%s\n",gline);
2892         }
2893         continue
2894     }
2895     if strings.Index(gline,"if") == 0 {
2896         //fmt.Printf("--D-- if start: %s\n",gline);
2897         skipping = true;
2898         continue
2899     }
2900     if false {
2901         os.Stdout.Write([]byte("gotline:"))
2902         os.Stdout.Write([]byte(gline))
2903         os.Stdout.Write([]byte("\n"))
2904     }
2905     gline = strsubst(&gshCtx,gline,true)
2906     if false {
2907         fmt.Printf("fmt.Printf %%v - %v\n",gline)
2908         fmt.Printf("fmt.Printf %%s - %s\n",gline)
2909         fmt.Printf("fmt.Printf %%x - %x\n",gline)
2910         fmt.Printf("fmt.Printf %%U - %U\n",gline)
2911         fmt.Printf("Stoutt.Write -")
2912         os.Stdout.Write([]byte(gline))
2913         fmt.Printf("\n")
2914     }
2915     /*
2916     // should be cared in substitution ?
2917     if 0 < len(gline) && gline[0] == '|' {
2918         xgline, set, err := searchHistory(gshCtx,gline)
2919         if err {
2920             continue
2921         }
2922         if set {
2923             // set the line in command line editor
2924         }
2925         gline = xgline
2926     }
2927     */
2928     ghist := gshCtx.CmdCurrent
2929     ghist.WorkDir,_ = os.Getwd()
2930     ghist.WorkDirX = len(gshCtx.ChdirHistory)-1
2931     //fmt.Printf("--D--ChdirHistory(%d)\n",len(gshCtx.ChdirHistory))
2932     ghist.StartAt = time.Now()
2933     rusagev1 := Getrusagev()
2934     gshCtx.CmdCurrent.FoundFile = []string{}
2935     xgshCtx, fin := tgshelll(gshCtx,gline)
2936     rusagev2 := Getrusagev()
2937     ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
2938     gshCtx = xgshCtx
2939     ghist.EndAt = time.Now()
2940     ghist.CmdLine = gline
2941     ghist.FoundFile = gshCtx.CmdCurrent.FoundFile
2942
2943     /* record it but not show in list by default
2944     if len(gline) == 0 {
2945         continue
2946     }
2947     if gline == "hi" || gline == "history" { // don't record it
2948         continue
2949     }
2950     */
2951     gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist)
2952     if fin {
2953         break;
2954     }
2955     prevline = gline;
2956     hix++;
2957 }
2958 return gshCtx
2959 }
2960 func main() {
2961     argv := os.Args
2962     if 1 < len(argv) {
2963         if isin("version",argv){
2964             Version(nil,argv)
2965             return
2966         }
2967         comx := isinX("-c",argv)
2968         if 0 < comx {
2969             gshCtx,err := setupGshContext()
2970             if !err {
2971                 gshellv(gshCtx,argv[comx+1:])
2972             }
2973             return
2974         }
2975     }
2976     script(nil)
2977     //gshCtx := script(nil)
2978     //gshelll(gshCtx,"time")
2979 }
2980 //</pre></details>
2981 //<details id=todo open><summary>Consideration</summary></pre>
2982 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
2983 // - merged histories of multiple parallel gsh sessions
2984 // - alias as a function
2985 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
2986 // - retrieval PATH of files by its type
2987 // - gsh as an IME
2988 // - gsh a scheduler in precise time of within a millisecond
2989 // - all commands have its subcomand after "---" symbol
2990 // - filename expansion by "-find" command
2991 // - history of ext code and output of each commoand
2992 // - "script" output for each command by pty-tee or telnet-tee
2993 // - $BULLETIN command in PATH to show the priority
2994 // - "?" symbol in the command (not as in arguments) shows help request
2995 // - searching command with wild card like: which ssh-*
2996 // - longformat prompt after long idle time (should dismiss by BS)
2997 // - customizing by building plugin and dynamically linking it
2998 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
2999 // - "!" symbol should be used for negation, don't wast it just for job control

```



```

3000 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
3001 // - making canonical form of command at the start adding quotation or white spaces
3002 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
3003 // - name? or name! might be useful
3004 // - htar format - packing directory contents into a single html file using data scheme
3005 // - filepath substitution should be done by each command, especially in case of builtins
3006 // - @# substitution for the history of working directory, and @spec for more generic ones
3007 // - @did prefix to do the command at there, that means like (chdir @dir; command)
3008 // - GSH_PATH for plugins
3009 // - standard command output: list of data with name, size, resource usage, modified time
3010 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
3011 // - wc word-count, grep match line count, ...
3012 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
3013 // - -tailf-filename like tail -f filename, repeat close and open before read
3014 // - max. size and max. duration and timeout of (generated) data transfer
3015 //---END--- ("")/ITS more</pre></details>
3016 *
3017 <details id=references><summary>References</summary></pre>
3018 <p>
3019 <a href=https://golang.org>The Go Programming Language</a>
3020 <iframe width=100% height=300 src=https://golang.org></iframe>
3021
3022 <a href=https://developer.mozilla.org/ja/docs/Web>MDN web docs</a>
3023 <a href=https://developer.mozilla.org/ja/docs/Web/HTML/Element>HTML</a>
3024 CSS
3025 <a href=https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors>Selectors</a>
3026 <a href=https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat>repeat</a>
3027 HTTP
3028 JavaScript:
3029 .
3030 </p>
3031 </pre></details>
3032 <div id=gsh-footer>Pin.</div>
3033 <style>
3034 #gsh {border-width:1;margin:0;padding:0;}
3035 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
3036 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
3037 #gsh header{height:100px;}
3038 #gsh-footer{height:100px;background-size:50px;background-repeat:no-repeat;}
3039 #gsh note{color:#000;font-size:10pt;}
3040 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
3041 #gsh details{color:#888;background-color:#aaa;font-family:monospace;}
3042 #gsh summary{font-size:16pt;color:#24a;background-color:#eef;height:30px;}
3043 #gsh pre{font-size:11pt;color:#223;background-color:#fafff;}
3044 #gsh a{color:#24a;}
3045 #gsh a[name]{color:#24a;font-size:16pt;}
3046 @print {
3047 #gsh pre{font-size:11pt !import;}
3048 }
3049 </style>
3050 <!--
3051 // Logo image should be drawn by JavaScript from a meta-font.
3052 // CSS seems not follow line-splitted URL
3053 -->
3054 <script>
3055 GshLogo="data:image/png;base64,\
3056 iVBORw0KGgoAAAANSUHEUGAAQAQAAB/CAYAANDv3f4AAAAANSR0IARS4c6QAAAHhLWELm\
3057 UOAKGAAAQABAEEAAUAAABAAAAPGEBAAUAAAABAAAARgEoAAMAAAABAAAIAAIpAAQAAAAB\
3058 AATAAAGAAAAAABIAAAAAAQAQAEEAAABAQAQAQADAAAQAABAQAAAGAAEAAMAAAQAQAQGAWE\
3059 AAAAQAHAHAHAAAAAAYx1BhGAAAAaW5FLzAAALEAACMBAJGcGAAAF3RJRFEUAHtgnQuFNWZ\
3060 x++tTuk23iCgg0/jY6Osb8WgMzAvn7uG4+biSTR7YnQdQPCKGj2aNwL2MS1RkeUAhPnoCdu\
3061 4iUx7jri1Yz50D0GmF2VqIBEiSggCoIMMA+mu+vu//ZMD9ULdaU6a2UbuV91KGrq3vvd6/q\
3062 fnXvd8tB888IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
3063 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
3064 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
3065 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIIPDI4A8dLP2\
3066 2ExS9H+ftSK8dHx5ic2qgd7YusS+lqaalKfnY5YsokMhWEPtdk4MQFz5UeEX1bLYsaYU15\
3067 npDIKXZCPIRMS3JUaUq9ScqU6i+2kK3StuOnY5ReEKGJ7Qw7mOvKec2TqoOiZwo1jHFS\
3068 jboVHCstMRbSUXEJ8hFu7DsdmPb2+u4vWVFWXbPMeZUIAE/hcKoGAB6eKGLNykH56PC\
3069 Hx2iVV8KORkq3UeKiLydaOFONJ56OkdI6W5BwomOQlpYzi0N9DLmXpFK/60p2P/Ply0f\
3070 NmfmM+JNWNgnjW9KqOTOlvGSFt2p2Rillgn3iJ0Vk7YsoVMzEuVPFRKYdfoak2LR80\
3071 zrWocCOG6EhVgRaCj/dktj3g7dXHX4GKN6ARS0zYzergS6AraZDQqfK79SKTRRXHu+e9FM\
3072 L6Ga88pU/PN1pN1TLQJKS73dPXSr20u7iWPC8QhbnNcyHUI1LryOTQvYF5fvqBL7jX\
3073 +cNHjBj5gJrYdJHjY39o84D402QxT8HaPeFIOU+wlc+KnyhK5FGEVOWGAEhX8xMxLX\
3074 rikbd9tH8E52Vq14h89FUA6kJyYFbbQbnzLj4z4FiesndHCWvUoeeiVQob/5C9FY9D1Ue0\
3075 +zGhU9SngOqrm0uWgurkiR9pjBD4Y6uQqQd5TU0W63zD3Mhesy14V49isbdKyxh81CPFR\
3076 Uj6toACF7F9Vf58NB4DHTOMBae74Ent+eWrW+rLz/QT60AdB7QUjPs/OA7COonBNCeMU\
3077 tCu/coG28flvpKELTPPv8juRasEahhVxAR1guoeBPYfUdo4+OfEbdyB8L4tz9XesXFMOC\
3078 bgGv0glzG6W4F392xnHhdc+Mwf3JTjftz2yCIYJBJXNUT5KIKyck1exXRdl6BmcevN\
3079 AJovy/VbaCevqEhP46/ZlnJj9x17VL53Z15Mtvap1QGLNHW5pQDqKYNQ12z8n6cMG2ZV\
3080 qOoFJsdYvV0AZZdEayidvFJ35CS4jXZk9hir7e2z7m6p3T8LJpkyicJpV1HTk/DJFU4Jw1\
3081 LImhM5I9fzRkX4W/C+HQSPe+krbIyrN3qEPTNaHSALDs2xh5Q5NCoPPVDpEqcgm/8e\
3082 7zdoAhpTAg/mlKJ77UOVG0xybTDX/Ex/Ptfa/i7r7Ku+cSoiCUXwrohUXf16wE9Vh+ccVJ\
3083 pd/CFU42AK2IUP1VTK1L/sjYej5PVHqR728NrvfuzvDODGy9GoopuuHMLNcfTx48YHL2Qh\
3084 f/8h8XVuu/43rQg9xtq6Ytvcv1XDC3fmdQn9nbE2le7wKE1bOK65icBu0Eghd3tAw82wKpU\
3085 hrauc6ZcWdkcjZK8EUXMae71zUqWcu2nbi6Evn1J19/P7eW+ioMAogF+NI3iJLSf8dn91pA\
3086 WNN4rPy9jXuPcdU/HXzNzgtSvesLD2vWHW19m5rvvVZx9f0s4v/LfmqdiEPHdGfmf2\
3087 gJTy2wONPz2FEbic+zyNcNYJrNyhYGAo8RoJTAMrRiQcNfRw5FPtN+f+rTwd4S1Uv\
3088 bVLWbhfLrCF0AqazRD7176/zBJyLD5PBi25W14wQu7tkPBecOpuw+Kj0sp8GNMaZuwl\
3089 iOZuyDh9ZBr2xODrqVQPI5QXh6OwvJRAAAW46Pvt+RxAJVLjw7vY9/+CeUBMk168/rPQn\
3090 mCuFkZalDFN/yr18A5iwc3dkIKhsyvZuCYSVG/KHchwFWRDKAMMcD8EKX+rHF12A9bt2d172\
3091 2qN0vzCdyMfENy7QoGXDXWIKAIQI7coQzchyADWnerqN5VXtctjScdGp2OtmWJU7A+EH7\
3092 yhYtUgm1X7F7k1DwaRyUfN42FTIuXNDVETamL65Yc9R26VtbZaw2pX8nfmezh3EM+gso1k\
3093 d3/ZnBGELXPGUWzXgLYCq5Ew5/zBgy54AwogWfKfNwbqptceVtW4FUBvov32gew8DLzDTMaj\
3094 augp7t/bMXx+yw/egJGKOtksy2d+gFb9V0DvX5B1ZTOR+wFjy0P6UOXGYNGr/qta3vB\
3095 Fgeua6qv2d7v8n8Fdv3r1dBw34SGP9iODG9h5XWknh9KAAmmyJ6dkiPzZmtD3cnu77wtv5C\
3096 h/YrClg7Wxp/VvUrDuc+wsq54ymm+8zKOGYRSPRA4IKoGz1i8b6ytageEPmb9v/m09CUATZ\
3097 Jow6tVnPCmxBzj+sNNPnHsCjYja6c9RsrMyGkiW4I5UiouliLIRW7fmLLeX3z2+/GfW1LU2\
3098 Y72b6EAzkfyoPctJil5QlnJyLdrFrUZpl/3pmkug/yn9gaOgYMTf7neVixv/6CHUg1h1uh/\
3099 f9Uov+og70g3qz7zFL8x+qzW+/8FP6WfV7XSHinlayWdz2XULm/4uLmPwNoA5UgdcoL9\
3100 ZFA6cgoxzhTG6Q4NR5DojXuvily+rFbcuYjVsnLkV0CefphUbICLRmV1+9KP4vngHg6F2\
3101 NCGMSiCsnKfEde+mtflBwuXdmpBoZQ2/194225Y3TCzPQwhthG2zHraJo/yb0kdhpanZq\
3102 GXWf66/8Cb5AHcbzdpnhUjeG6YFowlgZemMtgNDeKzTiXVuc3Lk4YvTJepuq5tqSfKXda\
3103 uFu9mFWiG3snqntK76+3EXOQWwzVeqSpvrZmc2afYSyV461+O4KvyVgicCugG2rPp0yPtWeJ\
3104 otUm2UJWE0+0x60FtNxfW2U9x7o/bqZct5z2Poi0+vdpvDjcdxR34U9XCEhLroSktt3ug\
3105 AcwtK009F2Fn+qWdW6S0DcFodFAxncOFrXWUS0K93BZXXV7Ae+qwr506/204LXngLbrC\
3106 7HgRdrvEHz2LMYVvgqm5zTP5+7volRR/zJLO1lx+8ohOzEb+CV/OTU5ic3NGfjks30Mz\
3107 FtUtl+Yi4YfCwkJzqzY6h1GjEbWpGLYx009/j8k//WW3s32zQPHV5AMtPIIDFN20p6f\
3108 fz5YwF4HfMxdt+Buy4Nvu73yEFBOK65icot+zjP+8qf4JkyiTngKtb/gST0zMKACq18jJPL\
3109 A4PCyVnPMKOTjREv84HpyOsws/bsqyT2RGZ6zr10gA9sBhEp46hsP2ratmoJegrugBWD2Pw\
3110 NYD1B4OSTMbmcdS2E/GG2ZvrF7UejSqyW/7A7guEH6Kyy19q3fp0QvGxtx4dz+Ueg+Lmy5V\
3111 jlyjtO+b5LSqpp5Nz6nbwFfHudaYgemZy4ap1z5dlbByA3NQTc4F3RKYfOTKAUF9XrYl0Wu8\
3112 sDMC/H290v0GTQV1C+izhTu27rgAebkb4+8H3P553q0Oyu/WHj2lZwbd7z2XLuv4f1gmQSV\
3113 2GML+6MkhorvaWqne1lyZ/gLLX+IBnCN2FQ7F9Y5XQFN/qua+Hr3UrAg9LMTLrG3bfPyEtP\
3114 6ed5oyCzcXnX9Q2JAggbYmXSL9vZqSgBfXUBjHpbXbzM+vKueBRRIotE/Bw8ogf/LIhZY\
3115 /9Tcnsb681t7DgnQRE1EVT2z9w5S5Jf71F5z0vLYfTlVqUT0B62etccbR01LHeS68SYeT\
3116 202udegemW787ng7dKrv19rLzt0MPBK73na4Yrdzfm+5DzsymDymaHnLokvPOVH5FrsQ\
3117 wC76RwU9Dkx5MUI9vOXMAx+ePguLw8/dvg6U1LPvPsPbXspOniQwagElsm9gNxtc0E0lvj5\
3118 7tBBBjAdhKMPD0/q/irWbf44t5CNKQWaq7DsuJzh16CLz8bk+1u2u78FXyWkLIQ4/qY2x\
3119 tYvjX8boyW6zwc9/Ojwz7pUtVlLp0NQZuLo8PKODMuluovotoJLyxcrNWHHEjQw5YKrKPs\
3120 ZJh14LpLcIQXOyp6nMs5fyKei1e0G95+WXEj3m5mcmjNe5b+lyHZYELXgJrDmYJ/HTMK0S\
3121 aP67Md34PueYw28WdovSjzXVF/xsFe+Lpz/wj0Q9ieH94ZwqVs62+CUhV31MtnjShfXorHf\
3122 WkZg9FwIrtCRJWh5+/ocSLzQlzeG2BvTtG+wOpqXRYEwcaRfRdbSgC5bd/PyScBHakPWO\
3123 qZx9y4L1L0uABBA4k5we8QdSO66+bn0WjzFXyUvYi6Ece001I7SAZkxOqtmzB9RcaVvxx\
3124 2CMBjAdTcruWwyxriw4myTH92t3R93/8XlJ0ESWetyy7qPTj1odwkAmhFEA2K6D1WNe6h\

```

```

3125 H52HuWwIaLQHQQUZwr6yznTls7rgu40YBJq4JBWJGayRhTyeYx4X8/xCw+rUS9L5yc50A+W\
3126 8v0w0NZZAxw7VADPZcEDpXpdsLx0DkefRwEM+yj47aEaa7yxzjXm+61FzUL46ch7cOd6Q/m\
3127 Wncf9BTVwXbs6Z3hhxPvIvmlkjhJubTFKRBbag1QCWiwbuilPFyKlHhWzaq8YKoeMcj191y9Ly\
3128 PwK79U/55Bk75FSMchwhj79Y35xY7qu8YspvTbqSG+55hdjJn6Y56ErFyqVOLZxoeLzbnwWj\
3129 YwkG5S2p1IOK5d1jzgs+2LB1B4Z6/Ag+uoSa6yuWOY1jzcCuOG4llqXVQ0Fep1wulxUL4pFR\
3130 zD3GL6w1VE4JA35xeKlNLSuBh/34RcwB6JXGgZ6rFlBBjbbjH7t1WbGDRVd4bieXgpPbhW\
3131 NQT3IqHHz7MTHVuXxv45r8FpQWRndiqVzq8LxEF16+rqDLV82CTnVYBIdBe2JfBpwMJF\
3132 aW3rXYbq9qXhLnmCjCvUW55fKMRc2LbzJbK8mU55cn4x/2rLdJQzNjtKkyuu01pdqccfMz\
3133 gKOp/ahXooVi-JfofiMzUjYn8P7QHmhMxkAaUeTX6c7F07sUUKgyG50z33vV/Z0C7b+scH\
3134 Ltnp1tH3YeW841pGt4JWAnu7Pn5xwqjxB4IMabBe3Q8rFLzPCJfTe09SF0b8NAdzSFwqYfHBU\
3135 nmljJITHChN3eSRt+42Mk5KwETsxPMe35RJTvorP3mm49VMOGfP8oId191X61dvbXmkqjvB\
3136 nfydY9m8wimzMLKZeSL/VzQSkDpZcdYcte7lq/B4XkfkQaek3mL47r29fQL/gaTt/vzEOM\
3137 gDTT0U9UWbKXUVMfH9MYULzJvPxxu0fPO0/pTtedhod/1XXxGZawFuXp6eG1Lz+eme2X91bo\
3138 oXuU119fObLAKGqGhafa5NVPhxjK7X0GuL0MRm+JAFefsnnaKzLRhZxLYbF5ed1UwKc1/wD7\
3139 fd+JL72vETDPE1ggWkZj6zEP/d5durt+ZhihxkLnhs7umT011AjkyVScenpJ1W1AAcZAE\
3140 dgY2S/S+nLNOdPglXVd/SkUr+JL5/9VsbL75z+byNS8Q2EuQn/Oa3x1/FJZS/Vz30EGCBq\
3141 ePtCYCRORCKR3q6vL0pof7XEXvDaaVzcjQECZX56CYcmxZ/7CyUwar2IINzXk4NOC075/\
3142 yfWRk3XuwYfGjgmxT/xdbpt8uSR17F11lUoFJtQm3U17cKXfygMVsFdvwpVqRPaeh07FRv\
3143 hUL469jpwu1lyy+FX0C+Cy0v+IWxZylh/w3n7fiibretTevURMitjpkWRVtmPKZmHdzFciM\
3144 dMf1f6+em10/651MmCDD2YFE12dFycj38aRabQSPGX1sCGUCaKRD0UySzaUvgZk6zAVTfE\
3145 LLGf1XPfjyIthCkphr+cN+r76LoL1d3d451+andv9Yr4veCWg9+srXtX6G/areZLXB4WX\
3146 tgy7Wk4n+Z8f/FfzURiA3ky5ULmo9CE8N3HgLinI5sRnY32hsXoRnTBMvWmiP9zT7o3\
3147 j0g8vnn35aeeGY1gCmlw2/fvicj0JytieoL0XvRGhMyNz1/IJtL6w3j5y8j+711dy057\
3148 xLjDm+X0F0gtrugEUTDvIpfCenwAE2KAeVArG5T3tjBCQT+5rCU+U1BzxP1PJumpRVP\
3149 4YtEz9wP9xlFv/oppuyDp9Nfyih91/XNXovNSd5dGG8C8wms31CzfrkCQUTCZSHj+vm8q\
3150 JY7XE3M6WqjLSr6LVB668ToExtHjJ/4Cdw24+uzFvsJrsT11RkFoOoAltznPzdf2D12QrO\
3151 8V5898DpsboVhRLDp6exvrEO3y94q9DQPKC5ZmjJyz021LdV7yb3zfl8qmsDmOPARTVWF3i\
3152 N1MQWk1jEavqOmz78D2ZvefMhdFCU86n6FBB5:KF1FPMRHE6Fo0S0atvM/d8V98km7D\
3153 C58YrseFuLvsplpXb79z64erdZnyNLKileJdalUak7j0orr315x+YA9CbQDF/cK7JHdDb\
3154 E5sg690KM9pdRld6v3vgEYvbdQcuc1VM9no/QaPP3KZ1ve8zWcmJjK30Kx+30RKQ8K1wN\
3155 blxaFhe29qLBo8fGKam6n5P9mdGP5bmtIkpmc22r7BHskjJP0kmCktCF/KAM1e0JXteJk\
3156 v7q+/0zmZbn/5iOHT3+NPgzn2eyx7uiZ0JDM9xoyTcZBTOya+vndqg3URPjYxmbDoe1/au\
3157 zq4BeygslmpghcdLIXcmLwXskzBGwa940stveB+sf714IiK3oiO5Mkod+r9/I2Vx80P9Ec3\
3158 xp7QYU8JGTfemcatO+NeY/99v3xhb+21bh03cnol1jdFCZnzkeapsDN/vjdg4XP4Cnb8+W9p\
3159 9zaduKz2Q3tevo5lytqto30pzK9Ee5shOY+FXEVL50r6xr5HkDFMGaAdKQ3yA09pydfdj\
3160 pp5k+jNq6grNj3DfyKI5h14oOKj1aZehBJ9NtWfBACv1uIawS2xVTahfB50dfpseEaP\
3161 mR1FLXOm8XmXnp/fBy6avg2fyt5Skwno2mMPEFS3gCf3o4UGGSj/wI548wVlfbVvab720b\
3162 Xx/MwzklF9zXPQMhX5C1afjhiYjhsR7BkMfG8mLT+D3cdJF2qod1vnn3V3d60xw7hyf\
3163 koSVf0pEpkzPegJWQtd70c6dnp1H7zi0z933hOLHWYJulREhZ7ptxeV69XWH+3Jdasm6tO\
3164 iEWSY1C5J8EajZNR0adga71evOR2LBSccVC820u5Ue1JbspxVgHEcusjRkKYLW0VSSUInTmW\
3165 LaycFXhpSwTKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3166 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3167 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3168 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3169 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3170 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
3171 n38w0nAAAAASUVRK5CYII=";
3172 document.getElementById('banner').style.backgroundImage="url("+GshLogo+")";
3173 document.getElementById('gsh-footer').style.backgroundImage="url("+QR-ITS-more.jp.png+")";
3174 //https://www.w3schools.com/JSREF/prop_style_backgroundposition.asp
3175 var bannerStop = false
3176 function shiftBG(){
3177   bannerStop = !bannerStop
3178   document.getElementById('banner').style.backgroundPosition = "0 0";
3179 }
3180 //https://www.w3schools.com/jsref/met_win_setinterval.asp
3181 function shiftBanner(){
3182   var now = new Date().getTime();
3183   //console.log("now="+now%10)
3184   if( !bannerStop ){
3185     document.getElementById('banner').style.backgroundPosition = ((now/10)%100000)+" 0";
3186   }
3187 }
3188 setInterval(shiftBanner,10);
3189 </script>
3190 -->
3191 *//</span></html>
3192

```