

```

1  /*<html>
2  <span id="gsh">
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-0.1.9 by SatoxITS</title>
7  <header id="banner" height="100px" onclick="shiftBG();" style="">
8  <div align="right"><note>GShell version 0.1.9 // 2020-08-23 // SatoxITS</note></div>
9  </header>
10 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
11 <p>
12 <note>
13 It is a shell for myself, by myself, of myself. --SatoxITS(^-^ )
14 </note>
15 </p>
16 <span id="gsh-menu">
17 | <span onclick="html_new();">NewWindow</span>
18 | <span onclick="html_open();">Unfold</span>
19 | <span onclick="html_fold();">Fold</span>
20 | <span onclick="html_stop();">Stop</span>
21 | <span onclick="html_close();">Close</span>
22 |</span>
23 */
24 /*
25 <details id="html-src" onclick="frame_open();"><summary>Total Source of GShell</summary><div>
26 <h2>The full of this HTML including the Go code is here.</h2>
27 <span id="src-frame"></span> // a window to show source code
28 </div></details>
29 */
30 /*
31 <details id="overview"><summary>Overview</summary><div class="gsh-src">
32 To be written
33 </div>
34 </details>
35 */
36 /*
37 <details id="index">
38 <summary>Go Source Code Index</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
39 Implementation
40 Structures
41 <a href="#import">import</a>
42 <a href="#struct">struct</a>
43 Main functions
44 <a href="#comexpansion">str-expansion</a> // macro processor
45 <a href="#finder">finder</a> // builtin find + du
46 <a href="#grep">grep</a> // builtin grep + wc + cksum + ...
47 <a href="#plugin">plugin</a> // plugin commands
48 <a href="#ex-commands">system</a> // external commands
49 <a href="#builtin">builtin</a> // builtin commands
50 <a href="#network">network</a> // socket handler
51 <a href="#remote-sh">remote-sh</a> // remote shell
52 <a href="#redirect">redirect</a> // StdIn/Out redireciton
53 <a href="#history">history</a> // command history
54 <a href="#rusage">rusage</a> // resouce usage
55 <a href="#encode">encode</a> // encode / decode
56 <a href="#IME">IME</a> // command line IME
57 <a href="#getline">getline</a> // line editor
58 <a href="#scanf">scanf</a> // string decomposer
59 <a href="#interpreter">interpreter</a> // command interpreter
60 <a href="#main">main</a>
61 </div>
62 </details>
63 */
64 <details id="gsh-gocode">
65 <summary>Go Source Code</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
66 // gsh - Go lang based Shell
67 // (c) 2020 ITS more Co., Ltd.
68 // 2020-0807 created by SatoxITS (sato@its-more.jp)
69
70 package main // gsh main
71 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
72 import (
73 "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
74 "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
75 "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
76 "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
77 "time" // <a href="https://golang.org/pkg/time/">time</a>
78 "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
79 "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
80 "os" // <a href="https://golang.org/pkg/os/">os</a>
81 "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
82 "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
83 "net" // <a href="https://golang.org/pkg/net/">net</a>
84 "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
85 // "html" // <a href="https://golang.org/pkg/html/">html</a>
86 "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
87 "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
88 "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
89 "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
90 "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
91 // "gshdata" // gshell's logo and source code
92 )
93
94 var NAME = "gsh"
95 var VERSION = "0.1.9"
96 var DATE = "2020-0823"
97 var LINE_SIZE = (8*1024)
98 var PATHSEP = ":" // should be ";" in Windows
99 var DIRSEP = "/" // canbe \ in Windows
100 var GSH_HOME = ".gsh" // under home directory
101 var MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
102 var PROMPT = ">"
103
104 // -xX logging control
105 // --A-- all
106 // --I-- info.
107 // --D-- debug
108 // --T-- time and resource usage
109 // --W-- warning
110 // --E-- error
111 // --F-- fatal error
112 // --Xn- network
113
114 // <a name="struct">Structures</a>
115 type GCommandHistory struct {
116 StartAt time.Time // command line execution started at
117 EndAt time.Time // command line execution ended at
118 ResCode int // exit code of (external command)
119 CmdError error // error string
120 OutData *os.File // output of the command
121 FoundFile []string // output - result of ufind
122 Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
123 CmdId int // maybe with identified with arguments or impact
124 // redirecton commands should not be the CmdId

```

```

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

```

```

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

```

```

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

```

```

500         addnl = true
501     }
502     fmt.Fprintf(os.Stderr, "--D-- write %d [%d.%d] %d %d/%d/%d\n",
503         toi, ni, oi, off, olen, remlen, ilen)
504     toi += 1
505     os.Stdout.Write([]byte(line[0:olen]))
506     if addnl {
507         //os.Stdout.Write([]byte("\r\n"))
508         os.Stdout.Write([]byte("\n"))
509         os.Stdout.Write([]byte("\n"))
510     }
511     line = line[olen:]
512     off += olen
513     remlen -= olen
514 }
515 }
516 fmt.Fprintf(os.Stderr, "--I-- lnspl %d to %d\n", ni, toi)
517 return gshCtx
518 }
519
520 // <a name="grep">grep</a>
521 // "lines", "lin" or "lmp" for "(text) line processor" or "scanner"
522 // a*,lab,c,... sequential combination of patterns
523 // what "LINE" is should be definable
524 // generic line-by-line processing
525 // grep [-v]
526 // cat -n -v
527 // uniq [-c]
528 // tail -f
529 // sed s/x/y/ or awk
530 // grep with line count like wc
531 // rewrite contents if specified
532 func (gsh*GshContext)xGrep(path string, rexp[[]string](int){
533     file, err := os.OpenFile(path, os.O_RDONLY, 0)
534     if err != nil {
535         fmt.Printf("--E-- grep %v (%v)\n", path, err)
536         return -1
537     }
538     defer file.Close()
539     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n", path, rexp) }
540     //reader := bufio.NewReaderSize(file, LINESIZE)
541     reader := bufio.NewReaderSize(file, 80)
542     li := 0
543     found := 0
544     for li = 0; ; li++ {
545         line, err := reader.ReadString('\n')
546         if len(line) <= 0 {
547             break
548         }
549         if 150 < len(line) {
550             // maybe binary
551             break;
552         }
553         if err != nil {
554             break
555         }
556         if 0 <= strings.Index(string(line), rexp[0]) {
557             found += 1
558             fmt.Printf("%s:%d: %s", path, li, line)
559         }
560     }
561     //fmt.Printf("total %d lines %s\n", li, path)
562     //if( 0 < found ){ fmt.Printf("((found %d lines %s))\n", found, path); }
563     return found
564 }
565
566 // <a name="finder">Finder</a>
567 // finding files with it name and contents
568 // file names are Ored
569 // show the content with %x fmt list
570 // ls -R
571 // tar command by adding output
572 type fileSum struct {
573     Err int64 // access error or so
574     Size int64 // content size
575     DupSize int64 // content size from hard links
576     Blocks int64 // number of blocks (of 512 bytes)
577     DupBlocks int64 // Blocks pointed from hard links
578     HLinks int64 // hard links
579     Words int64
580     Lines int64
581     Files int64
582     Dirs int64 // the num. of directories
583     SymLink int64
584     Flats int64 // the num. of flat files
585     MaxDepth int64
586     MaxNamlen int64 // max. name length
587     nextRepo time.Time
588 }
589 func showFusage(dir string, fusage *fileSum){
590     bsum := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
591     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
592
593     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
594         dir,
595         fusage.Files,
596         fusage.Dirs,
597         fusage.SymLink,
598         fusage.HLinks,
599         float64(fusage.Size)/1000000.0, bsum);
600 }
601 const (
602     S_IFMT = 0170000
603     S_IFCHR = 0020000
604     S_IFDIR = 0040000
605     S_IFREG = 0100000
606     S_IFLNK = 0120000
607     S_IFSOCK = 0140000
608 )
609 func cumPinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[[]string, verb bool)(*fileSum){
610     now := time.Now()
611     if time.Second <= now.Sub(fsum.nextRepo) {
612         if !fsum.nextRepo.IsZero(){
613             tstamp := now.Format(time.Stamp)
614             showFusage(tstamp, fsum)
615         }
616         fsum.nextRepo = now.Add(time.Second)
617     }
618     if staterr != nil {
619         fsum.Err += 1
620         return fsum
621     }
622     fsum.Files += 1
623     if 1 <= fstat.Nlink {
624         // must count only once...

```

```

625 // at least ignore ones in the same directory
626 //if finfo.Mode().IsRegular() {
627 if (fstat.Mode & S_IFMT) == S_IFREG {
628     fsum.HLinks += 1
629     fsum.DupBlocks += int64(fstat.Blocks)
630     //fmt.Printf("---Dup HardLink %v %s\n", fstat.Nlink, path)
631 }
632 }
633 //fsum.Size += finfo.Size()
634 fsum.Size += fstat.Size
635 fsum.Blocks += int64(fstat.Blocks)
636 //if verb { fmt.Printf("(%dBk) %s", fstat.Blocks/2, path) }
637 if isin("-ls", argv){
638     //if verb { fmt.Printf("%4d %8d ", fstat.Blksize, fstat.Blocks) }
639 //     fmt.Printf("%d\t", fstat.Blocks/2)
640 }
641 //if finfo.IsDir()
642 if (fstat.Mode & S_IFMT) == S_IFDIR {
643     fsum.Dirs += 1
644 }
645 //if (finfo.Mode() & os.ModeSymlink) != 0
646 if (fstat.Mode & S_IFMT) == S_IFLNK {
647     //if verb { fmt.Printf("symlink(%v,%s)\n", fstat.Mode, finfo.Name()) }
648     //{ fmt.Printf("symlink(%o,%s)\n", fstat.Mode, finfo.Name()) }
649     fsum.SymLink += 1
650 }
651 return fsum
652 }
653 func (gsh*GshContext)xxFindEntv(depth int, total *fileSum, dir string, dstat syscall.Stat_t, ei int, entv []string, npatv []string, argv []string)(*fileSum){
654     nols := isin("-grep", argv)
655     // sort entv
656     /*
657     if isin("-t", argv){
658         sort.Slice(filev, func(i, j int) bool {
659             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
660         })
661     }
662     */
663     /*
664     if isin("-u", argv){
665         sort.Slice(filev, func(i, j int) bool {
666             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
667         })
668     }
669     if isin("-U", argv){
670         sort.Slice(filev, func(i, j int) bool {
671             return 0 < filev[i].CreatTime().Sub(filev[j].CreatTime())
672         })
673     }
674     */
675     /*
676     if isin("-S", argv){
677         sort.Slice(filev, func(i, j int) bool {
678             return filev[j].Size() < filev[i].Size()
679         })
680     }
681     */
682     for _, filename := range entv {
683         for _, npat := range npatv {
684             match := true
685             if npat == "*" {
686                 match = true
687             }else{
688                 match, _ = filepath.Match(npat, filename)
689             }
690             path := dir + DIRSEP + filename
691             if !match {
692                 continue
693             }
694             var fstat syscall.Stat_t
695             staterr := syscall.Lstat(path, &fstat)
696             if staterr != nil {
697                 if !isin("-w", argv){fmt.Printf("ufind: %v\n", staterr) }
698                 continue;
699             }
700             if isin("-du", argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
701                 // should not show size of directory in "-du" mode ...
702             }else
703             if !nols && !isin("-s", argv) && (!isin("-du", argv) || isin("-a", argv)) {
704                 if isin("-du", argv) {
705                     fmt.Printf("%d\t", fstat.Blocks/2)
706                 }
707                 showFileInfo(path, argv)
708             }
709             if true { // && isin("-du", argv)
710                 total = cumFinfo(total, path, staterr, fstat, argv, false)
711             }
712             /*
713             if isin("-wc", argv) {
714                 */
715             /*
716             x := isinX("-grep", argv); // -grep will be convenient like -ls
717             if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls
718                 if IsRegFile(path){
719                     found := gsh.xGrep(path, argv[x+1:])
720                     if 0 < found {
721                         foundv := gsh.CmdCurrent.FoundFile
722                         if len(foundv) < 10 {
723                             gsh.CmdCurrent.FoundFile =
724                                 append(gsh.CmdCurrent.FoundFile, path)
725                         }
726                     }
727                 }
728             }
729             if !isin("-r0", argv) { // -d 0 in du, -depth n in find
730                 //total.Depth += 1
731                 if (fstat.Mode & S_IFMT) == S_IFLNK {
732                     continue
733                 }
734                 if dstat.Rdev != fstat.Rdev {
735                     fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
736                         dir, dstat.Rdev, path, fstat.Rdev)
737                 }
738                 if (fstat.Mode & S_IFMT) == S_IFDIR {
739                     total = gsh.xxFind(depth+1, total, path, npatv, argv)
740                 }
741             }
742         }
743     }
744     return total
745 }
746 func (gsh*GshContext)xxFind(depth int, total *fileSum, dir string, npatv []string, argv []string)(*fileSum){
747     nols := isin("-grep", argv)
748     dirfile, oerr := os.OpenFile(dir, os.O_RDONLY, 0)
749     if oerr == nil {

```

```

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

```

```

875 }
876 func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
877     var fullpath string
878     if strBegins(name,DIRSEP){
879         fullpath = name
880     }else{
881         fullpath = dir + DIRSEP + name
882     }
883     fi, err := os.Stat(fullpath)
884     if err != nil {
885         fullpath = dir + DIRSEP + name + ".go"
886         fi, err = os.Stat(fullpath)
887     }
888     if err == nil {
889         fm := fi.Mode()
890         if fm.IsRegular() {
891             // R_OK=4, W_OK=2, X_OK=1, F_OK=0
892             if syscall.Access(fullpath,5) == nil {
893                 ffullpath = fullpath
894                 ffound = true
895                 if !isin("-s", argv) {
896                     showFileInfo(fullpath,argv)
897                 }
898             }
899         }
900     }
901     return ffullpath, ffound
902 }
903 func which(list string, argv []string) (fullpathv []string, itis bool){
904     if len(argv) <= 1 {
905         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
906         return []string{"", false
907     }
908     path := argv[1]
909     if strBegins(path,"/") {
910         // should check if excecutable?
911         exOK := showIfExecutable(path,"/",argv)
912         fmt.Printf("--D-- %v exOK=%v\n",path,exOK)
913         return []string{path},exOK
914     }
915     pathenv, efound := os.LookupEnv(list)
916     if !efound {
917         fmt.Printf("--E-- which: no \"%s\" environment\n",list)
918         return []string{"", false
919     }
920     showall := isin("-a",argv) || 0 <= strings.Index(path,"*")
921     dirv := strings.Split(pathenv,PATHSEP)
922     ffound := false
923     ffullpath := path
924     for _, dir := range dirv {
925         if 0 <= strings.Index(path,"*") { // by wild-card
926             list,_ := ioutil.ReadDir(dir)
927             ffullpath, ffound = showMatchFile(list,path,dir,argv)
928         }else{
929             ffullpath, ffound = showIfExecutable(path,dir,argv)
930         }
931         //if ffound && !isin("-a", argv) {
932         if ffound && !showall {
933             break;
934         }
935     }
936     return []string{ffullpath}, ffound
937 }
938
939 func stripLeadingWSParg(argv[]string)([]string){
940     for i, 0 < len(argv); {
941         if len(argv[0]) == 0 {
942             argv = argv[1:]
943         }else{
944             break
945         }
946     }
947     return argv
948 }
949 func xEval(argv []string, nlend bool){
950     argv = stripLeadingWSParg(argv)
951     if len(argv) == 0 {
952         fmt.Printf("eval [%%format] [Go-expression]\n")
953         return
954     }
955     pfmt := "%v"
956     if argv[0][0] == '$' {
957         pfmt = argv[0]
958         argv = argv[1:]
959     }
960     if len(argv) == 0 {
961         return
962     }
963     gocode := strings.Join(argv, " ");
964     //fmt.Printf("eval [%v] [%v]\n",pfmt,gocode)
965     fset := token.NewFileSet()
966     rval, _ := types.Eval(fset,nil,token.NoPos,gocode)
967     fmt.Printf(pfmt,rval.Value)
968     if nlend { fmt.Printf("\n") }
969 }
970
971 func getval(name string) (found bool, val int) {
972     /* should expand the name here */
973     if name == "gsh.pid" {
974         return true, os.Getpid()
975     }else
976     if name == "gsh.ppid" {
977         return true, os.Getppid()
978     }
979     return false, 0
980 }
981
982 func echo(argv []string, nlend bool){
983     for ai := 1; ai < len(argv); ai++ {
984         if 1 < ai {
985             fmt.Printf(" ");
986         }
987         arg := argv[ai]
988         found, val := getval(arg)
989         if found {
990             fmt.Printf("%d",val)
991         }else{
992             fmt.Printf("%s",arg)
993         }
994     }
995     if nlend {
996         fmt.Printf("\n");
997     }
998 }
999

```



```

1000 func resfile() string {
1001     return "gsh.tmp"
1002 }
1003 //var resF *File
1004 func resmap() {
1005     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1006     // https://devepaper.com/solution-to-golang-bad-file-descriptor-problem/
1007     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1008     if err != nil {
1009         fmt.Printf("refF could not open: %s\n",err)
1010     }else{
1011         fmt.Printf("refF opened\n")
1012     }
1013 }
1014
1015 // @@2020-0821
1016 func gshScanArg(str string,strip int)(argv []string){
1017     var si = 0
1018     var sb = 0
1019     var inBracket = 0
1020     var arg1 = make([]byte,LINESIZE)
1021     var ax = 0
1022     debug := false
1023
1024     for ; si < len(str); si++ {
1025         if str[si] != ' ' {
1026             break
1027         }
1028     }
1029     sb = si
1030     for ; si < len(str); si++ {
1031         if sb <= si {
1032             if debug {
1033                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1034                     inBracket,sb,si,arg1[0:ax],str[si:])
1035             }
1036         }
1037         ch := str[si]
1038         if ch == '{' {
1039             inBracket += 1
1040             if 0 < strip && inBracket <= strip {
1041                 //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1042                 continue
1043             }
1044         }
1045         if 0 < inBracket {
1046             if ch == '}' {
1047                 inBracket -= 1
1048                 if 0 < strip && inBracket < strip {
1049                     //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1050                     continue
1051                 }
1052             }
1053             arg1[ax] = ch
1054             ax += 1
1055             continue
1056         }
1057         if str[si] == ' ' {
1058             argv = append(argv,string(arg1[0:ax]))
1059             if debug {
1060                 fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1061                     -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1062             }
1063             sb = si+1
1064             ax = 0
1065             continue
1066         }
1067         arg1[ax] = ch
1068         ax += 1
1069     }
1070     if sb < si {
1071         argv = append(argv,string(arg1[0:ax]))
1072         if debug {
1073             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1074                 -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1075         }
1076     }
1077     if debug {
1078         fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,si,len(argv),argv)
1079     }
1080     return argv
1081 }
1082
1083 // should get stderr (into tmpfile ?) and return
1084 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1085     var pv = []int{-1,-1}
1086     syscall.Pipe(pv)
1087
1088     xarg := gshScanArg(name,1)
1089     name = strings.Join(xarg," ")
1090
1091     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
1092     pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
1093     fdix := 0
1094     dir := "?"
1095     if mode == "r" {
1096         dir = "<"
1097         fdix = 1 // read from the stdout of the process
1098     }else{
1099         dir = ">"
1100         fdix = 0 // write to the stdin of the process
1101     }
1102     gshPA := gsh.gshPA
1103     savfd := gshPA.Files[fdix]
1104
1105     var fd uintptr = 0
1106     if mode == "r" {
1107         fd = pout.Fd()
1108         gshPA.Files[fdix] = pout.Fd()
1109     }else{
1110         fd = pin.Fd()
1111         gshPA.Files[fdix] = pin.Fd()
1112     }
1113     fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1114     // should do this by Goroutine?
1115     gsh.BackGround = true
1116     gshell1(*gsh,name)
1117     gsh.BackGround = false
1118
1119     gshPA.Files[fdix] = savfd
1120     return pin,pout,false
1121 }
1122
1123 // <a name="ex-commands">External commands</a>
1124 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {

```

```

1125 if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1126
1127 gshPA := gsh.gshPA
1128 fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1129 if itis == false {
1130     return true,false
1131 }
1132 fullpath := fullpathv[0]
1133 argv = unescapeWhiteSPV(argv)
1134 if 0 < strings.Index(fullpath, ".go") {
1135     nargv := argv // []string{}
1136     gofullpathv, itis := which("PATH",[]string{"which", "go", "-s"})
1137     if itis == false {
1138         fmt.Printf("--F-- Go not found\n")
1139         return false,true
1140     }
1141     gofullpath := gofullpathv[0]
1142     nargv = []string{ gofullpath, "run", fullpath }
1143     fmt.Printf("--I-- %s (%s %s %s)\n",gofullpath,
1144         nargv[0],nargv[1],nargv[2])
1145     if exec {
1146         syscall.Exec(gofullpath,nargv,os.Environ())
1147     }else{
1148         pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1149         if gsh.BackGround {
1150             fmt.Printf("--Ip- in Background pid[%d]\n",pid)
1151             gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1152         }else{
1153             rusage := syscall.Rusage {}
1154             syscall.Wait4(pid,nil,0,&rusage)
1155             gsh.LastRusage = rusage
1156             gsh.CmdCurrent.Rusagev[1] = rusage
1157         }
1158     }
1159 }else{
1160     if exec {
1161         syscall.Exec(fullpath,argv,os.Environ())
1162     }else{
1163         pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1164         //fmt.Printf("[%d]\n",pid); // '&' to be background
1165         if gsh.BackGround {
1166             fmt.Printf("--Ip- in Background pid[%d]\n",pid)
1167             gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1168         }else{
1169             rusage := syscall.Rusage {}
1170             syscall.Wait4(pid,nil,0,&rusage);
1171             gsh.LastRusage = rusage
1172             gsh.CmdCurrent.Rusagev[1] = rusage
1173         }
1174     }
1175 }
1176 return false,false
1177 }
1178
1179 // <a name="builtin">Builtin Commands</a>
1180 func sleep(gshCtx GshContext, argv []string) {
1181     if len(argv) < 2 {
1182         fmt.Printf("Sleep 100ms, 100us, 100ns, ...)\n")
1183         return
1184     }
1185     duration := argv[1];
1186     d, err := time.ParseDuration(duration)
1187     if err != nil {
1188         d, err = time.ParseDuration(duration+"s")
1189         if err != nil {
1190             fmt.Printf("duration ? %s (%s)\n",duration,err)
1191             return
1192         }
1193     }
1194     //fmt.Printf("Sleep %v\n",duration)
1195     time.Sleep(d)
1196     if 0 < len(argv[2:]) {
1197         gshellv(gshCtx, argv[2:])
1198     }
1199 }
1200 func repeat(gshCtx GshContext, argv []string) {
1201     if len(argv) < 2 {
1202         return
1203     }
1204     start0 := time.Now()
1205     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1206         if 0 < len(argv[2:]) {
1207             //start := time.Now()
1208             gshellv(gshCtx, argv[2:])
1209             end := time.Now()
1210             elps := end.Sub(start0);
1211             if( 1000000000 < elps ){
1212                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1213             }
1214         }
1215     }
1216 }
1217
1218 func gen(gshCtx GshContext, argv []string) {
1219     gshPA := gshCtx.gshPA
1220     if len(argv) < 2 {
1221         fmt.Printf("Usage: %s N\n",argv[0])
1222         return
1223     }
1224     // should br repeated by "repeat" command
1225     count, _ := strconv.Atoi(argv[1])
1226     fd := gshPA.Files[1] // Stdout
1227     file := os.NewFile(fd,"internalStdOut")
1228     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1229     //buf := []byte{}
1230     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1231     for gi := 0; gi < count; gi++ {
1232         file.WriteString(outdata)
1233     }
1234     //file.WriteString("\n")
1235     fmt.Printf("\n(%d B)\n",count*len(outdata));
1236     //file.Close()
1237 }
1238
1239 // <a name="rexec">Remote Execution</a> // 2020-0820
1240 func Elapsed(from time.Time)(string){
1241     elps := time.Now().Sub(from)
1242     if 1000000000 < elps {
1243         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1244     }else
1245     if 1000000 < elps {
1246         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1247     }else{
1248         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1249     }
}

```

```

1250 }
1251 func absSize(size int64)(string){
1252     fsize := float64(size)
1253     if 1024*1024*1024 < size {
1254         return fmt.Sprintf("%8.2fGiB", fsize/(1024*1024*1024))
1255     }else
1256     if 1024*1024 < size {
1257         return fmt.Sprintf("%8.3fMiB", fsize/(1024*1024))
1258     }else{
1259         return fmt.Sprintf("%8.3fKiB", fsize/1024)
1260     }
1261 }
1262 func absSpeed(totalB int64, ns time.Duration)(string){
1263     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1264     if 1000 < MBs {
1265         return fmt.Sprintf("%6.3fGBps", MBs/1000)
1266     }
1267     if 1 <= MBs {
1268         return fmt.Sprintf("%6.3fMBps", MBs)
1269     }else{
1270         return fmt.Sprintf("%6.3fKBps", MBs*1000)
1271     }
1272 }
1273 func fileRelay(what string, in*os.File, out*os.File, size int64, bsiz int)(wcount int64){
1274     Start := time.Now()
1275     buff := make([]byte, bsiz)
1276     var total int64 = 0
1277     var rem int64 = size
1278     nio := 0
1279     Prev := time.Now()
1280     var PrevSize int64 = 0
1281
1282     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1283         what, absSize(total), size, nio)
1284
1285     for i:= 0; ; i++ {
1286         var len = bsiz
1287         if int(rem) < len {
1288             len = int(rem)
1289         }
1290         Now := time.Now()
1291         Elps := Now.Sub(Prev);
1292         if 1000000000 < Now.Sub(Prev) {
1293             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1294                 what, absSize(total), size, nio,
1295                 absSpeed((total-PrevSize), Elps))
1296             Prev = Now;
1297             PrevSize = total
1298         }
1299         rlen := len
1300         if in != nil {
1301             // should watch the disconnection of out
1302             rcc, err := in.Read(buff[0:rlen])
1303             if err != nil {
1304                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1305                     what, rcc, err, in.Name())
1306                 break
1307             }
1308             rlen = rcc
1309             if string(buff[0:10]) == "(SoftEOF " {
1310                 var ecc int64 = 0
1311                 fmt.Sscanf(string(buff), "(SoftEOF %v", &ecc)
1312                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/%v\n",
1313                     what, ecc, total)
1314                 if ecc == total {
1315                     break
1316                 }
1317             }
1318         }
1319         wlen := rlen
1320         if out != nil {
1321             wcc, err := out.Write(buff[0:rlen])
1322             if err != nil {
1323                 fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1324                     what, wcc, err, out.Name())
1325                 break
1326             }
1327             wlen = wcc
1328         }
1329         if wlen < rlen {
1330             fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1331                 what, wlen, rlen)
1332             break;
1333         }
1334     }
1335     nio += 1
1336     total += int64(rlen)
1337     rem -= int64(rlen)
1338     if rem <= 0 {
1339         break
1340     }
1341 }
1342 Done := time.Now()
1343 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1344 TotalMB := float64(total)/1000000 //MB
1345 MBps := TotalMB / Elps
1346 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %3fMB/s\n",
1347     what, total, size, nio, absSize(total), MBps)
1348 return total
1349 }
1350 func (gsh*GshContext)RexecServer(argv []string){
1351     debug := true
1352     Start0 := time.Now()
1353     Start := Start0
1354     // if local == "" { local = "0.0.0.0:9999" }
1355     local := "0.0.0.0:9999"
1356
1357     if 0 < len(argv) {
1358         if argv[0] == "-s" {
1359             debug = false
1360             argv = argv[1:]
1361         }
1362     }
1363     if 0 < len(argv) {
1364         argv = argv[1:]
1365     }
1366     port, err := net.ResolveTCPAddr("tcp", local);
1367     if err != nil {
1368         fmt.Printf("--En- S: Address error: %s (%s)\n", local, err)
1369         return
1370     }
1371     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n", local);
1372     sconn, err := net.ListenTCP("tcp", port)
1373     if err != nil {
1374         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n", local, err)

```

```

1375     return
1376 }
1377
1378 reqbuf := make([]byte,LINESIZE)
1379 res := ""
1380 for {
1381     fmt.Printf(Elapsed(Start0)+"--In- S: Accepting at %s...\n",local);
1382     acconn, err := sconn.AcceptTCP()
1383     Start = time.Now()
1384     if err != nil {
1385         fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1386         return
1387     }
1388     clnt, _ := acconn.File()
1389     fd := clnt.Fd()
1390     if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d]\n",local,fd) }
1391     res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1392     fmt.Fprintf(clnt,"%s",res)
1393     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1394     count, err := clnt.Read(reqbuf)
1395     if err != nil {
1396         fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1397             count,err,string(reqbuf))
1398     }
1399     req := string(reqbuf[:count])
1400     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1401     reqv := strings.Split(string(req),"r")
1402     cmdv := gshScanArg(reqv[0],0)
1403     //cmdv := strings.Split(reqv[0]," ")
1404     switch cmdv[0] {
1405     case "HELO":
1406         res = fmt.Sprintf("250 %v",req)
1407     case "GET":
1408         // download {remotefile|-zN} [localfile]
1409         var dsize int64 = 32*1024*1024
1410         var bsize int = 64*1024
1411         var fname string = ""
1412         var in *os.File = nil
1413         var pseudoEOF = false
1414         if 1 < len(cmdv) {
1415             fname = cmdv[1]
1416             if strBegins(fname,"-z") {
1417                 fmt.Sscanf(fname[2:], "%d",&dsize)
1418             }else
1419             if strBegins(fname,"{") {
1420                 xin,xout,err := gsh.Popen(fname,"r")
1421                 if err {
1422                     }else{
1423                         xout.Close()
1424                         defer xin.Close()
1425                         in = xin
1426                         dsize = MaxStreamSize
1427                         pseudoEOF = true
1428                     }
1429                 }else{
1430                     xin,err := os.Open(fname)
1431                     if err != nil {
1432                         fmt.Printf("--En- GET (%v)\n",err)
1433                     }else{
1434                         defer xin.Close()
1435                         in = xin
1436                         fi,_ := xin.Stat()
1437                         dsize = fi.Size()
1438                     }
1439                 }
1440             }
1441             //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsize,bsize)
1442             res = fmt.Sprintf("200 %v\r\n",dsize)
1443             fmt.Fprintf(clnt,"%v",res)
1444             fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1445             wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
1446             if pseudoEOF {
1447                 // show end of stream data (its size) by OOB?
1448                 time.Sleep(100*1000*1000)
1449                 SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1450                 fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1451                 fmt.Fprintf(clnt,"%v\r\n",SoftEOF)
1452                 // with client generated random?
1453             }
1454             res = fmt.Sprintf("200 GET done\r\n")
1455         case "PUT":
1456             // upload {srcfile|-zN} [dstfile]
1457             var dsize int64 = 32*1024*1024
1458             var bsize int = 64*1024
1459             var fname string = ""
1460             var out *os.File = nil
1461             if 1 < len(cmdv) { // localfile
1462                 fmt.Sscanf(cmdv[1],"%d",&dsize)
1463             }
1464             if 2 < len(cmdv) {
1465                 fname = cmdv[2]
1466                 if fname == "-" {
1467                     // nul dev
1468                 }else
1469                 if strBegins(fname,"{") {
1470                     xin,xout,err := gsh.Popen(fname,"w")
1471                     if err {
1472                         }else{
1473                             xin.Close()
1474                             defer xout.Close()
1475                             out = xout
1476                         }
1477                     }else{
1478                         // should write to temporary file
1479                         // should suppress ^C on tty
1480                     }
1481                     xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1482                     //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1483                     if err != nil {
1484                         fmt.Printf("--En- PUT (%v)\n",err)
1485                     }else{
1486                         out = xout
1487                     }
1488                 }
1489                 fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1490                     fname,local,err)
1491             }
1492             fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1493             fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1494             fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1495             fileRelay("RecvPUT",clnt,out,dsize,bsize)
1496             res = fmt.Sprintf("200 PUT done\r\n")
1497         default:
1498             res = fmt.Sprintf("400 What? %v",req)
1499     }
1500     clnt.Write([]byte(res))

```

```

1500     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1501     aconn.Close();
1502     cInt.Close();
1503 }
1504     sconn.Close();
1505 }
1506 func (gsh*GshContext)RexecClient(argv[]string){
1507     debug := true
1508     Start := time.Now()
1509     if len(argv) == 1 {
1510         return
1511     }
1512     argv = argv[1:]
1513     if argv[0] == "-serv" {
1514         gsh.RexecServer(argv[1:])
1515         return
1516     }
1517     remote := "0.0.0.0:9999"
1518     if argv[0][0] == '#' {
1519         remote = argv[0][1:]
1520         argv = argv[1:]
1521     }
1522     if argv[0] == "-s" {
1523         debug = false
1524         argv = argv[1:]
1525     }
1526     dport, err := net.ResolveTCPAddr("tcp",remote);
1527     if err != nil {
1528         fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1529         return
1530     }
1531     fmt.Printf(Elapsed(Start)+"--In- C: Socket: connecting to %s\n",remote)
1532     serv, err := net.DialTCP("tcp",nil,dport)
1533     if err != nil {
1534         fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1535         return
1536     }
1537     if debug { fmt.Printf(Elapsed(Start)+"--In- C: Socket: connected to %s\n",remote) }
1538
1539     req := ""
1540     res := make([]byte,LINESIZE)
1541     count,err := serv.Read(res)
1542     if err != nil {
1543         fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1544     }
1545     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1546
1547     if argv[0] == "GET" {
1548         savPA := gsh.gshPA
1549         var bsize int = 64*1024
1550         req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1551         fmt.Printf(Elapsed(Start)+"--In- C: %v",req)
1552         fmt.Fprintf(serv,req)
1553         count,err = serv.Read(res)
1554         if err != nil {
1555             }else{
1556                 var dsize int64 = 0
1557                 var out *os.File = nil
1558                 var out_tobeclosed *os.File = nil
1559                 var fname string = ""
1560                 var rcode int = 0
1561                 var pid int = -1
1562                 fmt.Sscanf(string(res),"%d %d",&rcode,&dsize)
1563                 fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count]))
1564                 if 3 <= len(argv) {
1565                     fname = argv[2]
1566                     if strBegins(fname,"{") {
1567                         xin,xout,err := gsh.Popen(fname,"w")
1568                         if err {
1569                             }else{
1570                                 xin.Close()
1571                                 defer xout.Close()
1572                                 out = xout
1573                                 out_tobeclosed = xout
1574                                 pid = 0 // should be its pid
1575                             }
1576                         }else{
1577                             // should write to temporary file
1578                             // should suppress ^C on tty
1579                             xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1580                             if err != nil {
1581                                 fmt.Print("--En- %v\n",err)
1582                             }
1583                             out = xout
1584                         }
1585                     }
1586                     in, _ := serv.File()
1587                     fileRelay("RecvGET",in,out,dsize,bsize)
1588                     if 0 <= pid {
1589                         gsh.gshPA = savPA // recovery of Fd(), and more?
1590                         fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
1591                         out_tobeclosed.Close()
1592                         //syscall.Wait4(pid,nil,0,nil) //@@
1593                     }
1594                 }
1595             }else
1596             if argv[0] == "PUT" {
1597                 remote, _ := serv.File()
1598                 var local *os.File = nil
1599                 var dsize int64 = 32*1024*1024
1600                 var bsize int = 64*1024
1601                 var ofile string = ""
1602                 //fmt.Printf("--I-- Rex %v\n",argv)
1603                 if 1 < len(argv) {
1604                     fname := argv[1]
1605                     if strBegins(fname,"-z") {
1606                         fmt.Sscanf(fname[2:], "%d",&dsize)
1607                     }else
1608                     if strBegins(fname,"{") {
1609                         xin,xout,err := gsh.Popen(fname,"r")
1610                         if err {
1611                             }else{
1612                                 xout.Close()
1613                                 defer xin.Close()
1614                                 //in = xin
1615                                 local = xin
1616                                 fmt.Printf("--In- [%d] < Upload output of %v\n",
1617                                     local.Fd(),fname)
1618                                 ofile = "-from."+fname
1619                                 dsize = MaxStreamSize
1620                             }
1621                         }else{
1622                             xlocal,err := os.Open(fname)
1623                             if err != nil {
1624                                 fmt.Printf("--En- (%s)\n",err)

```

```

1625         local = nil
1626     }else{
1627         local = xlocal
1628         fi,_ := local.Stat()
1629         dsize = fi.Size()
1630         defer local.Close()
1631         //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsize)
1632     }
1633     ofile = fname
1634     fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
1635         fname,dsize,local,err)
1636 }
1637 }
1638 if 2 < len(argv) && argv[2] != "" {
1639     ofile = argv[2]
1640     //fmt.Printf("(%)%v B.ofile=%v\n",len(argv),argv,ofile)
1641 }
1642 //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
1643 fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1644 req = fmt.Sprintf("PUT %v %v \r\n",dsize,ofile)
1645 if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
1646 fmt.Fprintf(serv,"%v",req)
1647 count,err = serv.Read(req)
1648 if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
1649 fileRelay("SendPUT",local,remote,dsize,bsize)
1650 }else{
1651     req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1652     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
1653     fmt.Fprintf(serv,"%v",req)
1654     //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
1655 }
1656 //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
1657 count,err = serv.Read(res)
1658 ress := ""
1659 if count == 0 {
1660     ress = "(nil)\r\n"
1661 }else{
1662     ress = string(res[:count])
1663 }
1664 if err != nil {
1665     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,ress)
1666 }else{
1667     fmt.Printf(Elapsed(Start)+"--In- S: %v",ress)
1668 }
1669 serv.Close()
1670 //conn.Close()
1671 }
1672 }
1673 // <a name="remote-sh">Remote Shell</a>
1674 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
1675 func (gsh*GshContext)FileCopy(argv []string){
1676     var host = ""
1677     var port = ""
1678     var upload = false
1679     var download = false
1680     var xargv = []string{"rex-gcp"}
1681     var srcv = []string{}
1682     var dstv = []string{}
1683     argv = argv[1:]
1684
1685     for _,v := range argv {
1686         /*
1687         if v[0] == '-' { // might be a pseudo file (generated date)
1688             continue
1689         }
1690         */
1691         obj := strings.Split(v,":")
1692         //fmt.Printf("%d %v %v\n",len(obj),v,obj)
1693         if 1 < len(obj) {
1694             host = obj[0]
1695             file := ""
1696             if 0 < len(host) {
1697                 gsh.LastServer.host = host
1698             }else{
1699                 host = gsh.LastServer.host
1700                 port = gsh.LastServer.port
1701             }
1702             if 2 < len(obj) {
1703                 port = obj[1]
1704                 if 0 < len(port) {
1705                     gsh.LastServer.port = port
1706                 }else{
1707                     port = gsh.LastServer.port
1708                 }
1709                 file = obj[2]
1710             }else{
1711                 file = obj[1]
1712             }
1713             if len(srcv) == 0 {
1714                 download = true
1715                 srcv = append(srcv,file)
1716                 continue
1717             }
1718             upload = true
1719             dstv = append(dstv,file)
1720             continue
1721         }
1722         /*
1723         idx := strings.Index(v,":")
1724         if 0 <= idx {
1725             remote = v[0:idx]
1726             if len(srcv) == 0 {
1727                 download = true
1728                 srcv = append(srcv,v[idx+1:])
1729                 continue
1730             }
1731             upload = true
1732             dstv = append(dstv,v[idx+1:])
1733             continue
1734         }
1735         */
1736         if download {
1737             dstv = append(dstv,v)
1738         }else{
1739             srcv = append(srcv,v)
1740         }
1741     }
1742     hostport := "@" + host + ":" + port
1743     if upload {
1744         if host != "" { xargv = append(xargv,hostport) }
1745         xargv = append(xargv,"PUT")
1746         xargv = append(xargv,srcv[0:]...)
1747         xargv = append(xargv,dstv[0:]...)
1748     }else{
1749         //fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v // %v\n",hostport,dstv,srcv)
1750         fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v\n",hostport,dstv,srcv)
1751     }
1752 }

```

```

1750     gsh.RexecClient(xargv)
1751 }else
1752 if download {
1753     if host != "" { xargv = append(xargv,hostport) }
1754     xargv = append(xargv,"GET")
1755     xargv = append(xargv,srcv[0]...)
1756     xargv = append(xargv,dstv[0]...)
1757 //fmt.Printf("-I--- FileCopy GET gsh://%v/%v > %v // %v\n",hostport,srcv,dstv,xargv)
1758 fmt.Printf("-I--- FileCopy GET gsh://%v/%v > %v\n",hostport,srcv,dstv)
1759     gsh.RexecClient(xargv)
1760 }else{
1761 }
1762 }
1763 }
1764 // <a name="network">network</a>
1765 // -s, -si, -so // bi-directional, source, sync (maybe socket)
1766 func sconnect(gshCtx GshContext, inTCP bool, argv []string) {
1767     gshPA := gshCtx.gshPA
1768     if len(argv) < 2 {
1769         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
1770         return
1771     }
1772     remote := argv[1]
1773     if remote == ":" { remote = "0.0.0.0:9999" }
1774
1775     if inTCP { // TCP
1776         dport, err := net.ResolveTCPAddr("tcp",remote);
1777         if err != nil {
1778             fmt.Printf("Address error: %s (%s)\n",remote,err)
1779             return
1780         }
1781         conn, err := net.DialTCP("tcp",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         fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
1789
1790         savfd := gshPA.Files[1]
1791         gshPA.Files[1] = fd;
1792         gshelly(gshCtx, argv[2:])
1793         gshPA.Files[1] = savfd
1794         file.Close()
1795         conn.Close()
1796     }else{
1797         //dport, err := net.ResolveUDPAddr("udp4",remote);
1798         dport, err := net.ResolveUDPAddr("udp",remote);
1799         if err != nil {
1800             fmt.Printf("Address error: %s (%s)\n",remote,err)
1801             return
1802         }
1803         //conn, err := net.DialUDP("udp4",nil,dport)
1804         conn, err := net.DialUDP("udp",nil,dport)
1805         if err != nil {
1806             fmt.Printf("Connection error: %s (%s)\n",remote,err)
1807             return
1808         }
1809         file, _ := conn.File();
1810         fd := file.Fd()
1811
1812         ar := conn.RemoteAddr()
1813         //al := conn.LocalAddr()
1814         fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
1815             remote,ar.String(),fd)
1816
1817         savfd := gshPA.Files[1]
1818         gshPA.Files[1] = fd;
1819         gshelly(gshCtx, argv[2:])
1820         gshPA.Files[1] = savfd
1821         file.Close()
1822         conn.Close()
1823     }
1824 }
1825 func saccept(gshCtx GshContext, inTCP bool, argv []string) {
1826     gshPA := gshCtx.gshPA
1827     if len(argv) < 2 {
1828         fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
1829         return
1830     }
1831     local := argv[1]
1832     if local == ":" { local = "0.0.0.0:9999" }
1833     if inTCP { // TCP
1834         port, err := net.ResolveTCPAddr("tcp",local);
1835         if err != nil {
1836             fmt.Printf("Address error: %s (%s)\n",local,err)
1837             return
1838         }
1839         //fmt.Printf("Listen at %s...\n",local);
1840         sconn, err := net.ListenTCP("tcp", port)
1841         if err != nil {
1842             fmt.Printf("Listen error: %s (%s)\n",local,err)
1843             return
1844         }
1845         //fmt.Printf("Accepting at %s...\n",local);
1846         aconn, err := sconn.AcceptTCP()
1847         if err != nil {
1848             fmt.Printf("Accept error: %s (%s)\n",local,err)
1849             return
1850         }
1851         file, _ := aconn.File()
1852         fd := file.Fd()
1853         fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
1854
1855         savfd := gshPA.Files[0]
1856         gshPA.Files[0] = fd;
1857         gshelly(gshCtx, argv[2:])
1858         gshPA.Files[0] = savfd
1859
1860         sconn.Close();
1861         aconn.Close();
1862         file.Close();
1863     }else{
1864         //port, err := net.ResolveUDPAddr("udp4",local);
1865         port, err := net.ResolveUDPAddr("udp",local);
1866         if err != nil {
1867             fmt.Printf("Address error: %s (%s)\n",local,err)
1868             return
1869         }
1870         //uconn, err := net.ListenUDP("udp4", port)
1871         uconn, err := net.ListenUDP("udp", port)
1872         if err != nil {
1873             fmt.Printf("Listen error: %s (%s)\n",local,err)
1874

```

```

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

```



```

2000 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2001     if 0 <= hix && hix < len(gsh.CommandHistory) {
2002         return gsh.CommandHistory[hix].CmdLine,true
2003     }
2004     return "",false
2005 }
2006
2007 // temporary adding to PATH environment
2008 // cd name -lib for LD_LIBRARY_PATH
2009 // chdir with directory history (date + full-path)
2010 // -s for sort option (by visit date or so)
2011 func (gsh*GshContext)ShowChdirHistory1(i int,v GChdirHistory, argv []string){
2012     fmt.Printf("%s-%d ",v.CmdIndex) // the first command at this WorkDir
2013     fmt.Printf("%@d ",i)
2014     fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
2015     showFileInfo(v.Dir,argv)
2016 }
2017 func (gsh*GshContext)ShowChdirHistory(argv []string){
2018     for i, v := range gsh.ChrdirHistory {
2019         gsh.ShowChdirHistory1(i,v,argv)
2020     }
2021 }
2022 func skipOpts(argv[]string)(int){
2023     for i,v := range argv {
2024         if strBegins(v,"-") {
2025             }else{
2026                 return i
2027             }
2028     }
2029     return -1
2030 }
2031 func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
2032     cdhist := gshCtx.ChrdirHistory
2033     if isin("?",argv) || isin("-t",argv) || isin("-a",argv) {
2034         gshCtx.ShowChdirHistory(argv)
2035         return gshCtx
2036     }
2037     pwd, _ := os.Getwd()
2038     dir := ""
2039     if len(argv) <= 1 {
2040         dir = toFullPath("-")
2041     }else{
2042         i := skipOpts(argv[1:])
2043         if i < 0 {
2044             dir = toFullPath("-")
2045         }else{
2046             dir = argv[1+i]
2047         }
2048     }
2049     if strBegins(dir,"@") {
2050         if dir == "@0" { // obsolete
2051             dir = gshCtx.StartDir
2052         }else
2053         if dir == "@1" {
2054             index := len(cdhist) - 1
2055             if 0 < index { index -= 1 }
2056             dir = cdhist[index].Dir
2057         }else{
2058             index, err := strconv.Atoi(dir[1:])
2059             if err != nil {
2060                 fmt.Printf("--E-- xChdir(%v)\n",err)
2061                 dir = "?"
2062             }else
2063             if len(gshCtx.ChrdirHistory) <= index {
2064                 fmt.Printf("--E-- xChdir(history range error)\n")
2065                 dir = "?"
2066             }else{
2067                 dir = cdhist[index].Dir
2068             }
2069         }
2070     }
2071     if dir != "?" {
2072         err := os.Chdir(dir)
2073         if err != nil {
2074             fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)
2075         }else{
2076             cwd, _ := os.Getwd()
2077             if cwd != pwd {
2078                 hist1 := GChdirHistory { }
2079                 hist1.Dir = cwd
2080                 hist1.MovedAt = time.Now()
2081                 hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2082                 gshCtx.ChrdirHistory = append(cdhist,hist1)
2083                 if !isin("-s",argv){
2084                     //cwd, _ := os.Getwd()
2085                     //fmt.Printf("%s\n",cwd)
2086                     ix := len(gshCtx.ChrdirHistory)-1
2087                     gshCtx.ShowChdirHistory1(ix,hist1,argv)
2088                 }
2089             }
2090         }
2091     }
2092     if isin("-ls",argv){
2093         cwd, _ := os.Getwd()
2094         showFileInfo(cwd,argv);
2095     }
2096     return gshCtx
2097 }
2098 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2099     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2100 }
2101 func RusageSubv(rul, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
2102     TimeValSub(&rul[0].Utime,&ru2[0].Utime)
2103     TimeValSub(&rul[0].Stime,&ru2[0].Stime)
2104     TimeValSub(&rul[1].Utime,&ru2[1].Utime)
2105     TimeValSub(&rul[1].Stime,&ru2[1].Stime)
2106     return rul
2107 }
2108 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2109     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2110     return tvs
2111 }
2112 /*
2113 func RusageAddv(rul, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
2114     TimeValAdd(rul[0].Utime,ru2[0].Utime)
2115     TimeValAdd(rul[0].Stime,ru2[0].Stime)
2116     TimeValAdd(rul[1].Utime,ru2[1].Utime)
2117     TimeValAdd(rul[1].Stime,ru2[1].Stime)
2118     return rul
2119 }
2120 */
2121
2122 // <a name="rusage">Resource Usage</a>
2123 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2124     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)

```

```

2125 st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2126 fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2127 fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2128 return ""
2129 }
2130 func Getrusage()([2]syscall.Rusage){
2131 var ruv = [2]syscall.Rusage{}
2132 syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2133 syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2134 return ruv
2135 }
2136 func showRusage(what string,argv []string, ru *syscall.Rusage){
2137 fmt.Printf("%s:",what);
2138 fmt.Printf("User=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2139 fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2140 fmt.Printf(" Rss=%vB",ru.Maxrss)
2141 if isin("-l",argv) {
2142 fmt.Printf(" MinFlt=%v",ru.Minflt)
2143 fmt.Printf(" MajFlt=%v",ru.Majflt)
2144 fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2145 fmt.Printf(" IdRSS=%vB",ru.Idrss)
2146 fmt.Printf(" Nswap=%vB",ru.Nswap)
2147 fmt.Printf(" Read=%v",ru.Inblock)
2148 fmt.Printf(" Write=%v",ru.Oublock)
2149 }
2150 fmt.Printf(" Snd=%v",ru.Msgsnd)
2151 fmt.Printf(" Rcv=%v",ru.Msgrcv)
2152 //if isin("-l",argv) {
2153 fmt.Printf(" Sig=%v",ru.Nsignals)
2154 //}
2155 fmt.Printf("\n");
2156 }
2157 func xTime(gshCtx GshContext, argv []string)(GshContext,bool){
2158 if 2 <= len(argv){
2159 gshCtx.LastRusage = syscall.Rusage{}
2160 rusagev1 := Getrusage()
2161 xgshCtx, fin := gshellv(gshCtx,argv[1:])
2162 rusagev2 := Getrusage()
2163 gshCtx = xgshCtx
2164 showRusage(argv[1],argv,&gshCtx.LastRusage)
2165 rusagev := RusageSubv(rusagev2,rusagev1)
2166 showRusage("self",argv,&rusagev[0])
2167 showRusage("chld",argv,&rusagev[1])
2168 return gshCtx, fin
2169 }else{
2170 rusage:= syscall.Rusage {}
2171 syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2172 showRusage("self",argv, &rusage)
2173 syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2174 showRusage("chld",argv, &rusage)
2175 return gshCtx, false
2176 }
2177 }
2178 func xJobs(gshCtx GshContext, argv []string){
2179 fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
2180 for ji, pid := range gshCtx.BackGroundJobs {
2181 //wstat := syscall.WaitStatus {0}
2182 rusage := syscall.Rusage {}
2183 //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2184 wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2185 if err != nil {
2186 fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
2187 }else{
2188 fmt.Printf("%%d[%d] (%d)\n",ji,pid,wpid)
2189 showRusage("chld",argv,&rusage)
2190 }
2191 }
2192 }
2193 func inBackground(gshCtx GshContext, argv []string)(GshContext,bool){
2194 if gshCtx.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2195 gshCtx.BackGround = true // set background option
2196 xfin := false
2197 gshCtx, xfin = gshellv(gshCtx,argv)
2198 gshCtx.BackGround = false
2199 return gshCtx,xfin
2200 }
2201 // -o file without command means just opening it and refer by #N
2202 // should be listed by "files" command
2203 func xOpen(gshCtx GshContext, argv []string)(GshContext){
2204 var pv = [int{-1,-1}]
2205 err := syscall.Pipe(pv)
2206 fmt.Printf("--I-- pipe()=[%d,%d] (%v)\n",pv[0],pv[1],err)
2207 return gshCtx
2208 }
2209 func fromPipe(gshCtx GshContext, argv []string)(GshContext){
2210 return gshCtx
2211 }
2212 func xClose(gshCtx GshContext, argv []string)(GshContext){
2213 return gshCtx
2214 }
2215 }
2216 // <a name="redirect">redirect</a>
2217 func redirect(gshCtx GshContext, argv []string)(GshContext,bool){
2218 if len(argv) < 2 {
2219 return gshCtx, false
2220 }
2221 }
2222 cmd := argv[0]
2223 fname := argv[1]
2224 var file *os.File = nil
2225 }
2226 fdix := 0
2227 mode := os.O_RDONLY
2228 }
2229 switch {
2230 case cmd == "-i" || cmd == "<":
2231 fdix = 0
2232 mode = os.O_RDONLY
2233 case cmd == "-o" || cmd == ">":
2234 fdix = 1
2235 mode = os.O_RDWR | os.O_CREATE
2236 case cmd == "-a" || cmd == ">>":
2237 fdix = 1
2238 mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2239 }
2240 if fname[0] == '#' {
2241 fd, err := strconv.Atoi(fname[1:])
2242 if err != nil {
2243 fmt.Printf("--E-- (%v)\n",err)
2244 return gshCtx, false
2245 }
2246 file = os.NewFile(uintptr(fd),"MaybePipe")
2247 }else{
2248 xfile, err := os.OpenFile(argv[1], mode, 0600)
2249 if err != nil {

```

```

2250         fmt.Printf("--E-- (%s)\n",err)
2251         return gshCtx, false
2252     }
2253     file = xfile
2254 }
2255 gshPA := gshCtx.gshPA
2256 savfd := gshPA.Files[fdix]
2257 gshPA.Files[fdix] = file.Fd()
2258 fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2259 gshCtx, _ = gshellv(gshCtx, argv[2:])
2260 gshPA.Files[fdix] = savfd
2261
2262 return gshCtx, false
2263 }
2264
2265 //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2266 func httpHandler(res http.ResponseWriter, req *http.Request){
2267     path := req.URL.Path
2268     fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2269     {
2270         gshCtx, _ := setupGshContext()
2271         fmt.Printf("--I-- %s\n",path[1:])
2272         gshCtx, _ = tgshelll(gshCtx,path[1:])
2273     }
2274     fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
2275 }
2276 func httpServer(gshCtx GshContext, argv []string){
2277     http.HandleFunc("/", httpHandler)
2278     accport := "localhost:9999"
2279     fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2280     http.ListenAndServe(accport,nil)
2281 }
2282 func xGo(gshCtx GshContext, argv []string){
2283     go gshellv(gshCtx,argv[1:]);
2284 }
2285 func xPs(gshCtx GshContext, argv []string)(GshContext){
2286     return gshCtx
2287 }
2288
2289 // <a name="plugin">Plugin</a>
2290 // plugin [-ls [names]] to list plugins
2291 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code
2292 func whichPlugin(gshCtx GshContext,name string,argv []string)(pi *PluginInfo){
2293     pi = nil
2294     for _,p := range gshCtx.PluginFuncs {
2295         if p.Name == name && pi == nil {
2296             pi = p
2297         }
2298         if !isin("-s",argv){
2299             //fmt.Printf("%v %v ",i,p)
2300             if isin("-ls",argv){
2301                 showFileInfo(p.Path,argv)
2302             }else{
2303                 fmt.Printf("%s\n",p.Name)
2304             }
2305         }
2306     }
2307     return pi
2308 }
2309 func xPlugin(gshCtx GshContext, argv []string)(GshContext,error){
2310     if len(argv) == 0 || argv[0] == "-ls" {
2311         whichPlugin(gshCtx,"",argv)
2312         return gshCtx, nil
2313     }
2314     name := argv[0]
2315     Pin := whichPlugin(gshCtx,name,[]string{"-s"})
2316     if Pin != nil {
2317         os.Args = argv // should be recovered?
2318         Pin.Addr.(func())()
2319         return gshCtx,nil
2320     }
2321     sofile := toFullpath(argv[0] + ".so") // or find it by which($PATH)
2322
2323     p, err := plugin.Open(sofile)
2324     if err != nil {
2325         fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2326         return gshCtx, err
2327     }
2328     fname := "Main"
2329     f, err := p.Lookup(fname)
2330     if( err != nil ){
2331         fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2332         return gshCtx, err
2333     }
2334     pin := PluginInfo {p,f,name,sofile}
2335     gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2336     fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2337
2338     //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2339     os.Args = argv
2340     f.(func())()
2341     return gshCtx, err
2342 }
2343 func Args(gshCtx *GshContext, argv []string){
2344     for i,v := range os.Args {
2345         fmt.Printf("[%v] %v\n",i,v)
2346     }
2347 }
2348 func Version(gshCtx *GshContext, argv []string){
2349     if isin("-l",argv) {
2350         fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2351     }else{
2352         fmt.Printf("%v",VERSION);
2353     }
2354     if !isin("-n",argv) {
2355         fmt.Printf("\n")
2356     }
2357 }
2358
2359 // <a name="scanf">Scanf</a> // string decomposer
2360 // scanf [format] [input]
2361 func scanf(sstr string)(strv []string){
2362     strv = strings.Split(sstr, " ")
2363     return strv
2364 }
2365 func scanUntil(src,end string)(rstr string,leng int){
2366     idx := strings.Index(src,end)
2367     if 0 <= idx {
2368         rstr = src[0:idx]
2369         return rstr,idx+leng(end)
2370     }
2371     return src,0
2372 }
2373
2374 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting

```

```

2375 func (gsh*GshContext)printVal(fmts string, vstr string, optv[jstring]){
2376 //vint,err := strconv.Atoi(vstr)
2377 var ival int64 = 0
2378 n := 0
2379 err := error(nil)
2380 if strBegins(vstr, " ") {
2381 vx, _ := strconv.Atoi(vstr[1:])
2382 if vx < len(gsh.iValues) {
2383 vstr = gsh.iValues[vx]
2384 }else{
2385 }
2386 }
2387 // should use Eval()
2388 if strBegins(vstr, "0x") {
2389 n,err = fmt.Sscanf(vstr[2:], "%x", &ival)
2390 }else{
2391 n,err = fmt.Sscanf(vstr, "%d", &ival)
2392 //fmt.Printf("--D-- n=%d err=(%v) (%s)=%v\n",n,err,vstr, ival)
2393 }
2394 if n == 1 && err == nil {
2395 //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
2396 fmt.Printf("%"+fmts,ival)
2397 }else{
2398 if isin("-bn",optv){
2399 fmt.Printf("%"+fmts,filepath.Base(vstr))
2400 }else{
2401 fmt.Printf("%"+fmts,vstr)
2402 }
2403 }
2404 }
2405 func (gsh*GshContext)printfv(fmts,div string,argv[jstring],optv[jstring],list[jstring]){
2406 //fmt.Printf("%d",len(list))
2407 //curfmt := "v"
2408 outlen := 0
2409 curfmt := gsh.iFormat
2410
2411 if 0 < len(fmts) {
2412 for xi := 0; xi < len(fmts); xi++ {
2413 fch := fmts[xi]
2414 if fch == '%' {
2415 if xi+1 < len(fmts) {
2416 curfmt = string(fmts[xi+1])
2417 gsh.iFormat = curfmt
2418 xi += 1
2419 if xi+1 < len(fmts) && fmts[xi+1] == '(' {
2420 vals, leng := scanUntil(fmts[xi+2:],")")
2421 //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n",curfmt,vals,leng)
2422 gsh.printVal(curfmt,vals,optv)
2423 xi += 2+leng-1
2424 outlen += 1
2425 }
2426 }
2427 continue
2428 }
2429 if fch == ' ' {
2430 hi, leng := scanInt(fmts[xi+1:])
2431 if 0 < leng {
2432 if hi < len(gsh.iValues) {
2433 gsh.printVal(curfmt,gsh.iValues[hi],optv)
2434 outlen += 1 // should be the real length
2435 }else{
2436 fmt.Printf("((out-range))")
2437 }
2438 xi += leng
2439 continue;
2440 }
2441 }
2442 fmt.Printf("%c",fch)
2443 outlen += 1
2444 }
2445 }else{
2446 //fmt.Printf("--D-- print (%s)\n")
2447 for i,v := range list {
2448 if 0 < i {
2449 fmt.Printf(div)
2450 }
2451 gsh.printVal(curfmt,v,optv)
2452 outlen += 1
2453 }
2454 }
2455 if 0 < outlen {
2456 fmt.Printf("\n")
2457 }
2458 }
2459 func (gsh*GshContext)Scanv(argv[jstring]){
2460 //fmt.Printf("--D-- Scanv(%v)\n",argv)
2461 if len(argv) == 1 {
2462 return
2463 }
2464 argv = argv[1:]
2465 fmts := ""
2466 if strBegins(argv[0],"-F") {
2467 fmts = argv[0]
2468 gsh.iDelimiter = fmts
2469 argv = argv[1:]
2470 }
2471 input := strings.Join(argv, " ")
2472 if fmts == "" { // simple decomposition
2473 v := scanv(input)
2474 gsh.iValues = v
2475 //fmt.Printf("%v\n",strings.Join(v,","))
2476 }else{
2477 v := make([]jstring,8)
2478 n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
2479 fmt.Printf("--D-- Sscanf ->(%v) n=%d err=(%v)\n",v,n,err)
2480 gsh.iValues = v
2481 }
2482 }
2483 func (gsh*GshContext)Printv(argv[jstring]){
2484 if false { //@@@
2485 fmt.Printf("%v\n",strings.Join(argv[1:], " "))
2486 return
2487 }
2488 //fmt.Printf("--D-- Printv(%v)\n",argv)
2489 //fmt.Printf("%v\n",strings.Join(gsh.iValues,","))
2490 div := gsh.iDelimiter
2491 fmts := ""
2492 argv = argv[1:]
2493 if 0 < len(argv) {
2494 if strBegins(argv[0],"-F") {
2495 div = argv[0][2:]
2496 argv = argv[1:]
2497 }
2498 }
2499 }

```

```

2500     optv := []string{}
2501     for _, v := range argv {
2502         if strBegins(v, "-"){
2503             optv = append(optv, v)
2504             argv = argv[1:]
2505         }else{
2506             break;
2507         }
2508     }
2509     if 0 < len(argv) {
2510         fmts = strings.Join(argv, " ")
2511     }
2512     gsh.printf(fmts, div, argv, optv, gsh.iValues)
2513 }
2514 func (gsh*GshContext)Basename(argv[]string){
2515     for i, v := range gsh.iValues {
2516         gsh.iValues[i] = filepath.Base(v)
2517     }
2518 }
2519 func (gsh*GshContext)Sortv(argv[]string){
2520     sv := gsh.iValues
2521     sort.Slice(sv, func(i, j int) bool {
2522         return sv[i] < sv[j]
2523     })
2524 }
2525 func (gsh*GshContext)Shiftv(argv[]string){
2526     vi := len(gsh.iValues)
2527     if 0 < vi {
2528         if isin("-r", argv) {
2529             top := gsh.iValues[0]
2530             gsh.iValues = append(gsh.iValues[1:], top)
2531         }else{
2532             gsh.iValues = gsh.iValues[1:]
2533         }
2534     }
2535 }
2536
2537 func (gsh*GshContext)Enq(argv[]string){
2538 }
2539 func (gsh*GshContext)Deq(argv[]string){
2540 }
2541 func (gsh*GshContext)Push(argv[]string){
2542     gsh.iValStack = append(gsh.iValStack, argv[1:])
2543     fmt.Printf("depth=%d\n", len(gsh.iValStack))
2544 }
2545 func (gsh*GshContext)Dump(argv[]string){
2546     for i, v := range gsh.iValStack {
2547         fmt.Printf("%d %v\n", i, v)
2548     }
2549 }
2550 func (gsh*GshContext)Pop(argv[]string){
2551     depth := len(gsh.iValStack)
2552     if 0 < depth {
2553         v := gsh.iValStack[depth-1]
2554         if isin("-cat", argv){
2555             gsh.iValues = append(gsh.iValues, v...)
2556         }else{
2557             gsh.iValues = v
2558         }
2559         gsh.iValStack = gsh.iValStack[0:depth-1]
2560         fmt.Printf("depth=%d %s\n", len(gsh.iValStack), gsh.iValues)
2561     }else{
2562         fmt.Printf("depth=%d\n", depth)
2563     }
2564 }
2565
2566 // <a name="interpreter">Command Interpreter</a>
2567 func gshellv(gshCtx GshContext, argv []string) (_ GshContext, fin bool) {
2568     fin = false
2569
2570     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr, "--I-- gshellv(%d)\n", len(argv)) }
2571     if len(argv) <= 0 {
2572         return gshCtx, false
2573     }
2574     xargv := []string{}
2575     for ai := 0; ai < len(argv); ai++ {
2576         xargv = append(xargv, strsubst(&gshCtx, argv[ai], false))
2577     }
2578     argv = xargv
2579     if false {
2580         for ai := 0; ai < len(argv); ai++ {
2581             fmt.Printf("[%d] %s [%d]T\n",
2582                 ai, argv[ai], len(argv[ai]), argv[ai])
2583         }
2584     }
2585     cmd := argv[0]
2586     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr, "--I-- gshellv(%d)%v\n", len(argv), argv) }
2587     switch { // https://tour.golang.org/flowcontrol/11
2588     case cmd == "":
2589         xPwd(gshCtx, []string{}); // empty command
2590     case cmd == "-x":
2591         gshCtx.CmdTrace = ! gshCtx.CmdTrace
2592     case cmd == "-xt":
2593         gshCtx.CmdTime = ! gshCtx.CmdTime
2594     case cmd == "-ot":
2595         sconnect(gshCtx, true, argv)
2596     case cmd == "-ou":
2597         sconnect(gshCtx, false, argv)
2598     case cmd == "-it":
2599         saccept(gshCtx, true, argv)
2600     case cmd == "-iu":
2601         saccept(gshCtx, false, argv)
2602     case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
2603         redirect(gshCtx, argv)
2604     case cmd == "|":
2605         gshCtx = fromPipe(gshCtx, argv)
2606     case cmd == "args":
2607         Args(&gshCtx, argv)
2608     case cmd == "bg" || cmd == "-bg":
2609         rgshCtx, rfin := inBackground(gshCtx, argv[1:])
2610         return rgshCtx, rfin
2611     case cmd == "-bn":
2612         gshCtx.Basename(argv)
2613     case cmd == "call":
2614         _, _ = gshCtx.excommand(false, argv[1:])
2615     case cmd == "cd" || cmd == "chdir":
2616         gshCtx = xChdir(gshCtx, argv);
2617     case cmd == "close":
2618         gshCtx = xClose(gshCtx, argv)
2619     case cmd == "gcp":
2620         gshCtx.FileCopy(argv)
2621     case cmd == "dec" || cmd == "decode":
2622         Dec(&gshCtx, argv)
2623     case cmd == "#define":
2624     case cmd == "dump":

```

```

2625     gshCtx.Dump(argv)
2626 case cmd == "echo":
2627     echo(argv,true)
2628 case cmd == "enc" || cmd == "encode":
2629     Enc(&gshCtx,argv)
2630 case cmd == "env":
2631     env(argv)
2632 case cmd == "eval":
2633     xEval(argv[1:],true)
2634 case cmd == "exec":
2635     _ = gshCtx.excommand(true,argv[1:])
2636     // should not return here
2637 case cmd == "exit" || cmd == "quit":
2638     // write Result code EXIT to 3>
2639     return gshCtx, true
2640 case cmd == "fds":
2641     // dump the attributes of fds (of other process)
2642 case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
2643     gshCtx.xFind(argv[1:])
2644 case cmd == "fu":
2645     gshCtx.xFind(argv[1:])
2646 case cmd == "fork":
2647     // mainly for a server
2648 case cmd == "-gen":
2649     gen(gshCtx, argv)
2650 case cmd == "-go":
2651     xGo(gshCtx, argv)
2652 case cmd == "-grep":
2653     gshCtx.xFind(argv)
2654 case cmd == "gdeg":
2655     gshCtx.Deg(argv)
2656 case cmd == "genq":
2657     gshCtx.Enq(argv)
2658 case cmd == "gpop":
2659     gshCtx.Pop(argv)
2660 case cmd == "gpush":
2661     gshCtx.Push(argv)
2662 case cmd == "history" || cmd == "hi": // hi should be alias
2663     gshCtx = xHistory(gshCtx, argv)
2664 case cmd == "jobs":
2665     xJobs(gshCtx,argv)
2666 case cmd == "lnsp":
2667     SplitLine(&gshCtx,argv)
2668 case cmd == "-ls":
2669     gshCtx.xFind(argv)
2670 case cmd == "nop":
2671     // do nothing
2672 case cmd == "pipe":
2673     gshCtx = xOpen(gshCtx,argv)
2674 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
2675     gshCtx, _ = xPlugin(gshCtx,argv[1:])
2676 case cmd == "print" || cmd == "-pr":
2677     // output internal slice // also sprintf should be
2678     gshCtx.Printv(argv)
2679 case cmd == "ps":
2680     xPs(gshCtx,argv)
2681 case cmd == "pstitle":
2682     // to be gsh.title
2683 case cmd == "rexecl" || cmd == "rexd":
2684     gshCtx.RexecServer(argv)
2685 case cmd == "rexec" || cmd == "rex":
2686     gshCtx.RexecClient(argv)
2687 case cmd == "repeat" || cmd == "rep": // repeat cond command
2688     repeat(gshCtx,argv)
2689 case cmd == "scan":
2690     // scan input (or so in fscanf) to internal slice (like Files or map)
2691     gshCtx.Scanv(argv)
2692 case cmd == "set":
2693     // set name ...
2694 case cmd == "serv":
2695     httpServer(gshCtx,argv)
2696 case cmd == "shift":
2697     gshCtx.Shiftv(argv)
2698 case cmd == "sleep":
2699     sleep(gshCtx,argv)
2700 case cmd == "-sort":
2701     gshCtx.Sortv(argv)
2702 case cmd == "time":
2703     gshCtx, fin = xTime(gshCtx,argv)
2704 case cmd == "pwd":
2705     xPwd(gshCtx,argv);
2706 case cmd == "ver" || cmd == "-ver" || cmd == "version":
2707     Version(&gshCtx,argv)
2708 case cmd == "where":
2709     // data file or so?
2710 case cmd == "which":
2711     which("PATH",argv);
2712 default:
2713     if whichPlugin(gshCtx,cmd,[string{"-s"}]) != nil {
2714         gshCtx, _ = xPlugin(gshCtx,argv)
2715     }else{
2716         notfound, _ := gshCtx.excommand(false,argv)
2717         if notfound {
2718             fmt.Printf("--E-- command not found (%v)\n",cmd)
2719         }
2720     }
2721 }
2722 return gshCtx, fin
2723 }
2724 }
2725 func gshell1(gshCtx GshContext, gline string) (gx GshContext, rfin bool) {
2726     argv := strings.Split(string(gline)," ")
2727     gshCtx, fin := gshellv(gshCtx,argv)
2728     return gshCtx, fin
2729 }
2730 func tgshell1(gshCtx GshContext, gline string) (gx GshContext, xfin bool) {
2731     start := time.Now()
2732     gshCtx, fin := gshell1(gshCtx,gline)
2733     end := time.Now()
2734     elps := end.Sub(start);
2735     if gshCtx.CmdTime {
2736         fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
2737             elps/1000000000,elps%1000000000)
2738     }
2739     return gshCtx, fin
2740 }
2741 func Ttyid() (int) {
2742     fi, err := os.Stdin.Stat()
2743     if err != nil {
2744         return 0;
2745     }
2746     //fmt.Printf("Stdin: %v Dev=%d\n",
2747     // fi.Mode(),fi.Mode()&os.ModeDevice)
2748     if (fi.Mode() & os.ModeDevice) != 0 {
2749         stat := syscall.Stat_t{};

```

```

2750     err := syscall.Fstat(0,&stat)
2751     if err != nil {
2752         //fmt.Printf("--I-- Stdin: (%v)\n",err)
2753     }else{
2754         //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
2755             // stat.Rdev&0xFF,stat.Rdev);
2756         //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
2757         return int(stat.Rdev & 0xFF)
2758     }
2759 }
2760 return 0
2761 }
2762 func ttyfile(gshCtx GshContext) string {
2763     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
2764     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
2765         fmt.Sprintf("%02d",gshCtx.TerminalId)
2766     //strconv.Itoa(gshCtx.TerminalId)
2767     //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
2768     return ttyfile
2769 }
2770 func ttyline(gshCtx GshContext) (*os.File){
2771     file,err := os.OpenFile(ttyfile(gshCtx),
2772         os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
2773     if err != nil {
2774         fmt.Printf("--F-- cannot open %s (%s)\n",ttyfile(gshCtx),err)
2775         return file;
2776     }
2777     return file
2778 }
2779 func getline(gshCtx *GshContext, hix int, skipping bool, prevline string) (string) {
2780     if( skipping ){
2781         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
2782         line, _, _ := reader.ReadLine()
2783         return string(line)
2784     }else
2785     if true {
2786         return xgetline(hix,prevline,gshCtx)
2787     }
2788     /*
2789     else
2790     if( with_exgetline && gshCtx.GetLine != "" ){
2791         //var xhix int64 = int64(hix); // cast
2792         newenv := os.Environ()
2793         newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
2794
2795         tty := ttyline(gshCtx)
2796         tty.WriteString(prevline)
2797         Pa := os.ProcAttr {
2798             "", // start dir
2799             newenv, //os.Environ(),
2800             []os.File{os.Stdin,os.Stdout,os.Stderr,tty},
2801             nil,
2802         }
2803         //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
2804         proc := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
2805         if err != nil {
2806             fmt.Printf("--F-- getline process error (%v)\n",err)
2807             // for i; { }
2808             return "exit (getline program failed)"
2809         }
2810         //stat, err := proc.Wait()
2811         proc.Wait()
2812         buff := make([]byte,LINESIZE)
2813         count, err := tty.Read(buff)
2814         //_, err = tty.Read(buff)
2815         //fmt.Printf("--D-- getline (%d)\n",count)
2816         if err != nil {
2817             if ! (count == 0) { // && err.String() == "EOF" } {
2818                 fmt.Printf("--E-- getline error (%s)\n",err)
2819             }
2820         }else{
2821             //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
2822         }
2823         tty.Close()
2824         gline := string(buff[0:count])
2825         return gline
2826     }else
2827     /*
2828     {
2829         // if isatty {
2830             fmt.Printf("!%d",hix)
2831             fmt.Print(PROMPT)
2832         // }
2833         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
2834         line, _, _ := reader.ReadLine()
2835         return string(line)
2836     }
2837 }
2838
2839 //== begin ===== getline
2840 /*
2841 * getline.c
2842 * 2020-0819 extracted from dog.c
2843 * getline.go
2844 * 2020-0822 ported to Go
2845 */
2846 /*
2847 package main // getline main
2848 import (
2849     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
2850     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
2851     "os" // <a href="https://golang.org/pkg/os/">os</a>
2852     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
2853     // "bytes" // <a href="https://golang.org/pkg/bytes/">bytes</a>
2854     // "os/exec" // <a href="https://golang.org/pkg/os/">os</a>
2855 )
2856 */
2857
2858 // C language compatibility functions
2859 var errno = 0
2860 var stdin *os.File = os.Stdin
2861 var stdout *os.File = os.Stdout
2862 var stderr *os.File = os.Stderr
2863 var EOF = -1
2864 var NULL = 0
2865 type FILE os.File
2866 type StrBuff []byte
2867 var NULL_FP *os.File = nil
2868 var NULLSP = 0
2869 //var LINESIZE = 1024
2870
2871 func system(cmdstr string)(int){
2872     PA := syscall.ProcAttr {
2873         "", // the starting directory
2874         os.Environ(),

```

```

2875     [uintptr(os.Stdin.Fd()),os.Stdout.Fd(),os.Stderr.Fd())},
2876     nil,
2877 }
2878 argv := strings.Split(cmdstr, " ")
2879 pid,err := syscall.ForkExec(argv[0],argv,&PA)
2880 if( err != nil ){
2881     fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
2882 }
2883 syscall.Wait4(pid,nil,0,nil)
2884
2885 /*
2886 argv := strings.Split(cmdstr, " ")
2887 fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
2888 //cmd := exec.Command(argv[0]:...)
2889 cmd := exec.Command(argv[0],argv[1],argv[2])
2890 cmd.Stdin = strings.NewReader("output of system")
2891 var out bytes.Buffer
2892 cmd.Stdout = &out
2893 var serr bytes.Buffer
2894 cmd.Stderr = &serr
2895 err := cmd.Run()
2896 if err != nil {
2897     fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)
2898     fmt.Printf("ERR:%s\n",serr.String())
2899 }else{
2900     fmt.Printf("%s",out.String())
2901 }
2902 */
2903 return 0
2904 }
2905 func atoi(str string)(ret int){
2906     ret,err := fmt.Sscanf(str,"%d",ret)
2907     if err == nil {
2908         return ret
2909     }else{
2910         // should set errno
2911         return 0
2912     }
2913 }
2914 func getenv(name string)(string){
2915     val,got := os.LookupEnv(name)
2916     if got {
2917         return val
2918     }else{
2919         return "?"
2920     }
2921 }
2922 func strcpy(dst StrBuff, src string){
2923     var i int
2924     srcb := []byte(src)
2925     for i = 0; i < len(src) && srcb[i] != 0; i++ {
2926         dst[i] = srcb[i]
2927     }
2928     dst[i] = 0
2929 }
2930 func xstrcpy(dst StrBuff, src StrBuff){
2931     dst = src
2932 }
2933 func strcat(dst StrBuff, src StrBuff){
2934     dst = append(dst,src...)
2935 }
2936 func strdup(str StrBuff)(string){
2937     return string(str[0:strlen(str)])
2938 }
2939 func strlen(str string)(int){
2940     return len(str)
2941 }
2942 func strlen(str StrBuff)(int){
2943     var i int
2944     for i = 0; i < len(str) && str[i] != 0; i++ {
2945     }
2946     return i
2947 }
2948 func sizeof(data StrBuff)(int){
2949     return len(data)
2950 }
2951 func isatty(fd int)(ret int){
2952     return 1
2953 }
2954
2955 func fopen(file string,mode string)(fp*os.File){
2956     if mode == "r" {
2957         fp,err := os.Open(file)
2958         if( err != nil ){
2959             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
2960             return NULL_FP;
2961         }
2962         return fp;
2963     }else{
2964         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
2965         if( err != nil ){
2966             return NULL_FP;
2967         }
2968         return fp;
2969     }
2970 }
2971 func fclose(fp*os.File){
2972     fp.Close()
2973 }
2974 func fflush(fp *os.File)(int){
2975     return 0
2976 }
2977 func fgetc(fp*os.File)(int){
2978     var buf [1]byte
2979     _,err := fp.Read(buf[0:1])
2980     if( err != nil ){
2981         return EOF;
2982     }else{
2983         return int(buf[0])
2984     }
2985 }
2986 func sfgets(str*string, size int, fp*os.File)(int){
2987     buf := make(StrBuff,size)
2988     var ch int
2989     var i int
2990     for i = 0; i < len(buf)-1; i++ {
2991         ch = fgetc(fp)
2992         //fprintf(stderr,"--fgets %d/%d %X\n",i,len(buf),ch)
2993         if( ch == EOF ){
2994             break;
2995         }
2996         buf[i] = byte(ch);
2997         if( ch == '\n' ){
2998             break;
2999         }

```



```

3000     }
3001     buf[i] = 0
3002     //fprintf(stderr, "--fgets %d/%d (%s)\n", i, len(buf), buf[0:i])
3003     return i
3004 }
3005 func fgets(buf StrBuff, size int, fp*os.File)(int){
3006     var ch int
3007     var i int
3008     for i = 0; i < len(buf)-1; i++ {
3009         ch = fgetc(fp)
3010         //fprintf(stderr, "--fgets %d/%d %X\n", i, len(buf), ch)
3011         if( ch == EOF ){
3012             break;
3013         }
3014         buf[i] = byte(ch);
3015         if( ch == '\n' ){
3016             break;
3017         }
3018     }
3019     buf[i] = 0
3020     //fprintf(stderr, "--fgets %d/%d (%s)\n", i, len(buf), buf[0:i])
3021     return i
3022 }
3023 func fputc(ch int , fp*os.File)(int){
3024     var buf []byte
3025     buf[0] = byte(ch)
3026     fp.Write(buf[0:1])
3027     return 0
3028 }
3029 func fputs(buf StrBuff, fp*os.File)(int){
3030     fp.Write(buf)
3031     return 0
3032 }
3033 func xputss(str string, fp*os.File)(int){
3034     return fputs([]byte(str), fp)
3035 }
3036 func sscanf(str StrBuff, fmts string, params ...interface{})(int){
3037     fmt.Sscanf(string(str[0:strlen(str)]), fmts, params...)
3038     return 0
3039 }
3040 func fprintf(fp*os.File, fmts string, params ...interface{})(int){
3041     fmt.Fprintf(fp, fmts, params...)
3042     return 0
3043 }
3044 }
3045 // <a name="IME">Command Line IME</a>
3046 //----- MyIME
3047 var MyIMEVER = "MyIME/0.0.2";
3048 type RomKana struct {
3049     pat string;
3050     out string;
3051 }
3052 var dicents = 0
3053 var romkana [1024]RomKana
3054 func readDic()(int){
3055     var rk *os.File;
3056     var dic = "MyIME-dic.txt";
3057     //rk = fopen("romkana.txt", "r");
3058     //rk = fopen("JK-JA-morse-dic.txt", "r");
3059     rk = fopen(dic, "r");
3060     if( rk == NULLFP ){
3061         if( true ){
3062             fprintf(stderr, "--%s-- Could not load %s\n", MyIMEVER, dic);
3063         }
3064         return -1;
3065     }
3066     if( true ){
3067         var di int;
3068         var line = make(StrBuff, 1024);
3069         var pat string
3070         var out string
3071         for di = 0; di < 1024; di++ {
3072             if( fgets(line, sizeof(line), rk) == NULLSP ){
3073                 break;
3074             }
3075             fmt.Sscanf(string(line[0:strlen(line)]), "%s %s", &pat, &out);
3076             //sscanf(line, "%s %[^\r\n]", &pat, &out);
3077             romkana[di].pat = pat;
3078             romkana[di].out = out;
3079             //fprintf(stderr, "--Dd- %10s %s\n", pat, out)
3080         }
3081         dicents += di
3082         if( false ){
3083             fprintf(stderr, "--%s-- loaded romkana.txt [%d]\n", MyIMEVER, di);
3084             for di = 0; di < dicents; di++ {
3085                 fprintf(stderr,
3086                     "%s %s\n", romkana[di].pat, romkana[di].out);
3087             }
3088         }
3089     }
3090     fclose(rk);
3091 }
3092 //romkana[dicents].pat = "//ddump"
3093 //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3094 return 0;
3095 }
3096 func matchlen(stri string, pati string)(int){
3097     if strBegins(stri, pati) {
3098         return len(pati)
3099     }else{
3100         return 0
3101     }
3102 }
3103 func convs(src string)(string){
3104     var si int;
3105     var sx = len(src);
3106     var di int;
3107     var mi int;
3108     var dstb []byte
3109 }
3110 for si = 0; si < sx; { // search max. match from the position
3111     if strBegins(src[si:], "%x/") {
3112         // %x/integer/ // s/a/b/
3113         ix := strings.Index(src[si+3:], "/")
3114         if 0 < ix {
3115             var iv int = 0
3116             //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3117             fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3118             sval := fmt.Sprintf("%x", iv)
3119             bval := []byte(sval)
3120             dstb = append(dstb, bval...)
3121             si = si+3+ix+1
3122             continue
3123         }
3124     }

```

```

3125     if strBegins(src[si:], "%d/") {
3126         // %d/integer/ // s/a/b/
3127         ix := strings.Index(src[si+3:], "/")
3128         if 0 < ix {
3129             var iv int = 0
3130             fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3131             sval := fmt.Sprintf("%d", iv)
3132             bval := []byte(sval)
3133             dstb = append(dstb, bval...)
3134             si = si+3+ix+1
3135             continue
3136         }
3137     }
3138     var maxlen int = 0;
3139     var len int;
3140     mi = -1;
3141     for di = 0; di < dicents; di++ {
3142         len = matchlen(src[si:], romkana[di].pat);
3143         if( maxlen < len ){
3144             maxlen = len;
3145             mi = di;
3146         }
3147     }
3148     if( 0 < maxlen ){
3149         out := romkana[mi].out;
3150         dstb = append(dstb, []byte(out)...);
3151         si += maxlen;
3152     }else{
3153         dstb = append(dstb, src[si])
3154         si += 1;
3155     }
3156 }
3157 return string(dstb)
3158 }
3159 func trans(src string)(int){
3160     dst := convs(src);
3161     xputss(dst, stderr);
3162     return 0;
3163 }
3164
3165 //----- LINEEDIT
3166 // "?" at the top of the line means searching history
3167
3168 var GO_UP = 201
3169 var GO_DOWN = 202
3170 var GO_RIGHT = 203
3171 var GO_LEFT = 204
3172
3173 func getesc(in *os.File)(int){
3174     var ch1 int
3175     var ch2 int
3176     ch1 = fgetc(in);
3177     ch2 = fgetc(in);
3178     if false {
3179         fprintf(stderr, "(%c/%X %c/%X)", ch1, ch1, ch2, ch2);
3180     }
3181     switch( ch1 ){
3182     case '|':
3183         switch( ch2 ){
3184             case 'A': return GO_UP; // ^
3185             case 'B': return GO_DOWN; // v
3186             case 'C': return GO_RIGHT; // >
3187             case 'D': return GO_LEFT; // <
3188         }
3189         break;
3190     }
3191     return 0;
3192 }
3193 func clearline(){
3194     var i int
3195     fprintf(stderr, "\r");
3196     for i = 0; i < 80; i++ {
3197         fputc(' ', os.Stderr);
3198     }
3199     fprintf(stderr, "\r");
3200 }
3201 var romkanmode bool;
3202 var insertmode int;
3203 func redraw(lno int, line string, right string){
3204     var bsi int
3205     var rlen int
3206     var romkanmark string
3207
3208     if( romkanmode ){
3209         //romkanmark = " *";
3210     }else{
3211         romkanmark = "";
3212     }
3213     clearline();
3214     xputss("\r", stderr);
3215     if( romkanmode ){
3216         fprintf(stderr, "[\343\201\202r]");
3217         //fprintf(stderr, "[R]");
3218     }
3219     fprintf(stderr, "%d! ", lno);
3220     if( romkanmode ){
3221         trans(line);
3222         //fputs(romkanmark, stderr);
3223         trans(right);
3224     }else{
3225         xputss(line, stderr);
3226         //fputs(romkanmark, stderr);
3227         xputss(right, stderr);
3228     }
3229     if true { //romkanmode {
3230         fprintf(stderr, "\r")
3231         if romkanmode {
3232             fprintf(stderr, "[\343\201\202r]");
3233             fprintf(stderr, "%d! ", lno);
3234             trans(line);
3235         }else{
3236             fprintf(stderr, "%d! ", lno);
3237             xputss(line, stderr);
3238         }
3239     }else{
3240         rlen = len(right) + len(romkanmark);
3241         if true {
3242             for bsi = 0; bsi < rlen; bsi++ {
3243                 fputc('\b', stderr);
3244             }
3245         }
3246     }
3247 }
3248 func delHeadChar(str string)(rline string, head string){
3249     _, clen := utf8.DecodeRune([]byte(str))

```

```

3250     head = string(str[0:clen])
3251     return str[clen:],head
3252 }
3253 func delTailChar(str string)(rline string, last string){
3254     var i = 0
3255     var clen = 0
3256     for {
3257         _,siz := utf8.DecodeRune([]byte(str)[i:])
3258         if siz <= 0 { break }
3259         clen = siz
3260         i += siz
3261     }
3262     last = str[len(str)-clen:]
3263     return str[0:len(str)-clen],last
3264 }
3265
3266 // 3> for output and history
3267 // 4> for keylog?
3268 // <a name="getline">Command Line Editor</a>
3269 func xgetline(lno int, prevline string, gsh*GshContext)(string){
3270     lastlno := lno;
3271     line := ""
3272     right := ""
3273
3274     //readDic();
3275     if( isatty(0) == 0 ){
3276         if( sfgets(&line,LINESIZE,stdin) == NULL ){
3277             line = "exit\n";
3278         }else{
3279         }
3280         goto EXIT_GOT;
3281     }
3282     if( true ){
3283         //var pts string;
3284         //pts = ptsname(0);
3285         //pts = ttyname(0);
3286         //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
3287     }
3288     if( false ){
3289         fprintf(stderr,"! ");
3290         fflush(stderr);
3291         sfgets(&line,LINESIZE,stdin);
3292     }else{
3293         var ch int;
3294
3295         system("/bin/stty -echo -icanon");
3296         redraw(lno,line,right);
3297         line = ""
3298         right = ""
3299         pch := -1
3300         for {
3301             if( pch != -1 ){
3302                 ch = pch
3303                 pch = -1
3304             }else{
3305                 ch = fgetc(stdin);
3306             }
3307             if( ch == 033 ){
3308                 ch = getesc(stdin);
3309             }
3310             if( ch == '\\ '){
3311                 fputc(ch,stderr)
3312                 ch = fgetc(stdin)
3313                 if( ch == 'j' || ch == 'J' ){
3314                     readDic();
3315                     romkanmode = !romkanmode;
3316                     if( ch == 'J' ){
3317                         fprintf(stderr,"J\r\n");
3318                     }
3319                     redraw(lno,line,right);
3320                     continue
3321                 }else
3322                 if( ch == 'i' || ch == 'I' ){
3323                     dst := convs(line+right);
3324                     line = dst
3325                     right = ""
3326                     if( ch == 'I' ){
3327                         fprintf(stderr,"I\r\n");
3328                     }
3329                     redraw(lno,line,right);
3330                     continue
3331                 }else{
3332                     pch = ch
3333                     ch = '\\ '
3334                 }
3335             }
3336             switch( ch ){
3337                 case 0:
3338                     continue;
3339                 case GO_UP:
3340                     if lno == 1 {
3341                         continue
3342                     }
3343                     cmd,ok := gsh.cmdStringInHistory(lno-1)
3344                     if ok {
3345                         line = cmd
3346                         right = ""
3347                         lno = lno - 1
3348                     }
3349                     redraw(lno,line,right);
3350                     continue
3351                 case GO_DOWN:
3352                     cmd,ok := gsh.cmdStringInHistory(lno+1)
3353                     if ok {
3354                         line = cmd
3355                         right = ""
3356                         lno = lno + 1
3357                     }else{
3358                         line = ""
3359                         right = ""
3360                         if lno == lastlno-1 {
3361                             lno = lno + 1
3362                         }
3363                     }
3364                     redraw(lno,line,right);
3365                     continue
3366                 case GO_LEFT:
3367                     if 0 < len(line) {
3368                         xline,tail := delTailChar(line)
3369                         line = xline
3370                         right = tail + right
3371                     }
3372                     redraw(lno,line,right);
3373                     continue;
3374                 case GO_RIGHT:

```

```

3375         if( 0 < len(right) && right[0] != 0 ){
3376             xright,head := delHeadChar(right)
3377             right = xright
3378             line += head
3379         }
3380         redraw(lno,line,right);
3381         continue;
3382     case EOF:
3383         goto EXIT;
3384     case 'R'-0x40: // replace
3385         dst := convs(line+right);
3386         line = dst
3387         right = ""
3388         redraw(lno,line,right);
3389         continue;
3390     case 'M'-0x40: // just show the result
3391         readDic();
3392         romkanmode = !romkanmode;
3393         redraw(lno,line,right);
3394         continue;
3395     case 'I'-0x40:
3396         redraw(lno,line,right);
3397         continue
3398     case 'K'-0x40:
3399         right = ""
3400         redraw(lno,line,right);
3401         continue
3402     case 'E'-0x40:
3403         line += right
3404         right = ""
3405         redraw(lno,line,right);
3406         continue
3407     case 'A'-0x40:
3408         right = line + right
3409         line = ""
3410         redraw(lno,line,right);
3411         continue
3412     case 'U'-0x40:
3413         line = ""
3414         right = ""
3415         clearline();
3416         redraw(lno,line,right);
3417         continue;
3418     case 0x7F: // DEL
3419         if( 0 < len(line) ){
3420             line,_ = delTailChar(line)
3421             redraw(lno,line,right);
3422         }
3423         continue;
3424     case 'H'-0x40:
3425         if( 0 < len(line) ){
3426             line,_ = delTailChar(line)
3427             redraw(lno,line,right);
3428         }
3429         continue;
3430     }
3431     if( ch == '\n' || ch == '\r' ){
3432         fputc(ch,stderr);
3433         break;
3434     }
3435     line += string(ch);
3436     redraw(lno,line,right);
3437 }
3438 EXIT:
3439 system("/bin/stty echo sane");
3440 }
3441 //fprintf(stderr,"\r\nLINE:%s\r\n",line);
3442
3443 EXIT_GOT:
3444 Return line + right;
3445 }
3446
3447 func getline_main(){
3448     line := xgetline(0,"",nil)
3449     fprintf(stderr,"%s\n",line);
3450 /*
3451     dp = strpbrk(line,"\r\n");
3452     if( dp != NULL ){
3453         *dp = 0;
3454     }
3455
3456     if( 0 ){
3457         fprintf(stderr,"\n%d\n",int(strlen(line)));
3458     }
3459     if( lseek(3,0,0) == 0 ){
3460         if( romkanmode ){
3461             var buf [8*1024]byte;
3462             convs(line,buf);
3463             strcpy(line,buf);
3464         }
3465         write(3,line,strlen(line));
3466         ftruncate(3,lseek(3,0,SEEK_CUR));
3467         //fprintf(stderr,"outsize=%d\n",(int)lseek(3,0,SEEK_END));
3468         lseek(3,0,SEEK_SET);
3469         close(3);
3470     }else{
3471         fprintf(stderr,"\r\nGotline: ");
3472         trans(line);
3473         //printf("%s\n",line);
3474         printf("\n");
3475     }
3476 */
3477 }
3478 //== end ====== getline
3479
3480 //
3481 // $USERHOME/.gsh/
3482 //   gsh-rc.txt, or gsh-configure.txt
3483 //   gsh-history.txt
3484 //   gsh-aliases.txt // should be conditional?
3485 //
3486 func gshSetupHomedir(gshCtx GshContext) (GshContext, bool) {
3487     homedir,found := userHomeDir()
3488     if !found {
3489         fmt.Printf("--E-- You have no UserHomeDir\n")
3490         return gshCtx, true
3491     }
3492     gshhome := homedir + "/" + GSH_HOME
3493     _, err2 := os.Stat(gshhome)
3494     if err2 != nil {
3495         err3 := os.Mkdir(gshhome,0700)
3496         if err3 != nil {
3497             fmt.Printf("--E-- Could not Create %s (%s)\n",
3498                 gshhome,err3)
3499             return gshCtx, true

```

```

3500     }
3501     fmt.Printf("--I-- Created %s\n",gshhome)
3502 }
3503 gshCtx.GshHomeDir = gshhome
3504 return gshCtx, false
3505 }
3506 func setupGshContext()(GshContext,bool){
3507     gshPA := syscall.ProcAttr {
3508         "", // the starting directory
3509         os.Environ(), // environ[]
3510         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3511         nil, // OS specific
3512     }
3513     cwd, _ := os.Getwd()
3514     gshCtx := GshContext {
3515         cwd, // StartDir
3516         "", // GetLine
3517         []CChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
3518         gshPA,
3519         []CCommandHistory{}, //something for invokation?
3520         CCommandHistory{}, // CmdCurrent
3521         false,
3522         []int{},
3523         syscall.Rusage{},
3524         "", // GshHomeDir
3525         Ttyid(),
3526         false,
3527         false,
3528         []PluginInfo{},
3529         []string{},
3530         "",
3531         "v",
3532         ValueStack{},
3533         GServer{"",""}, // LastServer
3534     }
3535     err := false
3536     gshCtx, err = gshSetupHomedir(gshCtx)
3537     return gshCtx, err
3538 }
3539 // <a name="main">Main loop</a>
3540 func script(gshCtxGiven *GshContext) (_ GshContext) {
3541     gshCtx,err0 := setupGshContext()
3542     if err0 {
3543         return gshCtx;
3544     }
3545     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3546     //resmap()
3547
3548     /*
3549     if false {
3550         gsh_getlinev, with_exgetline :=
3551             which("PATH",[]string{"which","gsh-getline","-s"})
3552         if with_exgetline {
3553             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
3554             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
3555         }else{
3556             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
3557         }
3558     }
3559     */
3560
3561     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
3562     gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
3563
3564     prevline := ""
3565     skipping := false
3566     for hix := len(gshCtx.CommandHistory); ; {
3567         gline := getline(&gshCtx,hix,skipping,prevline)
3568         if skipping {
3569             if strings.Index(gline,"fi") == 0 {
3570                 fmt.Printf("fi\n");
3571                 skipping = false;
3572             }else{
3573                 //fmt.Printf("%s\n",gline);
3574             }
3575             continue
3576         }
3577         if strings.Index(gline,"if") == 0 {
3578             //fmt.Printf("--D-- if start: %s\n",gline);
3579             skipping = true;
3580             continue
3581         }
3582         if false {
3583             os.Stdout.Write([]byte("gotline:"))
3584             os.Stdout.Write([]byte(gline))
3585             os.Stdout.Write([]byte("\n"))
3586         }
3587         gline = strsubst(&gshCtx,gline,true)
3588         if false {
3589             fmt.Printf("fmt.Printf %v - %v\n",gline)
3590             fmt.Printf("fmt.Printf %s - %s\n",gline)
3591             fmt.Printf("fmt.Printf %x - %s\n",gline)
3592             fmt.Printf("fmt.Printf %U - %s\n",gline)
3593             fmt.Printf("Stoutt.Write -")
3594             os.Stdout.Write([]byte(gline))
3595             fmt.Printf("\n")
3596         }
3597         /*
3598         // should be cared in substitution ?
3599         if 0 < len(gline) && gline[0] == '!' {
3600             xgline, set, err := searchHistory(gshCtx,gline)
3601             if err {
3602                 continue
3603             }
3604             if set {
3605                 // set the line in command line editor
3606             }
3607             gline = xgline
3608         }
3609         */
3610         ghist := gshCtx.CmdCurrent
3611         ghist.WorkDir, _ = os.Getwd()
3612         ghist.WorkDirX = len(gshCtx.ChdirHistory)-1
3613         //fmt.Printf("--D--ChdirHistory(%#d)\n",len(gshCtx.ChdirHistory))
3614         ghist.StartAt = time.Now()
3615         rusagev1 := Getrusagev()
3616         gshCtx.CmdCurrent.FoundFile = []string{}
3617         xgshCtx, fin := tgshell1(gshCtx,gline)
3618         rusagev2 := Getrusagev()
3619         ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
3620         gshCtx = xgshCtx
3621         ghist.EndAt = time.Now()
3622         ghist.CmdLine = gline
3623         ghist.FoundFile = gshCtx.CmdCurrent.FoundFile
3624     }

```

```

3625     /* record it but not show in list by default
3626     if len(gline) == 0 {
3627         continue
3628     }
3629     if gline == "hi" || gline == "history" { // don't record it
3630         continue
3631     }
3632     */
3633     gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist)
3634     if fin {
3635         break;
3636     }
3637     prevline = gline;
3638     hix++;
3639 }
3640 return gshCtx
3641 }
3642 func main() {
3643     argv := os.Args
3644     if 1 < len(argv) {
3645         if isin("version", argv) {
3646             Version(nil, argv)
3647             return
3648         }
3649         comx := isinX("-c", argv)
3650         if 0 < comx {
3651             gshCtx, err := setupGshContext()
3652             if !err {
3653                 gshellv(gshCtx, argv[comx+1:])
3654             }
3655             return
3656         }
3657     }
3658     script(nil)
3659     //gshCtx := script(nil)
3660     //gshelll(gshCtx, "time")
3661 }
3662 //</div></details>
3663 //<details id="todo"><summary>Consideration</summary><div class="gsh-src">
3664 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
3665 // - merged histories of multiple parallel gsh sessions
3666 // - alias as a function or macro
3667 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
3668 // - retrieval PATH of files by its type
3669 // - gsh as an IME with completion using history and file names as dictionaries
3670 // - gsh a scheduler in precise time of within a millisecond
3671 // - all commands have its subcommand after "----" symbol
3672 // - filename expansion by "-find" command
3673 // - history of ext code and output of each command
3674 // - "script" output for each command by pty-tee or telnet-tee
3675 // - $BUILTLIN command in PATH to show the priority
3676 // - "?" symbol in the command (not as in arguments) shows help request
3677 // - searching command with wild card like: which ssh-*
3678 // - longformat prompt after long idle time (should dismiss by BS)
3679 // - customizing by building plugin and dynamically linking it
3680 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
3681 // - "!" symbol should be used for negation, don't waste it just for job control
3682 // - don't put too long output to tty, record it into GSH_HOME/session-id/command-id.log
3683 // - making canonical form of command at the start adding quotation or white spaces
3684 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
3685 // - name? or name! might be useful
3686 // - htar format - packing directory contents into a single html file using data scheme
3687 // - filepath substitution should be done by each command, especially in case of builtins
3688 // - @N substitution for the history of working directory, and @spec for more generic ones
3689 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
3690 // - GSH_PATH for plugins
3691 // - standard command output: list of data with name, size, resource usage, modified time
3692 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
3693 // -wc word-count, grep match line count, ...
3694 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
3695 // - -tailf-filename like tail -f filename, repeat close and open before read
3696 // - max. size and max. duration and timeout of (generated) data transfer
3697 // - auto. numbering, aliasing, IME completion of file name (especially rm of queer name)
3698 // - IME "?" at the top of the command line means searching history
3699 // - IME %d/0x10000/ %x/ffff/
3700 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
3701 // - gsh in WebAssembly
3702 // - gsh as a HTTP server of online-manual
3703 //---END--- (^-^)/ITS more</div></details>
3704 /*
3705 <details id="references"><summary>References</summary><div class="gsh-src">
3706 <p>
3707 <a href="https://golang.org">The Go Programming Language</a>
3708 <iframe src="https://golang.org" width="100%" height="300"></iframe>
3709 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
3710 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
3711 CSS:
3712 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors">Selectors</a>
3713 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
3714 HTTP
3715 JavaScript:
3716 ...
3717 </p>
3718 </div></details>
3719 <div id="gsh-footer" style="">Fin.</div>
3720 <style id="gsh-style">
3721 #gsh {border-width:1;margin:0;padding:0;}
3722 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
3723 #gsh header{height:100px;}
3724 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
3725 #gsh-menu{font-size:14pt;color:#f88;}
3726 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
3727 #gsh note{color:#000;font-size:10pt;}
3728 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
3729 #gsh details{color:#888;background-color:#aaa;font-family:monospace;}
3730 #gsh summary{font-size:16pt;color:#24a;background-color:#eef;height:30px;}
3731 #gsh pre{font-size:11pt;color:#223;background-color:#fafff;}
3732 #gsh a{color:#24a;}
3733 #gsh a[name]{color:#24a;font-size:16pt;}
3734 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
3735 #gsh .gsh-src{background-color:#fafff;color:#223;}
3736 #gsh-src-src{spellcheck:false}
3737 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
3738 #src-frame-textarea{background-color:#fafff;color:#223;}
3739 @media print {
3740 #gsh pre{font-size:11pt !important;}
3741 }
3742 </style>
3743 <!--
3744 // Logo image should be drawn by JavaScript from a meta-font.
3745 // CSS seems not follow line-splitted URL
3746 -->
3747 <script id="gsh-run">
3748 GshLogo="data:image/png;base64,\

```



```

3875
3876 document.getElementById('banner').style.backgroundImage="url("+GshLogo+")";
3877 //document.getElementById('gsh-footer').style.backgroundImage="url("+QR-ITS-more.jp.png+")";
3878 document.getElementById('gsh-footer').style.backgroundImage="url("+ITSmoreQR+")";
3879 //https://www.w3schools.com/JSREF/prop_style_backgroundposition.asp
3880 var bannerStop = false
3881 function shiftBG(){
3882   bannerStop = !bannerStop
3883   document.getElementById('banner').style.backgroundPosition = "0 0";
3884 }
3885
3886 function html_fold(){
3887   document.getElementById('index').open=false
3888   document.getElementById('gsh-gocode').open=false
3889   document.getElementById('todo').open=false
3890   document.getElementById('reference').open=false
3891 }
3892 function html_open(){
3893   document.getElementById('index').open=true
3894   document.getElementById('gsh-gocode').open=true
3895   document.getElementById('todo').open=true
3896   document.getElementById('reference').open=true
3897 }
3898 function html_stop(){
3899   bannerStop = !bannerStop
3900 }
3901
3902 //https://www.w3schools.com/jsref/met_win_setinterval.asp
3903 function shiftBanner(){
3904   var now = new Date().getTime();
3905   //console.log("now="+now%10)
3906   if( !bannerStop ){
3907     document.getElementById('banner').style.backgroundPosition = ((now/10)%100000)+" 0";
3908   }
3909 }
3910 setInterval(shiftBanner,10);
3911
3912 // from embedded html to standalone page
3913 function html_close(){
3914   window.close()
3915 }
3916
3917 // from embedded html to standalone page
3918 function html_new(){
3919   newwin = window.open("", "", "");
3920   src = document.getElementById("gsh");
3921   newwin.document.write("<"+html>\n");
3922   newwin.document.write("<"+span id="gsh">");
3923   newwin.document.write(src.innerHTML);
3924   newwin.document.write("<"+/span>"+/html>\n"); // gsh span
3925   newwin.document.close();
3926   newwin.focus();
3927 }
3928
3929 // source code viewr
3930 function frame_close(){
3931   srcframe = document.getElementById("src-frame");
3932   srcframe.innerHTML = "";
3933   //srcframe.style.cols = 1;
3934   srcframe.style.rows = 1;
3935   srcframe.style.height = 0;
3936   srcframe.style.display = false;
3937   src = document.getElementById("src-frame-textarea");
3938   src.innerHTML = ""
3939   //src.cols = 0
3940   src.rows = 0
3941   src.display = false
3942   //alert("--closed--")
3943 }
3944 //<!-- | <span onclick="html_view();">Source</span> -->
3945 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
3946 //<!--| <span>Download</span> -->
3947 function frame_open(){
3948   oldsrc = document.getElementById("GENSRC");
3949   if( oldsrc != null ){
3950     //alert("--I--(erasing old text)")
3951     oldsrc.innerHTML = "";
3952     return
3953   }else{
3954     //alert("--I--(no old text)")
3955   }
3956   banner = document.getElementById('banner').style.backgroundImage;
3957   footer = document.getElementById('gsh-footer').style.backgroundImage;
3958   document.getElementById('banner').style.backgroundImage = "";
3959   document.getElementById('banner').style.backgroundPosition = "";
3960   document.getElementById('gsh-footer').style.backgroundImage = "";
3961
3962   src = document.getElementById("gsh");
3963   srcframe = document.getElementById("src-frame");
3964   srcframe.innerHTML = ""
3965   + "<"+cite id="GENSRC">\n"
3966   + "<"+style>\n"
3967   + "#GENSRC textarea{tab-size:4;}\n"
3968   + "#GENSRC textarea(-o-tab-size:4;)\n"
3969   + "#GENSRC textarea(-moz-tab-size:4;)\n"
3970   + "#GENSRC textarea(spellcheck:false;)\n"
3971   + "<"+style>\n"
3972   + "<h2>\n"
3973   //+ "<"+span onclick="frame_close();\>Close</"+span>\n"
3974   //+ " | <"+span onclick="html_stop();\>Run</"+span>\n"
3975   + "</h2>\n"
3976   + "<"+textarea id="src-frame-textarea" cols=100 rows=40>"
3977   + /*<"+html>\n"
3978   + "<"+span id="gsh">"
3979   + src.innerHTML
3980   + "<"+/span>"+/html>\n"
3981   + "</"+textarea>\n"
3982   + "</"+cite><!-- GENSRC -->\n";
3983
3984   //srcframe.style.cols = 80;
3985   //srcframe.style.rows = 80;
3986
3987   document.getElementById('banner').style.backgroundImage = banner;
3988   document.getElementById('gsh-footer').style.backgroundImage = footer
3989 }
3990 function html_view(){
3991   html_stop();
3992
3993   banner = document.getElementById('banner').style.backgroundImage;
3994   footer = document.getElementById('gsh-footer').style.backgroundImage;
3995   document.getElementById('banner').style.backgroundImage = "";
3996   document.getElementById('banner').style.backgroundPosition = "";
3997   document.getElementById('gsh-footer').style.backgroundImage = "";
3998
3999   //srcwin = window.open("", "CodeView2", "");

```



```
4000 srcwin = window.open("", "", "");
4001 srcwin.document.write("<span id=\"gsh\">\n");
4002
4003 src = document.getElementById("gsh");
4004 srcwin.document.write("<style>\n");
4005 srcwin.document.write("textarea{tab-size:4;}\n");
4006 srcwin.document.write("textarea{-o-tab-size:4;}\n");
4007 srcwin.document.write("textarea{-moz-tab-size:4;}\n");
4008 srcwin.document.write("</style>\n");
4009 srcwin.document.write("<h2>\n");
4010 srcwin.document.write("<+span onclick=\"window.close();\n>Close</span> | \n");
4011 //srcwin.document.write("<+span onclick=\"html_stop();\n>Run</span>\n");
4012 srcwin.document.write("</h2>\n");
4013 srcwin.document.write("<textarea id=\"gsh-src-src\" cols=100 rows=60>");
4014 srcwin.document.write("/<+html>\n");
4015 srcwin.document.write("<+span id=\"gsh\">");
4016 srcwin.document.write(src.innerHTML);
4017 srcwin.document.write("<+span>+\"/html>\n");
4018 srcwin.document.write("</+textarea>\n");
4019
4020 document.getElementById('banner').style.backgroundImage = banner;
4021 document.getElementById('gsh-footer').style.backgroundImage = footer
4022
4023 sty = document.getElementById("gsh-style");
4024 srcwin.document.write("<+style>\n");
4025 srcwin.document.write(sty.innerHTML);
4026 srcwin.document.write("<+style>\n");
4027
4028 run = document.getElementById("gsh-run");
4029 srcwin.document.write("<+script>\n");
4030 srcwin.document.write(run.innerHTML);
4031 srcwin.document.write("<+script>\n");
4032
4033 srcwin.document.write("<+span>+\"/html>\n"); // gsh span
4034 srcwin.document.close();
4035 srcwin.focus();
4036 }
4037 </script>
4038 -->
4039 */ //</span></html>
4040
```