

```

1 //<html><details open><summary>GShell-0.2.4-HtmlArchive</summary>
2 /*<span id="gsh">
3 <link rel="icon" id="gsh-iconurl" href=""/><!-- place holder -->
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>GShell-0.2.4 by SatoxITS</title>
7 <header id="gsh-banner" height="100px" onclick="shiftBG();" style=""/>
8 <div align="right"><note>GShell version 0.2.4 // 2020-08-28 // 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(^-^)</note>
14 </note>
15 </p>
16 <span id="gsh-WinId" onclick="win_jump('0.1');">0</span>
17 <span id="gsh-menu">
18 | <span id="gsh-menu-exit" onclick="html_close();"></span>
19 | <span id="gsh-menu-fork" onclick="html_fork();">Fork</span>
20 | <span id="gsh-menu-stop" onclick="html_stop(this,true);">Stop</span>
21 | <span id="gsh-menu-fold" onclick="html_fold(this);">Unfold</span>
22 |<!-- |<span id="gsh-menu-pure" onclick="html_pure(this);">Pure</span> -->
23 |</span>
24 */
25 /*
26 <details id="gsh-statement" open><summary>Statement</summary><p id="gsh-statement">
27 <h2>Fun to create a shell</h2>
28 <p>For a programmer, it must be far easy and fun to create his own simple shell
29 rightly fitting to his favor and necessities, than learning existing shells with
30 complex full features that he never use.
31 I, as one of programmers, am writing this tiny shell for my own real needs,
32 totally from scratch, with fun.
33 </p><p>
34 For a programmer, it is fun to learn new computer languages. For long years before
35 writing this software, I had been specialized to C and early HTML2 :-).
36 Now writing this software, I'm learning Go language, HTML5, JavaScript and CSS
37 on demand as a novice of these, with fun.
38 </p><p>
39 This single file "gsh.go", that is executable by Go, contains all of the code written
40 in Go. Also it can be displayed as "gsh.go.html" by browsers. It is a standalone
41 HTML file that works as the viewer of the code of itself, and as the "home page" of
42 this software.
43 </p><p>
44 Because this HTML file is a Go program, you may run it as a real shell program
45 on your computer.
46 But you must be aware that this program is written under situation like above.
47 Needless to say, there is no warranty for this program in any means.
48 </p>
49 <address>Aug 2020, SatoxITS (sato@its-more.jp)</address>
50 </details>
51 */
52 /*
53 <details id="gsh-gindex" open>
54 <summary>Index</summary><div class="gsh-src">
55 Documents
56 <span class="gsh-link" onclick="jumpto_JavaScriptView();">Command summary</span>
57 Go lang part<span class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
58 Package structures
59 <a href="#import">import</a>
60 <a href="#struct">struct</a>
61 Main functions
62 <a href="#comexpansion">str-expansion</a> // macro processor
63 <a href="#finder">finder</a> // builtin find + du
64 <a href="#grep">grep</a> // builtin grep + wc + cksum + ...
65 <a href="#plugin">plugin</a> // plugin commands
66 <a href="#ex-commands">system</a> // external commands
67 <a href="#builtin">builtin</a> // builtin commands
68 <a href="#network">network</a> // socket handler
69 <a href="#remote-sh">remote-sh</a> // remote shell
70 <a href="#redirect">redirect</a> // StdIn/Out redirection
71 <a href="#history">history</a> // command history
72 <a href="#rusage">rusage</a> // resource usage
73 <a href="#encode">encode</a> // encode / decode
74 <a href="#IME">IME</a> // command line IME
75 <a href="#getline">getline</a> // line editor
76 <a href="#scanf">scanf</a> // string decomposer
77 <a href="#interpreter">interpreter</a> // command interpreter
78 <a href="#main">main</a>
79 </span>
80 JavaScript part
81 <a href="#script-src-view" class="gsh-link" onclick="jumpto_JavaScriptView();">Source</a>
82 <a href="#gsh-data-frame" class="gsh-link" onclick="jumpto_DataView();">Builtin data</a>
83 CSS part
84 <a href="#style-src-view" class="gsh-link" onclick="jumpto_StyleView();">Source</a>
85 References
86 <a href="#" class="gsh-link" onclick="jumpto_WholeView();">Internal</a>
87 <a href="#gsh-reference" class="gsh-link" onclick="jumpto_ReferenceView();">External</a>
88 Whole parts
89 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Source</a>
90 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Download</a>
91 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Dump</a>
92 </div>
93 </details>
94 </div>
95 */
96 //<details id="gsh-gocode">
97 //<summary>Go Source</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
98 // gsh - Go lang based Shell
99 // (c) 2020 ITS more Co., Ltd.
100 // 2020-0807 created by SatoxITS (sato@its-more.jp)
101
102 package main // gsh main
103 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
104 import (
105 "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
106 "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
107 "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
108 "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
109 "time" // <a href="https://golang.org/pkg/time/">time</a>
110 "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
111 "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
112 "os" // <a href="https://golang.org/pkg/os/">os</a>
113 "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
114 "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
115 "net" // <a href="https://golang.org/pkg/net/">net</a>
116 "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
117 "html" // <a href="https://golang.org/pkg/html/">html</a>
118 "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
119 "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
120 "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
121 "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
122 "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
123 // "gshdata" // gshell's logo and source code
124 "hash/crc32" // <a href="https://golang.org/pkg/hash/crc32/">crc32</a>

```

```

125 )
126 const (
127     NAME = "gsh"
128     VERSION = "0.2.4"
129     DATE = "2020-08-28"
130     AUTHOR = "SatoxITS(^-^)/"
131 )
132 var (
133     GSH_HOME = ".gsh" // under home directory
134     GSH_PORT = 9999
135     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
136     PROMPT = ">"
137     LINESIZE = (8*1024)
138     PATHSEP = ";" // should be ";" in Windows
139     DIRSEP = "/" // canbe \ in Windows
140 )
141
142 // -x logging control
143 // --A-- all
144 // --I-- info.
145 // --D-- debug
146 // --T-- time and resource usage
147 // --W-- warning
148 // --E-- error
149 // --F-- fatal error
150 // --Xn- network
151
152 // <a name="struct">Structures</a>
153 type GCommandHistory struct {
154     StartAt time.Time // command line execution started at
155     EndAt time.Time // command line execution ended at
156     ResCode int // exit code of (external command)
157     CmdError error // error string
158     OutData *os.File // output of the command
159     FoundFile []string // output - result of unfid
160     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
161     CmdId int // maybe with identified with arguments or impact
162     // redirection commands should not be the CmdId
163     WorkDir string // working directory at start
164     WorkDirX int // index in ChdirHistory
165     CmdLine string // command line
166 }
167 type GChdirHistory struct {
168     Dir string
169     MovedAt time.Time
170     CmdIndex int
171 }
172 type CmdMode struct {
173     Background bool
174 }
175 type PluginInfo struct {
176     Spec *plugin.Plugin
177     Addr plugin.Symbol
178     Name string // maybe relative
179     Path string // this is in Plugin but hidden
180 }
181 type GServer struct {
182     host string
183     port string
184 }
185
186 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
187 const ( // SumType
188     SUM_ITEMS = 0x000001 // items count
189     SUM_SIZE = 0x000002 // data length (simply added)
190     SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
191     SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
192     // also envelope attributes like time stamp can be a part of digest
193     // hashed value of sizes or mod-date of files will be useful to detect changes
194
195     SUM_WORDS = 0x000010 // word count is a kind of digest
196     SUM_LINES = 0x000020 // line count is a kind of digest
197     SUM_SUM64 = 0x000040 // simple add of bytes, useful for human too
198
199     SUM_SUM32_BITS = 0x000100 // the number of true bits
200     SUM_SUM32_2BYTE = 0x000200 // 16bits words
201     SUM_SUM32_4BYTE = 0x000400 // 32bits words
202     SUM_SUM32_8BYTE = 0x000800 // 64bits words
203
204     SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
205     SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
206     SUM_UNIXFILE = 0x004000
207     SUM_CRCIEEE = 0x008000
208 )
209 type CheckSum struct {
210     Files int64 // the number of files (or data)
211     Size int64 // content size
212     Words int64 // word count
213     Lines int64 // line count
214     SumType int
215     Sum64 uint64
216     Crc32Table crc32.Table
217     Crc32Val uint32
218     Sum16 int
219     Ctime time.Time
220     Atime time.Time
221     Mtime time.Time
222     Start time.Time
223     Done time.Time
224     RusgAtStart [2]syscall.Rusage
225     RusgAtEnd [2]syscall.Rusage
226 }
227 type ValueStack [][]string
228 type GshContext struct {
229     StartDir string // the current directory at the start
230     GetLine string // gsh-getline command as a input line editor
231     ChdirHistory []GChdirHistory // the 1st entry is wd at the start
232     gshPA syscall.ProcAttr
233     CommandHistory []GCommandHistory
234     CmdCurrent GCommandHistory
235     Background bool
236     BackgroundJobs []int
237     LastRusage syscall.Rusage
238     GshHomeDir string
239     TerminalId int
240     CmdTrace bool // should be [map]
241     CmdTime bool // should be [map]
242     PluginFuncs []PluginInfo
243     ivalues []string
244     iDelimiter string // field separator of print out
245     iFormat string // default print format (of integer)
246     iValStack ValueStack
247     LastServer GServer
248     RSERV string // [gsh://]host[:port]
249     RWD string // remote (target, there) working directory

```

```

250     lastChecksum    CheckSum
251 }
252
253 func nsleep(ns time.Duration){
254     time.Sleep(ns)
255 }
256 func usleep(ns time.Duration){
257     nsleep(ns*1000)
258 }
259 func msleep(ns time.Duration){
260     nsleep(ns*1000000)
261 }
262 func sleep(ns time.Duration){
263     nsleep(ns*1000000000)
264 }
265
266 func strBegins(str, pat string)(bool){
267     if len(pat) <= len(str){
268         yes := str[0:len(pat)] == pat
269         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat, yes)
270         return yes
271     }
272     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
273     return false
274 }
275 func isin(what string, list []string) bool {
276     for _, v := range list {
277         if v == what {
278             return true
279         }
280     }
281     return false
282 }
283 func isinX(what string,list[]string)(int){
284     for i,v := range list {
285         if v == what {
286             return i
287         }
288     }
289     return -1
290 }
291
292 func env(opts []string) {
293     env := os.Environ()
294     if isin("--s", opts){
295         sort.Slice(env, func(i,j int) bool {
296             return env[i] < env[j]
297         })
298     }
299     for _, v := range env {
300         fmt.Printf("%v\n",v)
301     }
302 }
303
304 // - rewriting should be context dependent
305 // - should postpone until the real point of evaluation
306 // - should rewrite only known notation of symbol
307 func scanInt(str string)(val int,leng int){
308     leng = -1
309     for i,ch := range str {
310         if '0' <= ch && ch <= '9' {
311             leng = i+1
312         }else{
313             break
314         }
315     }
316     if 0 < leng {
317         ival,_ := strconv.Atoi(str[0:leng])
318         return ival,leng
319     }else{
320         return 0,0
321     }
322 }
323 func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
324     if len(str[i+1:]) == 0 {
325         return 0,rstr
326     }
327     hi := 0
328     histlen := len(gshCtx.CommandHistory)
329     if str[i+1] == '!' {
330         hi = histlen - 1
331         leng = 1
332     }else{
333         hi,leng = scanInt(str[i+1:])
334         if leng == 0 {
335             return 0,rstr
336         }
337         if hi < 0 {
338             hi = histlen + hi
339         }
340     }
341     if 0 <= hi && hi < histlen {
342         var ext byte
343         if 1 < len(str[i+leng:]) {
344             ext = str[i+leng:][1]
345         }
346         //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
347         if ext == 'f' {
348             leng += 1
349             xlist := []string{}
350             list := gshCtx.CommandHistory[hi].FoundFile
351             for _,v := range list {
352                 //list[i] = escapeWhiteSP(v)
353                 xlist = append(xlist,escapeWhiteSP(v))
354             }
355             //rstr += strings.Join(list," ")
356             rstr += strings.Join(xlist," ")
357         }else{
358             if ext == 'e' || ext == 'd' {
359                 // !N0 .. workdir at the start of the command
360                 leng += 1
361                 rstr += gshCtx.CommandHistory[hi].WorkDir
362             }else{
363                 rstr += gshCtx.CommandHistory[hi].CmdLine
364             }
365         }else{
366             leng = 0
367         }
368         return leng,rstr
369     }
370 func escapeWhiteSP(str string)(string){
371     if len(str) == 0 {
372         return "\\z" // empty, to be ignored
373     }
374     rstr := ""

```

```

375 for _,ch := range str {
376     switch ch {
377         case '\\': rstr += "\\\"
378         case ' ': rstr += "\s"
379         case '\t': rstr += "\t"
380         case '\r': rstr += "\r"
381         case '\n': rstr += "\n"
382         default: rstr += string(ch)
383     }
384 }
385 return rstr
386 }
387 func unescapeWhiteSP(str string)(string){ // strip original escapes
388     rstr := ""
389     for i := 0; i < len(str); i++ {
390         ch := str[i]
391         if ch == '\\' {
392             if i+1 < len(str) {
393                 switch str[i+1] {
394                     case 'z':
395                         continue;
396                 }
397             }
398         }
399         rstr += string(ch)
400     }
401     return rstr
402 }
403 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
404     ustrv := []string{}
405     for _,v := range strv {
406         ustrv = append(ustrv,unescapeWhiteSP(v))
407     }
408     return ustrv
409 }
410
411 // <a name="comexpansion">str-expansion</a>
412 // - this should be a macro processor
413 func strsubst(gshCtx *GshContext,str string,histonly bool) string {
414     rbuff := []byte{}
415     if false {
416         //@@@U Unicode should be cared as a character
417         return str
418     }
419     //rstr := ""
420     inEsc := 0 // escape characer mode
421     for i := 0; i < len(str); i++ {
422         //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
423         ch := str[i]
424         if inEsc == 0 {
425             if ch == '|' {
426                 //leng,xrstr := substHistory(gshCtx,str,i,rstr)
427                 leng,rs := substHistory(gshCtx,str,i,"")
428                 if 0 < leng {
429                     //_,rs := substHistory(gshCtx,str,i,"")
430                     rbuff = append(rbuff,[]byte(rs)...)
431                     i += leng
432                     //rstr = xrstr
433                     continue
434                 }
435             }
436             switch ch {
437                 case '\\': inEsc = '\\'; continue
438                 //case '%': inEsc = '%'; continue
439                 case '$':
440             }
441         }
442         switch inEsc {
443         case '\\':
444             switch ch {
445                 case '\\': ch = '\\'
446                 case 's': ch = ' '
447                 case 't': ch = '\t'
448                 case 'r': ch = '\r'
449                 case 'n': ch = '\n'
450                 case 'z': inEsc = 0; continue // empty, to be ignored
451             }
452             inEsc = 0
453         case '%':
454             switch {
455                 case ch == '%': ch = '%'
456                 case ch == 'm':
457                     //rstr = rstr + time.Now().Format(time.Stamp)
458                     rs := time.Now().Format(time.Stamp)
459                     rbuff = append(rbuff,[]byte(rs)...)
460                     inEsc = 0
461                     continue;
462                 default:
463                     // postpone the interpretation
464                     //rstr = rstr + "%" + string(ch)
465                     rbuff = append(rbuff,ch)
466                     inEsc = 0
467                     continue;
468             }
469             inEsc = 0
470         }
471         //rstr = rstr + string(ch)
472         rbuff = append(rbuff,ch)
473     }
474     //fmt.Printf("--D--subst(%s)(%s)\n",str,string(rbuff))
475     return string(rbuff)
476     //return rstr
477 }
478 func showFileInfo(path string, opts []string) {
479     if isin("-l",opts) || isin("-ls",opts) {
480         fi, err := os.Stat(path)
481         if err != nil {
482             fmt.Printf("----- ((%v))",err)
483         }else{
484             mod := fi.ModTime()
485             date := mod.Format(time.Stamp)
486             fmt.Printf("%v %v %s ",fi.Mode(),fi.Size(),date)
487         }
488     }
489     fmt.Printf("%s",path)
490     if isin("-sp",opts) {
491         fmt.Printf(" ")
492     }else
493     if ! isin("-n",opts) {
494         fmt.Printf("\n")
495     }
496 }
497 func userHomeDir()(string,bool){
498     /*
499     homedir,_ = os.UserHomeDir() // not implemented in older Golang

```

```

500 */
501 homedir,found := os.LookupEnv("HOME")
502 //fmt.Printf("--I-- HOME=%v\n",homedir,found)
503 if !found {
504     return "/tmp",found
505 }
506 return homedir,found
507 }
508
509 func toFullpath(path string) (fullpath string) {
510     if path[0] == '/' {
511         return path
512     }
513     pathv := strings.Split(path,DIRSEP)
514     switch {
515     case pathv[0] == ".":
516         pathv[0],_ = os.Getwd()
517     case pathv[0] == "..": // all ones should be interpreted
518         cwd,_ = os.Getwd()
519         ppathv := strings.Split(cwd,DIRSEP)
520         pathv[0] = strings.Join(ppathv,DIRSEP)
521     case pathv[0] == "-":
522         pathv[0],_ = userHomeDir()
523     default:
524         cwd,_ = os.Getwd()
525         pathv[0] = cwd + DIRSEP + pathv[0]
526     }
527     return strings.Join(pathv,DIRSEP)
528 }
529
530 func IsRegFile(path string)(bool){
531     fi, err := os.Stat(path)
532     if err == nil {
533         fm := fi.Mode()
534         return fm.IsRegular();
535     }
536     return false
537 }
538
539 // <a name="encode">Encode / Decode</a>
540 // <a href="https://golang.org/pkg/encoding/base64/#example_NewEncoder">Encoder</a>
541 func (gshCtx *GshContext)Enc(argv[]string){
542     file := os.Stdin
543     buff := make([]byte,LINESIZE)
544     li := 0
545     encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
546     for li = 0; li++ {
547         count, err := file.Read(buff)
548         if count <= 0 {
549             break
550         }
551         if err != nil {
552             break
553         }
554         encoder.Write(buff[0:count])
555     }
556     encoder.Close()
557 }
558 func (gshCtx *GshContext)Dec(argv[]string){
559     decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
560     li := 0
561     buff := make([]byte,LINESIZE)
562     for li = 0; li++ {
563         count, err := decoder.Read(buff)
564         if count <= 0 {
565             break
566         }
567         if err != nil {
568             break
569         }
570         os.Stdout.Write(buff[0:count])
571     }
572 }
573 // lnspp [N] [-crlf][-C \\\]
574 func (gshCtx *GshContext)SplitLine(argv[]string){
575     reader := bufio.NewReaderSize(os.Stdin,64*1024)
576     ni := 0
577     toi := 0
578     for ni = 0; ni++ {
579         line, err := reader.ReadString('\n')
580         if len(line) <= 0 {
581             if err != nil {
582                 fmt.Fprintf(os.Stderr,"--I-- lnspp %d to %d (%v)\n",ni,toi,err)
583                 break
584             }
585         }
586         off := 0
587         ilen := len(line)
588         remlen := len(line)
589         for oi := 0; oi < remlen; oi++ {
590             olen := remlen
591             addnl := false
592             if 72 < olen {
593                 olen = 72
594                 addnl = true
595             }
596             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
597                 toi,ni,oi,off,olen,remlen,ilen)
598             toi += 1
599             os.Stdout.Write([]byte(line[0:olen]))
600             if addnl {
601                 //os.Stdout.Write([]byte("\r\n"))
602                 os.Stdout.Write([]byte("\\"))
603                 os.Stdout.Write([]byte("\n"))
604             }
605             line = line[olen:]
606             off += olen
607             remlen -= olen
608         }
609     }
610     fmt.Fprintf(os.Stderr,"--I-- lnspp %d to %d\n",ni,toi)
611 }
612
613 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
614 // 1 0000 0100 1100 0001 0001 1101 1011 0111
615 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
616 var CRC32IEEE uint32 = uint32(0xEDB88320)
617 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
618     var i uint64
619     for i = 0; i < len; i++ {
620         var oct = str[i]
621         for bi := 0; bi < 8; bi++ {
622             ovf1 := (crc & 0x80000000) != 0
623             ovf2 := (oct & 0x80) != 0
624             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)

```

```

625     oct <<= 1
626     crc <<= 1
627     if ovf { crc ^= CRC32UNIX }
628 }
629 }
630 return crc;
631 }
632 func byteCRC32end(crc uint32, len uint64)(uint32){
633     var slen = make([]byte,4)
634     var li = 0
635     for li = 0; li < 4; {
636         slen[li] = byte(len)
637         li += 1
638         len >>= 8
639         if( len == 0 ){
640             break
641         }
642     }
643     crc = byteCRC32add(crc,slen,uint64(li))
644     crc ^= 0xFFFFFFFF
645     return crc
646 }
647 func byteCRC32(str[]byte, len uint64)(crc uint32){
648     crc = byteCRC32add(0,str,len)
649     crc = byteCRC32end(crc,len)
650     return crc
651 }
652 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
653     var slen = make([]byte,4)
654     var li = 0
655     for li = 0; li < 4; {
656         slen[li] = byte(len & 0xFF)
657         li += 1
658         len >>= 8
659         if( len == 0 ){
660             break
661         }
662     }
663     crc = crc32.Update(crc,table,slen)
664     crc ^= 0xFFFFFFFF
665     return crc
666 }
667 }
668 func (gsh*GshContext)xChecksum(path string,argv[]string, sum*Checksum)(int64){
669     if isin("-type/f",argv) && !IsRegFile(path){
670         return 0
671     }
672     if isin("-type/d",argv) && IsRegFile(path){
673         return 0
674     }
675     file, err := os.OpenFile(path,os.O_RDONLY,0)
676     if err != nil {
677         fmt.Printf("--E-- cksum %v (%v)\n",path,err)
678         return -1
679     }
680     defer file.Close()
681     if gsh.CmdTrace { fmt.Printf("--I-- cksum %v %v\n",path,argv) }
682 }
683 bi := 0
684 var buff = make([]byte,32*1024)
685 var total int64 = 0
686 var initTime = time.Time{}
687 if sum.Start == initTime {
688     sum.Start = time.Now()
689 }
690 for bi = 0; ; bi++ {
691     count,err := file.Read(buff)
692     if count <= 0 || err != nil {
693         break
694     }
695     if (sum.SumType & SUM_SUM64) != 0 {
696         s := sum.Sum64
697         for _,c := range buff[0:count] {
698             s += uint64(c)
699         }
700         sum.Sum64 = s
701     }
702     if (sum.SumType & SUM_UNIXFILE) != 0 {
703         sum.Crc32Val = byteCRC32add(sum.Crc32Val,buff,uint64(count))
704     }
705     if (sum.SumType & SUM_CRCIEEE) != 0 {
706         sum.Crc32Val = crc32.Update(sum.Crc32Val,&sum.Crc32Table,buff[0:count])
707     }
708     // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
709     if (sum.SumType & SUM_SUM16_BSD) != 0 {
710         s := sum.Sum16
711         for _,c := range buff[0:count] {
712             s = (s >> 1) + ((s & 1) << 15)
713             s += int(c)
714             s &= 0xFFFF
715             //fmt.Printf("BSDsum: %d[%d] %d\n",sum.Size+int64(i),i,s)
716         }
717         sum.Sum16 = s
718     }
719     if (sum.SumType & SUM_SUM16_SYSV) != 0 {
720         for bj := 0; bj < count; bj++ {
721             sum.Sum16 += int(buff[bj])
722         }
723     }
724     total += int64(count)
725 }
726 sum.Done = time.Now()
727 sum.Files += 1
728 sum.Size += total
729 if !isin("-s",argv) {
730     fmt.Printf("%v ",total)
731 }
732 return 0
733 }
734 }
735 // <a name="grep">grep</a>
736 // "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
737 // a*,lab,c, ... sequential combination of patterns
738 // what "LINE" is should be definable
739 // generic line-by-line processing
740 // grep [-v]
741 // cat -n -v
742 // uniq [-c]
743 // tail -f
744 // sed s/x/y/ or awk
745 // grep with line count like wc
746 // rewrite contents if specified
747 func (gsh*GshContext)xGrep(path string,rexpv[]string)(int){
748     file, err := os.OpenFile(path,os.O_RDONLY,0)
749     if err != nil {

```

```

750     fmt.Printf("--E-- grep %v (%v)\n",path,err)
751     return -1
752 }
753 defer file.Close()
754 if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path,rxpv) }
755 //reader := bufio.NewReaderSize(file,LINESIZE)
756 reader := bufio.NewReaderSize(file,80)
757 li := 0
758 found := 0
759 for li = 0; ; li++ {
760     line, err := reader.ReadString('\n')
761     if len(line) <= 0 {
762         break
763     }
764     if 150 < len(line) {
765         // maybe binary
766         break;
767     }
768     if err != nil {
769         break
770     }
771     if 0 <= strings.Index(string(line),rxpv[0]) {
772         found += 1
773         fmt.Printf("%s:%d: %s",path,li,line)
774     }
775 }
776 //fmt.Printf("total %d lines %s\n",li,path)
777 //if( 0 < found ){ fmt.Printf("((found %d lines %s)\n",found,path); }
778 return found
779 }
780
781 // <a name="finder">Finder</a>
782 // finding files with it name and contents
783 // file names are Ored
784 // show the content with %x fmt list
785 // ls -R
786 // tar command by adding output
787 type fileSum struct {
788     Err int64 // access error or so
789     Size int64 // content size
790     DupSize int64 // content size from hard links
791     Blocks int64 // number of blocks (of 512 bytes)
792     DupBlocks int64 // Blocks pointed from hard links
793     HLinks int64 // hard links
794     Words int64
795     Lines int64
796     Files int64
797     Dirs int64 // the num. of directories
798     SymLink int64
799     Flats int64 // the num. of flat files
800     MaxDepth int64
801     MaxNamlen int64 // max. name length
802     nextRepo time.Time
803 }
804 func showFusage(dir string,fusage *fileSum){
805     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
806     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
807
808     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
809         dir,
810         fusage.Files,
811         fusage.Dirs,
812         fusage.SymLink,
813         fusage.HLinks,
814         float64(fusage.Size)/1000000.0,bsume);
815 }
816 const (
817     S_IFMT = 0170000
818     S_IFCHR = 0020000
819     S_IFDIR = 0040000
820     S_IFREG = 0100000
821     S_IFLNK = 0120000
822     S_IFSOCK = 0140000
823 )
824 func cumPinfo(fsum *fileSum, path string, stater error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
825     now := time.Now()
826     if time.Second <= now.Sub(fsum.nextRepo) {
827         if !fsum.nextRepo.IsZero(){
828             tstamp := now.Format(time.Stamp)
829             showFusage(tstamp, fsum)
830         }
831         fsum.nextRepo = now.Add(time.Second)
832     }
833     if stater != nil {
834         fsum.Err += 1
835         return fsum
836     }
837     fsum.Files += 1
838     if 1 <= fstat.Nlink {
839         // must count only once...
840         // at least ignore ones in the same directory
841         //if finfo.Mode().IsRegular() {
842             if (fstat.Mode & S_IFMT) == S_IFREG {
843                 fsum.HLinks += 1
844                 fsum.DupBlocks += int64(fstat.Blocks)
845                 //fmt.Printf("---Dup HardLink %v %s\n",fstat.Nlink,path)
846             }
847         }
848         //fsum.Size += finfo.Size()
849         fsum.Size += fstat.Size
850         fsum.Blocks += int64(fstat.Blocks)
851         //if verb { fmt.Printf("(%8dBlk) %s",fstat.Blocks/2,path) }
852         if isin("-ls",argv){
853             //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }
854             // fmt.Printf("%d\t",fstat.Blocks/2)
855         }
856         //if finfo.IsDir()
857         if (fstat.Mode & S_IFMT) == S_IFDIR {
858             fsum.Dirs += 1
859         }
860         //if (finfo.Mode() & os.ModeSymlink) != 0
861         if (fstat.Mode & S_IFMT) == S_IFLNK {
862             //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }
863             //{ fmt.Printf("symlink(%o,%s)\n",fstat.Mode,finfo.Name()) }
864             fsum.SymLink += 1
865         }
866     }
867     return fsum
868 }
869 func (gsh*GshContext)xxFindEntv(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(*fileSum){
870     nols := isin("-grep",argv)
871     // sort entv
872     /*
873     if isin("-t",argv){
874         sort.Slice(filev, func(i,j int) bool {
875             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
876         })
877     }
878     */

```

```

875     })
876 }
877 */
878 /*
879 if isin("-u",argv){
880     sort.Slice(filev, func(i,j int) bool {
881         return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
882     })
883 }
884 if isin("-U",argv){
885     sort.Slice(filev, func(i,j int) bool {
886         return 0 < filev[i].CreateTime().Sub(filev[j].CreateTime())
887     })
888 }
889 */
890 /*
891 if isin("-S",argv){
892     sort.Slice(filev, func(i,j int) bool {
893         return filev[j].Size() < filev[i].Size()
894     })
895 }
896 */
897 for _,filename := range entv {
898     for _,npat := range npatv {
899         match := true
900         if npat == "*" {
901             match = true
902         }else{
903             match, _ = filepath.Match(npat,filename)
904         }
905         path := dir + DIRSEP + filename
906         if !match {
907             continue
908         }
909         var fstat syscall.Stat_t
910         staterr := syscall.Lstat(path,&fstat)
911         if staterr != nil {
912             if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
913             continue;
914         }
915         if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
916             // should not show size of directory in "-du" mode ...
917         }else{
918             if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
919                 if isin("-du",argv) {
920                     fmt.Printf("%d\t",fstat.Blocks/2)
921                 }
922                 showFileInfo(path,argv)
923             }
924             if true { // && isin("-du",argv)
925                 total = cumFinfo(total,path,staterr,fstat,argv,false)
926             }
927             /*
928             if isin("-wc",argv) {
929             }
930             */
931             if gsh.lastCheckSum.SumType != 0 {
932                 gsh.xCksum(path,argv,&gsh.lastCheckSum);
933             }
934             x := isinX("-grep",argv); // -grep will be convenient like -ls
935             if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls
936                 if IsRegFile(path){
937                     found := gsh.xGrep(path,argv[x+1:])
938                     if 0 < found {
939                         foundv := gsh.CmdCurrent.FoundFile
940                         if len(foundv) < 10 {
941                             gsh.CmdCurrent.FoundFile =
942                                 append(gsh.CmdCurrent.FoundFile,path)
943                         }
944                     }
945                 }
946             }
947             if !isin("-r0",argv) { // -d 0 in du, -depth n in find
948                 //total.Depth += 1
949                 if (fstat.Mode & S_IFMT) == S_IFLNK {
950                     continue
951                 }
952                 if dstat.Rdev != fstat.Rdev {
953                     fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
954                         dir,dstat.Rdev,path,fstat.Rdev)
955                 }
956                 if (fstat.Mode & S_IFMT) == S_IFDIR {
957                     total = gsh.xxFind(depth+1,total,path,npatv,argv)
958                 }
959             }
960         }
961     }
962     return total
963 }
964 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
965     nols := isin("-grep",argv)
966     dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
967     if oerr == nil {
968         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
969         defer dirfile.Close()
970     }else{
971     }
972 }
973 prev := *total
974 var dstat syscall.Stat_t
975 staterr := syscall.Lstat(dir,&dstat) // should be flstat
976
977 if staterr != nil {
978     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
979     return total
980 }
981 //filev,err := ioutil.ReadDir(dir)
982 //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
983 /*
984 if err != nil {
985     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
986     return total
987 }
988 */
989 if depth == 0 {
990     total = cumFinfo(total,dir,staterr,dstat,argv,true)
991     if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
992         showFileInfo(dir,argv)
993     }
994 }
995 // it is not a directory, just scan it and finish
996
997 for ei := 0; ; ei++ {
998     entv,rderr := dirfile.Readdirnames(8*1024)
999     if len(entv) == 0 || rderr != nil {

```



```

1000 //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
1001 break
1002 }
1003 if 0 < ei {
1004     fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
1005 }
1006 total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npats,argv)
1007 }
1008 if isin("-du",argv) {
1009     // if in "du" mode
1010     fmt.Printf("%d\t%s\n",(total.Blocks-prev.Blocks)/2,dir)
1011 }
1012 return total
1013 }
1014 }
1015 // {ufind|fu|ls} [Files] [-- Expressions]
1016 // Files is "." by default
1017 // Names is "*" by default
1018 // Expressions is "-print" by default for "ufind", or -du for "fu" command
1019 func (gsh*GshContext)xFind(argv[]string){
1020     if 0 < len(argv) && strBegins(argv[0],"?"){
1021         showFound(gsh,argv)
1022         return
1023     }
1024     if isin("-cksum",argv) || isin("-sum",argv) {
1025         gsh.lastCheckSum = CheckSum{}
1026         if isin("-sum",argv) && isin("-add",argv) {
1027             gsh.lastCheckSum.SumType |= SUM_SUM64
1028         }else
1029         if isin("-sum",argv) && isin("-size",argv) {
1030             gsh.lastCheckSum.SumType |= SUM_SIZE
1031         }else
1032         if isin("-sum",argv) && isin("-bsd",argv) {
1033             gsh.lastCheckSum.SumType |= SUM_SUM16_BSD
1034         }else
1035         if isin("-sum",argv) && isin("-sysv",argv) {
1036             gsh.lastCheckSum.SumType |= SUM_SUM16_SYSV
1037         }else
1038         if isin("-sum",argv) {
1039             gsh.lastCheckSum.SumType |= SUM_SUM64
1040         }
1041         if isin("-unix",argv) {
1042             gsh.lastCheckSum.SumType |= SUM_UNIXFILE
1043             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32UNIX)
1044         }
1045         if isin("-ieee",argv){
1046             gsh.lastCheckSum.SumType |= SUM_CRCIEEE
1047             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32IEEE)
1048         }
1049         gsh.lastCheckSum.RusgAtStart = Getrusagev()
1050     }
1051     var total = fileSum{}
1052     npats := []string{}
1053     for _,v := range argv {
1054         if 0 < len(v) && v[0] != '-' {
1055             npats = append(npats,v)
1056         }
1057         if v == "/" { break }
1058         if v == "--" { break }
1059         if v == "-grep" { break }
1060         if v == "-ls" { break }
1061     }
1062     if len(npats) == 0 {
1063         npats = []string{"*"}
1064     }
1065     cwd := "."
1066     // if to be fullpath :: cwd, _ := os.Getwd()
1067     if len(npats) == 0 { npats = []string{"*"} }
1068     fusage := gsh.xxFind(0,total,cwd,npats,argv)
1069     if gsh.lastCheckSum.SumType != 0 {
1070         var sumi uint64 = 0
1071         sum := gsh.lastCheckSum
1072         if (sum.SumType & SUM_SIZE) != 0 {
1073             sumi = uint64(sum.Size)
1074         }
1075         if (sum.SumType & SUM_SUM64) != 0 {
1076             sumi = sum.Sum64
1077         }
1078         if (sum.SumType & SUM_SUM16_SYSV) != 0 {
1079             s := uint32(sum.Sum16)
1080             r := (s & 0xFFFF) + ((s & 0xFFFFFFFF) >> 16)
1081             s = (r & 0xFFFF) + (r >> 16)
1082             sum.Crc32Val = uint32(s)
1083             sumi = uint64(s)
1084         }
1085         if (sum.SumType & SUM_SUM16_BSD) != 0 {
1086             sum.Crc32Val = uint32(sum.Sum16)
1087             sumi = uint64(sum.Sum16)
1088         }
1089         if (sum.SumType & SUM_UNIXFILE) != 0 {
1090             sum.Crc32Val = byteCRC32end(sum.Crc32Val,uint64(sum.Size))
1091             sumi = uint64(byteCRC32end(sum.Crc32Val,uint64(sum.Size)))
1092         }
1093         if 1 < sum.Files {
1094             fmt.Printf("%v %v // %v / %v files, %v/file\r\n",
1095                 sumi,sum.Size,
1096                 abssize(sum.Size),sum.Files,
1097                 abssize(sum.Size/sum.Files))
1098         }else{
1099             fmt.Printf("%v %v %v\n",
1100                 sumi,sum.Size,npats[0])
1101         }
1102     }
1103     if !isin("-grep",argv) {
1104         showFusage("total",fusage)
1105     }
1106     if !isin("-s",argv){
1107         hits := len(gsh.CmdCurrent.FoundFile)
1108         if 0 < hits {
1109             fmt.Printf("--I-- %d files hits // can be refered with !%df\n",
1110                 hits,len(gsh.CommandHistory))
1111         }
1112     }
1113     if gsh.lastCheckSum.SumType != 0 {
1114         if isin("-ru",argv) {
1115             sum := gsh.lastCheckSum
1116             sum.Done = time.Now()
1117             gsh.lastCheckSum.RusgAtEnd = Getrusagev()
1118             elps := sum.Done.Sub(sum.Start)
1119             fmt.Printf("--cksum-size: %v (%v) / %v files, %v/file\r\n",
1120                 sum.Size,abssize(sum.Size),sum.Files,abssize(sum.Size/sum.Files))
1121             nanos := int64(elps)
1122             fmt.Printf("--cksum-time: %v/total, %v/file, %.1f files/s, %v\r\n",
1123                 abstime(nanos),
1124                 abstime(nanos/sum.Files),

```

```

1125         (float64(sum.Files)*1000000000.0)/float64(nanos),
1126         abbspeed(sum.Size,nanos))
1127     diff := RusageSubv(sum.RusgAtEnd,sum.RusgAtStart)
1128     fmt.Printf("--cksum-rusg: %v\n",sRusagef("",argv,diff))
1129 }
1130 }
1131 return
1132 }
1133 }
1134 func showFiles(files[]string){
1135     sp := ""
1136     for i,file := range files {
1137         if 0 < i { sp = " " } else { sp = "" }
1138         fmt.Printf(sp+"%s",escapeWhiteSP(file))
1139     }
1140 }
1141 func showFound(gshCtx *GshContext, argv[]string){
1142     for i,v := range gshCtx.CommandHistory {
1143         if 0 < len(v.FoundFile) {
1144             fmt.Printf("%d %d ",i,len(v.FoundFile))
1145             if isin("-ls",argv){
1146                 fmt.Printf("\n")
1147                 for _,file := range v.FoundFile {
1148                     fmt.Printf("%s\n") //sub number?
1149                     showFileInfo(file,argv)
1150                 }
1151             }else{
1152                 showFiles(v.FoundFile)
1153                 fmt.Printf("\n")
1154             }
1155         }
1156     }
1157 }
1158 }
1159 func showMatchFile(filev []os.FileInfo, npat,dir string, argv[]string)(string,bool){
1160     fname := ""
1161     found := false
1162     for _,v := range filev {
1163         match, _ := filepath.Match(npat,(v.Name()))
1164         if match {
1165             fname = v.Name()
1166             found = true
1167             //fmt.Printf("[%d] %s\n",i,v.Name())
1168             showIfExecutable(fname,dir,argv)
1169         }
1170     }
1171     return fname,found
1172 }
1173 func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
1174     var fullpath string
1175     if strBegins(name,DIRSEP){
1176         fullpath = name
1177     }else{
1178         fullpath = dir + DIRSEP + name
1179     }
1180     fi, err := os.Stat(fullpath)
1181     if err != nil {
1182         fullpath = dir + DIRSEP + name + ".go"
1183         fi, err = os.Stat(fullpath)
1184     }
1185     if err == nil {
1186         fm := fi.Mode()
1187         if fm.IsRegular() {
1188             // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1189             if syscall.Access(fullpath,5) == nil {
1190                 ffullpath = fullpath
1191                 ffound = true
1192                 if ! isin("-s", argv) {
1193                     showFileInfo(fullpath,argv)
1194                 }
1195             }
1196         }
1197     }
1198     return ffullpath, ffound
1199 }
1200 func which(list string, argv []string) (fullpathv []string, itis bool){
1201     if len(argv) <= 1 {
1202         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1203         return []string{"", false}
1204     }
1205     path := argv[1]
1206     if strBegins(path,"/") {
1207         // should check if excecutable?
1208         _,exOK := showIfExecutable(path,"/",argv)
1209         fmt.Printf("--D-- %v exOK=%v\n",path,exOK)
1210         return []string{path},exOK
1211     }
1212     pathenv, efound := os.LookupEnv(list)
1213     if ! efound {
1214         fmt.Printf("--E-- which: no \"%s\" environment\n",list)
1215         return []string{"", false}
1216     }
1217     showall := isin("-a",argv) || 0 <= strings.Index(path,"*")
1218     dirv := strings.Split(pathenv,PATHSEP)
1219     ffound := false
1220     ffullpath := path
1221     for _, dir := range dirv {
1222         if 0 <= strings.Index(path,"*") { // by wild-card
1223             list, _ := ioutil.ReadDir(dir)
1224             ffullpath, ffound = showMatchFile(list,path,dir,argv)
1225         }else{
1226             ffullpath, ffound = showIfExecutable(path,dir,argv)
1227         }
1228         //if ffound && !isin("-a", argv) {
1229         if ffound && !showall {
1230             break;
1231         }
1232     }
1233     return []string{ffullpath}, ffound
1234 }
1235 }
1236 func stripLeadingWSParg(argv[]string)([]string){
1237     for ; 0 < len(argv); {
1238         if len(argv[0]) == 0 {
1239             argv = argv[1:]
1240         }else{
1241             break
1242         }
1243     }
1244     return argv
1245 }
1246 func xEval(argv []string, nlend bool){
1247     argv = stripLeadingWSParg(argv)
1248     if len(argv) == 0 {
1249         fmt.Printf("eval [%&&format] [Go-expression]\n")

```

```

1250     return
1251 }
1252 pfmt := "%v"
1253 if argv[0][0] == '%' {
1254     pfmt = argv[0]
1255     argv = argv[1:]
1256 }
1257 if len(argv) == 0 {
1258     return
1259 }
1260 gocode := strings.Join(argv, " ");
1261 //fmt.Printf("eval [%v] [%v]\n",pfmt,gocode)
1262 fset := token.NewFileSet()
1263 rval, _ := types.Eval(fset,nil,token.NoPos,gocode)
1264 fmt.Printf(pfmt,(rval.Value) )
1265 if nlend { fmt.Printf("\n") }
1266 }
1267
1268 func getval(name string) (found bool, val int) {
1269     /* should expand the name here */
1270     if name == "gsh.pid" {
1271         return true, os.Getpid()
1272     }else
1273     if name == "gsh.ppid" {
1274         return true, os.Getppid()
1275     }
1276     return false, 0
1277 }
1278
1279 func echo(argv []string, nlend bool){
1280     for ai := 1; ai < len(argv); ai++ {
1281         if 1 < ai {
1282             fmt.Printf(" ");
1283         }
1284         arg := argv[ai]
1285         found, val := getval(arg)
1286         if found {
1287             fmt.Printf("%d",val)
1288         }else{
1289             fmt.Printf("%s",arg)
1290         }
1291     }
1292     if nlend {
1293         fmt.Printf("\n");
1294     }
1295 }
1296
1297 func resfile() string {
1298     return "gsh.tmp"
1299 }
1300 //var resF *File
1301 func resmap() {
1302     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1303     // https://deveoppaper.com/solution-to-golang-bad-file-descriptor-problem/
1304     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1305     if err != nil {
1306         fmt.Printf("refF could not open: %s\n",err)
1307     }else{
1308         fmt.Printf("refF opened\n")
1309     }
1310 }
1311
1312 // @2020-0821
1313 func gshScanArg(str string,strip int)(argv []string){
1314     var si = 0
1315     var sb = 0
1316     var inBracket = 0
1317     var arg1 = make([]byte,LINESIZE)
1318     var ax = 0
1319     debug := false
1320
1321     for ; si < len(str); si++ {
1322         if str[si] != ' ' {
1323             break
1324         }
1325     }
1326     sb = si
1327     for ; si < len(str); si++ {
1328         if sb <= si {
1329             if debug {
1330                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1331                     inBracket,sb,si,arg1[0:ax],str[si:])
1332             }
1333         }
1334         ch := str[si]
1335         if ch == '{' {
1336             inBracket += 1
1337             if 0 < strip && inBracket <= strip {
1338                 //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1339                 continue
1340             }
1341         }
1342         if 0 < inBracket {
1343             if ch == '}' {
1344                 inBracket -= 1
1345                 if 0 < strip && inBracket < strip {
1346                     //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1347                     continue
1348                 }
1349             }
1350             arg1[ax] = ch
1351             ax += 1
1352             continue
1353         }
1354         if str[si] == ' ' {
1355             argv = append(argv,string(arg1[0:ax]))
1356             if debug {
1357                 fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1358                     -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1359             }
1360             sb = si+1
1361             ax = 0
1362             continue
1363         }
1364         arg1[ax] = ch
1365         ax += 1
1366     }
1367     if sb < si {
1368         argv = append(argv,string(arg1[0:ax]))
1369         if debug {
1370             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1371                 -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1372         }
1373     }
1374     if debug {

```

```

1375     fmt.Printf("--Da- %d [%s] => [%d]%\n",strip,str,len(argv),argv)
1376 }
1377 return argv
1378 }
1379
1380 // should get stderr (into tmpfile ?) and return
1381 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1382     var pv = []int{-1,-1}
1383     syscall.Pipe(pv)
1384
1385     xarg := gshScanArg(name,1)
1386     name = strings.Join(xarg," ")
1387
1388     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
1389     pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
1390     fdix := 0
1391     dir := "?"
1392     if mode == "r" {
1393         dir = "<"
1394         fdix = 1 // read from the stdout of the process
1395     }else{
1396         dir = ">"
1397         fdix = 0 // write to the stdin of the process
1398     }
1399     gshPA := gsh.gshPA
1400     savfd := gshPA.Files[fdix]
1401
1402     var fd uintptr = 0
1403     if mode == "r" {
1404         fd = pout.Fd()
1405         gshPA.Files[fdix] = pout.Fd()
1406     }else{
1407         fd = pin.Fd()
1408         gshPA.Files[fdix] = pin.Fd()
1409     }
1410     // should do this by Goroutine?
1411     if false {
1412         fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1413         fmt.Printf("--RED1 [%d,%d,%d]->[%d,%d,%d]\n",
1414             os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1415             pin.Fd(),pout.Fd(),pout.Fd())
1416     }
1417     savi := os.Stdin
1418     savo := os.Stdout
1419     save := os.Stderr
1420     os.Stdin = pin
1421     os.Stdout = pout
1422     os.Stderr = pout
1423     gsh.BackGround = true
1424     gsh.gshellh(name)
1425     gsh.BackGround = false
1426     os.Stdin = savi
1427     os.Stdout = savo
1428     os.Stderr = save
1429
1430     gshPA.Files[fdix] = savfd
1431     return pin,pout,false
1432 }
1433
1434 // <a name="ex-commands">External commands</a>
1435 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {
1436     if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1437
1438     gshPA := gsh.gshPA
1439     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1440     if itis == false {
1441         return true,false
1442     }
1443     fullpath := fullpathv[0]
1444     argv = unescapeWhiteSPV(argv)
1445     if 0 < strings.Index(fullpath,".go") {
1446         nargv := argv // []string{}
1447         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1448         if itis == false {
1449             fmt.Printf("--F-- Go not found\n")
1450             return false,true
1451         }
1452         gofullpath := gofullpathv[0]
1453         nargv = []string{ gofullpath, "run", fullpath }
1454         fmt.Printf("--I-- %s [%s %s %s]\n",gofullpath,
1455             nargv[0],nargv[1],nargv[2])
1456         if exec {
1457             syscall.Exec(gofullpath,nargv,os.Environ())
1458         }else{
1459             pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1460             if gsh.BackGround {
1461                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%(v)\n",pid,len(argv),nargv)
1462                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1463             }else{
1464                 rusage := syscall.Rusage {}
1465                 syscall.Wait4(pid,nil,0,&rusage)
1466                 gsh.LastRusage = rusage
1467                 gsh.CmdCurrent.Rusagev[1] = rusage
1468             }
1469         }
1470     }else{
1471         if exec {
1472             syscall.Exec(fullpath,argv,os.Environ())
1473         }else{
1474             pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1475             //fmt.Printf("[%d]\n",pid); // '&' to be background
1476             if gsh.BackGround {
1477                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%(v)\n",pid,len(argv),argv)
1478                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1479             }else{
1480                 rusage := syscall.Rusage {}
1481                 syscall.Wait4(pid,nil,0,&rusage);
1482                 gsh.LastRusage = rusage
1483                 gsh.CmdCurrent.Rusagev[1] = rusage
1484             }
1485         }
1486     }
1487     return false,false
1488 }
1489
1490 // <a name="builtin">Builtin Commands</a>
1491 func (gshCtx *GshContext) sleep(argv []string) {
1492     if len(argv) < 2 {
1493         fmt.Printf("Sleep 100ms, 100us, 100ns, ...)\n")
1494         return
1495     }
1496     duration := argv[1];
1497     d, err := time.ParseDuration(duration)
1498     if err != nil {
1499         d, err = time.ParseDuration(duration+"s")

```

```

1500     if err != nil {
1501         fmt.Printf("duration ? %s (%s)\n",duration,err)
1502         return
1503     }
1504 }
1505 //fmt.Printf("Sleep %v\n",duration)
1506 time.Sleep(d)
1507 if 0 < len(argv[2:]) {
1508     gshCtx.gshellv(argv[2:])
1509 }
1510 }
1511 func (gshCtx *GshContext)repeat(argv []string) {
1512     if len(argv) < 2 {
1513         return
1514     }
1515     start0 := time.Now()
1516     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1517         if 0 < len(argv[2:]) {
1518             //start := time.Now()
1519             gshCtx.gshellv(argv[2:])
1520             end := time.Now()
1521             elps := end.Sub(start0);
1522             if( 1000000000 < elps ){
1523                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1524             }
1525         }
1526     }
1527 }
1528 }
1529 func (gshCtx *GshContext)gen(argv []string) {
1530     gshPA := gshCtx.gshPA
1531     if len(argv) < 2 {
1532         fmt.Printf("Usage: %s N\n",argv[0])
1533         return
1534     }
1535     // should br repeated by "repeat" command
1536     count, _ := strconv.Atoi(argv[1])
1537     fd := gshPA.Files[1] // Stdout
1538     file := os.NewFile(fd,"internalStdOut")
1539     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1540     //buf := []byte{}
1541     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1542     for gi := 0; gi < count; gi++ {
1543         file.WriteString(outdata)
1544     }
1545     //file.WriteString("\n")
1546     fmt.Printf("\n(%d B)\n",count*len(outdata));
1547     //file.Close()
1548 }
1549 }
1550 // <a name="rexec">Remote Execution</a> // 2020-0820
1551 func Elapsed(from time.Time)(string){
1552     elps := time.Now().Sub(from)
1553     if 1000000000 < elps {
1554         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1555     }else
1556     if 1000000 < elps {
1557         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1558     }else{
1559         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1560     }
1561 }
1562 func abptime(nanos int64)(string){
1563     if 1000000000 < nanos {
1564         return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/1000000)
1565     }else
1566     if 1000000 < nanos {
1567         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1568     }else{
1569         return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1570     }
1571 }
1572 func absbsize(size int64)(string){
1573     fsize := float64(size)
1574     if 1024*1024*1024 < size {
1575         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1576     }else
1577     if 1024*1024 < size {
1578         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1579     }else{
1580         return fmt.Sprintf("%.3fKiB",fsize/1024)
1581     }
1582 }
1583 func absze(size int64)(string){
1584     fsize := float64(size)
1585     if 1024*1024*1024 < size {
1586         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1587     }else
1588     if 1024*1024 < size {
1589         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1590     }else{
1591         return fmt.Sprintf("%.3fKiB",fsize/1024)
1592     }
1593 }
1594 func abbspd(totalB int64,ns int64)(string){
1595     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1596     if 1000 <= MBs {
1597         return fmt.Sprintf("%.3fGB/s",MBs/1000)
1598     }
1599     if 1 <= MBs {
1600         return fmt.Sprintf("%.3fMB/s",MBs)
1601     }else{
1602         return fmt.Sprintf("%.3fKB/s",MBs*1000)
1603     }
1604 }
1605 func abspsd(totalB int64,ns time.Duration)(string){
1606     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1607     if 1000 <= MBs {
1608         return fmt.Sprintf("%.3fGBps",MBs/1000)
1609     }
1610     if 1 <= MBs {
1611         return fmt.Sprintf("%.3fMBps",MBs)
1612     }else{
1613         return fmt.Sprintf("%.3fKBps",MBs*1000)
1614     }
1615 }
1616 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1617     Start := time.Now()
1618     buff := make([]byte,bsiz)
1619     var total int64 = 0
1620     var rem int64 = size
1621     nio := 0
1622     Prev := time.Now()
1623     var PrevSize int64 = 0
1624 }

```

```

1625 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1626 what,absize(total),size,nio)
1627
1628 for i:= 0; ; i++ {
1629     var len = bsiz
1630     if int(rem) < len {
1631         len = int(rem)
1632     }
1633     Now := time.Now()
1634     Elps := Now.Sub(Prev);
1635     if 1000000000 < Now.Sub(Prev) {
1636         fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1637             what,absize(total),size,nio,
1638             abspeed((total-PrevSize),Elps))
1639         Prev = Now;
1640         PrevSize = total
1641     }
1642     rlen := len
1643     if in != nil {
1644         // should watch the disconnection of out
1645         rcc,err := in.Read(buff[0:rlen])
1646         if err != nil {
1647             fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1648                 what,rcc,err,in.Name())
1649             break
1650         }
1651         rlen = rcc
1652         if string(buff[0:10]) == "(SoftEOF " {
1653             var ecc int64 = 0
1654             fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1655             fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v)/%v)\n",
1656                 what,ecc,total)
1657             if ecc == total {
1658                 break
1659             }
1660         }
1661     }
1662
1663     wlen := rlen
1664     if out != nil {
1665         wcc,err := out.Write(buff[0:rlen])
1666         if err != nil {
1667             fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1668                 what,wcc,err,out.Name())
1669             break
1670         }
1671         wlen = wcc
1672     }
1673     if wlen < rlen {
1674         fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1675             what,wlen,rlen)
1676         break;
1677     }
1678
1679     nio += 1
1680     total += int64(rlen)
1681     rem -= int64(rlen)
1682     if rem <= 0 {
1683         break
1684     }
1685 }
1686 Done := time.Now()
1687 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1688 TotalMB := float64(total)/1000000 //MB
1689 MBps := TotalMB / Elps
1690 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %v.3fMB/s\n",
1691     what,total,size,nio,absize(total),MBps)
1692 return total
1693 }
1694 func tcpPush(clnt *os.File){
1695     // shrink socket buffer and recover
1696     usleep(100);
1697 }
1698 func (gsh*GshContext)RexecServer(argv []string){
1699     debug := true
1700     Start0 := time.Now()
1701     Start := Start0
1702     // if local == "": { local = "0.0.0.0:9999" }
1703     local := "0.0.0.0:9999"
1704
1705     if 0 < len(argv) {
1706         if argv[0] == "-s" {
1707             debug = false
1708             argv = argv[1:]
1709         }
1710     }
1711     if 0 < len(argv) {
1712         argv = argv[1:]
1713     }
1714     port, err := net.ResolveTCPAddr("tcp",local);
1715     if err != nil {
1716         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1717         return
1718     }
1719     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1720     sconn, err := net.ListenTCP("tcp", port)
1721     if err != nil {
1722         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1723         return
1724     }
1725
1726     reqbuf := make([]byte,LINESIZE)
1727     res := ""
1728     for {
1729         fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1730         aconn, err := sconn.AcceptTCP()
1731         Start = time.Now()
1732         if err != nil {
1733             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1734             return
1735         }
1736         clnt, _ := aconn.File()
1737         fd := Clnt.Fd()
1738         ar := aconn.RemoteAddr()
1739         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1740             local,fd,ar) }
1741         res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1742         fmt.Fprintf(clnt,"%s",res)
1743         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1744         count, err := clnt.Read(reqbuf)
1745         if err != nil {
1746             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1747                 count,err,string(reqbuf))
1748         }
1749         req := string(reqbuf[:count])

```

```

1750     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1751     reqv := strings.Split(string(req),"r")
1752     cmdv := gshScanArg(reqv[0],0)
1753     //cmdv := strings.Split(reqv[0]," ")
1754     switch cmdv[0] {
1755     case "HELLO":
1756         res = fmt.Sprintf("250 %v",req)
1757     case "GET":
1758         // download {remotefile|-zN} [localfile]
1759         var dsz int64 = 32*1024*1024
1760         var bsize int = 64*1024
1761         var fname string = ""
1762         var in *os.File = nil
1763         var pseudoEOF = false
1764         if 1 < len(cmdv) {
1765             fname = cmdv[1]
1766             if strBegins(fname,"-z") {
1767                 fmt.Sscanf(fname[2:], "%d",&dsz)
1768             }else{
1769                 if strBegins(fname,"{") {
1770                     xin,xout,err := gsh.Popen(fname,"r")
1771                     if err {
1772                         }else{
1773                             xout.Close()
1774                             defer xin.Close()
1775                             in = xin
1776                             dsz = MaxStreamSize
1777                             pseudoEOF = true
1778                         }
1779                     }else{
1780                         xin,err := os.Open(fname)
1781                         if err != nil {
1782                             fmt.Printf("--En- GET (%v)\n",err)
1783                         }else{
1784                             defer xin.Close()
1785                             in = xin
1786                             fi,_ := xin.Stat()
1787                             dsz = fi.Size()
1788                         }
1789                     }
1790                 }
1791             //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsz,bsize)
1792             res = fmt.Sprintf("200 %v\r\n",dsz)
1793             fmt.Fprintf(clnt, "%v", res)
1794             tcpPush(clnt); // should be separated as line in receiver
1795             fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1796             wcount := fileRelay("SendGET",in,clnt,dsz,bsize)
1797             if pseudoEOF {
1798                 in.Close() // pipe from the command
1799                 // show end of stream data (its size) by OOB?
1800                 SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1801                 fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1802
1803                 tcpPush(clnt); // to let SoftEOF data apper at the top of received data
1804                 fmt.Fprintf(clnt, "%v\r\n",SoftEOF)
1805                 tcpPush(clnt); // to let SoftEOF alone in a packet (separate with 200 OK)
1806                 // with client generated random?
1807                 //fmt.Printf("--In- L: close %v (%v)\n",in.Fd(),in.Name())
1808             }
1809             res = fmt.Sprintf("200 GET done\r\n")
1810         case "PUT":
1811             // upload {srcfile|-zN} [dstfile]
1812             var dsz int64 = 32*1024*1024
1813             var bsize int = 64*1024
1814             var fname string = ""
1815             var out *os.File = nil
1816             if 1 < len(cmdv) { // localfile
1817                 fmt.Sscanf(cmdv[1], "%d",&dsz)
1818             }
1819             if 2 < len(cmdv) {
1820                 fname = cmdv[2]
1821                 if fname == "-" {
1822                     // nul dev
1823                 }else{
1824                     if strBegins(fname,"{") {
1825                         xin,xout,err := gsh.Popen(fname,"w")
1826                         if err {
1827                             }else{
1828                                 xin.Close()
1829                                 defer xout.Close()
1830                                 out = xout
1831                             }
1832                         }else{
1833                             // should write to temporary file
1834                             // should suppress ^C on tty
1835                             xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1836                             //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1837                             if err != nil {
1838                                 fmt.Printf("--En- PUT (%v)\n",err)
1839                             }else{
1840                                 out = xout
1841                             }
1842                         }
1843                     }
1844                     fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1845                         fname,local,err)
1846                 }
1847                 fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsz,bsize)
1848                 fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsz)
1849                 fmt.Fprintf(clnt, "200 %v OK\r\n",dsz)
1850                 fileRelay("RecvPUT",clnt,out,dsz,bsize)
1851                 res = fmt.Sprintf("200 PUT done\r\n")
1852             default:
1853                 res = fmt.Sprintf("400 What? %v",req)
1854             }
1855             swcc,serr := clnt.Write([]byte(res))
1856             if serr != nil {
1857                 fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1858             }else{
1859                 fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1860             }
1861             aconn.Close();
1862             clnt.Close();
1863         }
1864     }
1865     sconn.Close();
1866 }
1867 func (gsh*GshContext)RexecClient(argv []string)(int,string){
1868     debug := true
1869     Start := time.Now()
1870     if len(argv) == 1 {
1871         return -1,"EmptyARG"
1872     }
1873     argv = argv[1:]
1874     if argv[0] == "-serv" {
1875         gsh.RexecServer(argv[1:])
1876         return 0,"Server"
1877     }

```

```

1875 }
1876 remote := "0.0.0.0:9999"
1877 if argv[0][0] == '0' {
1878     remote = argv[0][1:]
1879     argv = argv[1:]
1880 }
1881 if argv[0] == "-s" {
1882     debug = false
1883     argv = argv[1:]
1884 }
1885 dport, err := net.ResolveTCPAddr("tcp",remote);
1886 if err != nil {
1887     fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1888     return -1,"AddressError"
1889 }
1890 fmt.Printf(Elapsed(Start)+"--In- C: Connecting to %s\n",remote)
1891 serv, err := net.DialTCP("tcp",nil,dport)
1892 if err != nil {
1893     fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1894     return -1,"CannotConnect"
1895 }
1896 if debug {
1897     al := serv.LocalAddr()
1898     fmt.Printf(Elapsed(Start)+"--In- C: Connected to %v <- %v\n",remote,al)
1899 }
1900
1901 req := ""
1902 res := make([]byte,LINESIZE)
1903 count,err := serv.Read(res)
1904 if err != nil {
1905     fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1906 }
1907 if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1908
1909 if argv[0] == "GET" {
1910     savPA := gsh.gshPA
1911     var bsize int = 64*1024
1912     req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1913     fmt.Printf(Elapsed(Start)+"--In- C: %v",req)
1914     fmt.Fprintf(serv,req)
1915     count,err = serv.Read(res)
1916     if err != nil {
1917     }else{
1918         var dsize int64 = 0
1919         var out *os.File = nil
1920         var out_tobeclosed *os.File = nil
1921         var fname string = ""
1922         var rcode int = 0
1923         var pid int = -1
1924         fmt.Sscanf(string(res),"%d %d",&rcode,&dsize)
1925         fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count]))
1926         if 3 <= len(argv) {
1927             fname = argv[2]
1928             if strBegins(fname,"{") {
1929                 xin,xout,err := gsh.Popen(fname,"w")
1930                 if err {
1931                 }else{
1932                     xin.Close()
1933                     defer xout.Close()
1934                     out = xout
1935                     out_tobeclosed = xout
1936                     pid = 0 // should be its pid
1937                 }
1938             }else{
1939                 // should write to temporary file
1940                 // should suppress ^C on tty
1941                 xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1942                 if err != nil {
1943                     fmt.Print("--En- %v\n",err)
1944                 }
1945                 out = xout
1946                 //fmt.Printf("--In-- %d > %s\n",out.Fd(),fname)
1947             }
1948         }
1949         in,_ := serv.File()
1950         fileRelay("RecvGET",in,out,dsize,bsize)
1951         if 0 <= pid {
1952             gsh.gshPA = savPA // recovery of Fd(), and more?
1953             fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
1954             out_tobeclosed.Close()
1955             //syscall.Wait4(pid,nil,0,nil) //@@
1956         }
1957     }
1958 }else
1959 if argv[0] == "PUT" {
1960     remote,_ := serv.File()
1961     var local *os.File = nil
1962     var dsize int64 = 32*1024*1024
1963     var bsize int = 64*1024
1964     var ofile string = "-"
1965     //fmt.Printf("--I-- Rex %v\n",argv)
1966     if 1 <= len(argv) {
1967         fname := argv[1]
1968         if strBegins(fname,"-z") {
1969             fmt.Sscanf(fname[2:], "%d",&dsize)
1970         }else
1971         if strBegins(fname,"{") {
1972             xin,xout,err := gsh.Popen(fname,"r")
1973             if err {
1974             }else{
1975                 xout.Close()
1976                 defer xin.Close()
1977                 //in = xin
1978                 local = xin
1979                 fmt.Printf("--In- [%d] < Upload output of %v\n",
1980                     local.Fd(),fname)
1981                 ofile = "-from."+fname
1982                 dsize = MaxStreamSize
1983             }
1984         }else{
1985             xlocal,err := os.Open(fname)
1986             if err != nil {
1987                 fmt.Printf("--En- (%s)\n",err)
1988                 local = nil
1989             }else{
1990                 local = xlocal
1991                 fi,_ := local.Stat()
1992                 dsize = fi.Size()
1993                 defer local.Close()
1994                 //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsize)
1995             }
1996             ofile = fname
1997             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
1998                 fname,dsize,local,err)
1999         }

```



```

2000     }
2001     if 2 < len(argv) && argv[2] != "" {
2002         ofile = argv[2]
2003         //fmt.Printf("(d)%v B.ofile=%v\n",len(argv),argv,ofile)
2004     }
2005     //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
2006     fmt.Printf(Elapsed(Start)+"--In- PUT %v (/%v)\n",dsize,bsize)
2007     req = fmt.Sprintf("PUT %v %v \r\n",dsize,ofile)
2008     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2009     fmt.Fprintf(serv, "%v",req)
2010     count,err = serv.Read(res)
2011     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
2012     fileRelay("SendPUT",local,remote,dsize,bsize)
2013 }else{
2014     req = fmt.Sprintf("%v\r\n",strings.Join(argv, " "))
2015     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2016     fmt.Fprintf(serv, "%v",req)
2017     //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
2018 }
2019 //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
2020 count,err = serv.Read(res)
2021 res := ""
2022 if count == 0 {
2023     res = "(nil)\r\n"
2024 }else{
2025     res = string(res[:count])
2026 }
2027 if err != nil {
2028     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,res)
2029 }else{
2030     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
2031 }
2032 serv.Close()
2033 //conn.Close()
2034
2035 var stat string
2036 var rcode int
2037 fmt.Sscanf(res, "%d %s", &rcode, &stat)
2038 //fmt.Printf("--D-- Client: %v (%v)",rcode,stat)
2039 return rcode,res
2040 }
2041
2042 // <a name="remote-sh">Remote Shell</a>
2043 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
2044 func (gsh*GshContext)FileCopy(argv []string){
2045     var host = ""
2046     var port = ""
2047     var upload = false
2048     var download = false
2049     var xargv = []string{"rex-gcp"}
2050     var srcv = []string{}
2051     var dstv = []string{}
2052     argv = argv[1:]
2053
2054     for _,v := range argv {
2055         /*
2056         if v[0] == '-' { // might be a pseudo file (generated date)
2057             continue
2058         }
2059         */
2060         obj := strings.Split(v,":")
2061         //fmt.Printf("%d %v %v\n",len(obj),v,obj)
2062         if 1 < len(obj) {
2063             host = obj[0]
2064             file := ""
2065             if 0 < len(host) {
2066                 gsh.LastServer.host = host
2067             }else{
2068                 host = gsh.LastServer.host
2069                 port = gsh.LastServer.port
2070             }
2071             if 2 < len(obj) {
2072                 port = obj[1]
2073                 if 0 < len(port) {
2074                     gsh.LastServer.port = port
2075                 }else{
2076                     port = gsh.LastServer.port
2077                 }
2078                 file = obj[2]
2079             }else{
2080                 file = obj[1]
2081             }
2082             if len(srcv) == 0 {
2083                 download = true
2084                 srcv = append(srcv,file)
2085                 continue
2086             }
2087             upload = true
2088             dstv = append(dstv,file)
2089             continue
2090         }
2091         /*
2092         idx := strings.Index(v,":")
2093         if 0 <= idx {
2094             remote = v[0:idx]
2095             if len(srcv) == 0 {
2096                 download = true
2097                 srcv = append(srcv,v[idx+1:])
2098                 continue
2099             }
2100             upload = true
2101             dstv = append(dstv,v[idx+1:])
2102             continue
2103         }
2104         */
2105         if download {
2106             dstv = append(dstv,v)
2107         }else{
2108             srcv = append(srcv,v)
2109         }
2110     }
2111     hostport := "@" + host + ":" + port
2112     if upload {
2113         if host != "" { xargv = append(xargv,hostport) }
2114         xargv = append(xargv,"PUT")
2115         xargv = append(xargv,srcv[0:]...)
2116         xargv = append(xargv,dstv[0:]...)
2117         //fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v // %v\n",hostport,dstv,srcv)
2118         fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v\n",hostport,dstv,srcv)
2119         gsh.RexecClient(xargv)
2120     }else
2121     if download {
2122         if host != "" { xargv = append(xargv,hostport) }
2123         xargv = append(xargv,"GET")
2124         xargv = append(xargv,srcv[0:]...)

```

```

2125     xargv = append(xargv,dstv[0:]...)
2126 //fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v // %v\n",hostport,srcv,dstv,xargv)
2127 fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v\n",hostport,srcv,dstv)
2128     gsh.RexecClient(xargv)
2129 }else{
2130 }
2131 }
2132 }
2133 // target
2134 func (gsh*GshContext)Trelpath(rloc string)(string){
2135     cwd, _ := os.Getwd()
2136     os.Chdir(gsh.RWD)
2137     os.Chdir(rloc)
2138     cwd, _ := os.Getwd()
2139     os.Chdir(cwd)
2140 }
2141 tpath := twd + "/" + rloc
2142 return tpath
2143 }
2144 // join to rremote GShell - [user@]host[:port] or cd host[:port]:path
2145 func (gsh*GshContext)Rjoin(argv[]string){
2146     if len(argv) <= 1 {
2147         fmt.Printf("--I-- current server = %v\n",gsh.RSERV)
2148         return
2149     }
2150     serv := argv[1]
2151     servv := strings.Split(serv,":")
2152     if 1 < len(servv) {
2153         if servv[0] == "lo" {
2154             servv[0] = "localhost"
2155         }
2156     }
2157     switch len(servv) {
2158     case 1:
2159         //if strings.Index(serv,":") < 0 {
2160             serv = servv[0] + ":" + fmt.Sprintf("%d",GSH_PORT)
2161         //}
2162     case 2: // host:port
2163         serv = strings.Join(servv,":")
2164     }
2165     xargv := []string{"rex-join","@"+serv,"HELO"}
2166     rcode,stat := gsh.RexecClient(xargv)
2167     if (rcode / 100) == 2 {
2168         fmt.Printf("--I-- OK Joined (%v) %v\n",rcode,stat)
2169         gsh.RSERV = serv
2170     }else{
2171         fmt.Printf("--I-- NG, could not joined (%v) %v\n",rcode,stat)
2172     }
2173 }
2174 func (gsh*GshContext)Rexec(argv[]string){
2175     if len(argv) <= 1 {
2176         fmt.Printf("--I-- rexec command [ | {file || {command} ]\n",gsh.RSERV)
2177         return
2178     }
2179 }
2180 /*
2181 nargv := gshScanArg(strings.Join(argv," "),0)
2182 fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2183 if nargv[1][0] != '{' {
2184     nargv[1] = "{" + nargv[1] + "}"
2185     fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2186 }
2187 argv = nargv
2188 */
2189 nargv := []string{}
2190 nargv = append(nargv,"{"+strings.Join(argv[1:], " ")+"}")
2191 fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2192 argv = nargv
2193 }
2194 xargv := []string{"rex-exec","@"+gsh.RSERV,"GET"}
2195 xargv = append(xargv,argv...)
2196 xargv = append(xargv,"/dev/tty")
2197 rcode,stat := gsh.RexecClient(xargv)
2198 if (rcode / 100) == 2 {
2199     fmt.Printf("--I-- OK Rexec (%v) %v\n",rcode,stat)
2200 }else{
2201     fmt.Printf("--I-- NG Rexec (%v) %v\n",rcode,stat)
2202 }
2203 }
2204 func (gsh*GshContext)Rchdir(argv[]string){
2205     if len(argv) <= 1 {
2206         return
2207     }
2208     cwd, _ := os.Getwd()
2209     os.Chdir(gsh.RWD)
2210     os.Chdir(argv[1])
2211     cwd, _ := os.Getwd()
2212     gsh.RWD = cwd
2213     fmt.Printf("--I-- JWD=%v\n",cwd)
2214     os.Chdir(cwd)
2215 }
2216 func (gsh*GshContext)Rpwd(argv[]string){
2217     fmt.Printf("%v\n",gsh.RWD)
2218 }
2219 func (gsh*GshContext)Rls(argv[]string){
2220     cwd, _ := os.Getwd()
2221     os.Chdir(gsh.RWD)
2222     argv[0] = "-ls"
2223     gsh.xFind(argv)
2224     os.Chdir(cwd)
2225 }
2226 func (gsh*GshContext)Rput(argv[]string){
2227     var local string = ""
2228     var remote string = ""
2229     if 1 < len(argv) {
2230         local = argv[1]
2231         remote = local // base name
2232     }
2233     if 2 < len(argv) {
2234         remote = argv[2]
2235     }
2236     fmt.Printf("--I-- jput from=%v to=%v\n",local,gsh.Trelpath(remote))
2237 }
2238 func (gsh*GshContext)Rget(argv[]string){
2239     var remote string = ""
2240     var local string = ""
2241     if 1 < len(argv) {
2242         remote = argv[1]
2243         local = remote // base name
2244     }
2245     if 2 < len(argv) {
2246         local = argv[2]
2247     }
2248     fmt.Printf("--I-- jget from=%v to=%v\n",gsh.Trelpath(remote),local)
2249 }

```

```

2250
2251 // <a name="network">network</a>
2252 // -s, -si, -so // bi-directional, source, sync (maybe socket)
2253 func (gshCtx*GshContext)sconnect(inTCP bool, argv []string) {
2254     gshPA := gshCtx.gshPA
2255     if len(argv) < 2 {
2256         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
2257         return
2258     }
2259     remote := argv[1]
2260     if remote == "" { remote = "0.0.0.0:9999" }
2261
2262     if inTCP { // TCP
2263         dport, err := net.ResolveTCPAddr("tcp",remote);
2264         if err != nil {
2265             fmt.Printf("Address error: %s (%s)\n",remote,err)
2266             return
2267         }
2268         conn, err := net.DialTCP("tcp",nil,dport)
2269         if err != nil {
2270             fmt.Printf("Connection error: %s (%s)\n",remote,err)
2271             return
2272         }
2273         file, _ := conn.File();
2274         fd := file.Fd()
2275         fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
2276
2277         savfd := gshPA.Files[1]
2278         gshPA.Files[1] = fd;
2279         gshCtx.gshelly(argv[2:])
2280         gshPA.Files[1] = savfd
2281         file.Close()
2282         conn.Close()
2283     }else{
2284         //dport, err := net.ResolveUDPAddr("udp4",remote);
2285         dport, err := net.ResolveUDPAddr("udp",remote);
2286         if err != nil {
2287             fmt.Printf("Address error: %s (%s)\n",remote,err)
2288             return
2289         }
2290         //conn, err := net.DialUDP("udp4",nil,dport)
2291         conn, err := net.DialUDP("udp",nil,dport)
2292         if err != nil {
2293             fmt.Printf("Connection error: %s (%s)\n",remote,err)
2294             return
2295         }
2296         file, _ := conn.File();
2297         fd := file.Fd()
2298
2299         ar := conn.RemoteAddr()
2300         //al := conn.LocalAddr()
2301         fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
2302             remote,ar.String(),fd)
2303
2304         savfd := gshPA.Files[1]
2305         gshPA.Files[1] = fd;
2306         gshCtx.gshelly(argv[2:])
2307         gshPA.Files[1] = savfd
2308         file.Close()
2309         conn.Close()
2310     }
2311 }
2312 func (gshCtx*GshContext)saccept(inTCP bool, argv []string) {
2313     gshPA := gshCtx.gshPA
2314     if len(argv) < 2 {
2315         fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
2316         return
2317     }
2318     local := argv[1]
2319     if local == "" { local = "0.0.0.0:9999" }
2320     if inTCP { // TCP
2321         port, err := net.ResolveTCPAddr("tcp",local);
2322         if err != nil {
2323             fmt.Printf("Address error: %s (%s)\n",local,err)
2324             return
2325         }
2326         //fmt.Printf("Listen at %s...\n",local);
2327         sconn, err := net.ListenTCP("tcp", port)
2328         if err != nil {
2329             fmt.Printf("Listen error: %s (%s)\n",local,err)
2330             return
2331         }
2332         //fmt.Printf("Accepting at %s...\n",local);
2333         aconn, err := sconn.AcceptTCP()
2334         if err != nil {
2335             fmt.Printf("Accept error: %s (%s)\n",local,err)
2336             return
2337         }
2338         file, _ := aconn.File()
2339         fd := file.Fd()
2340         fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
2341
2342         savfd := gshPA.Files[0]
2343         gshPA.Files[0] = fd;
2344         gshCtx.gshelly(argv[2:])
2345         gshPA.Files[0] = savfd
2346
2347         sconn.Close();
2348         aconn.Close();
2349         file.Close();
2350     }else{
2351         //port, err := net.ResolveUDPAddr("udp4",local);
2352         port, err := net.ResolveUDPAddr("udp",local);
2353         if err != nil {
2354             fmt.Printf("Address error: %s (%s)\n",local,err)
2355             return
2356         }
2357         fmt.Printf("Listen UDP at %s...\n",local);
2358         //uconn, err := net.ListenUDP("udp4", port)
2359         uconn, err := net.ListenUDP("udp", port)
2360         if err != nil {
2361             fmt.Printf("Listen error: %s (%s)\n",local,err)
2362             return
2363         }
2364         file, _ := uconn.File()
2365         fd := file.Fd()
2366         ar := uconn.RemoteAddr()
2367         remote := ""
2368         if ar != nil { remote = ar.String() }
2369         if remote == "" { remote = "?" }
2370
2371         // not yet received
2372         //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")
2373
2374         savfd := gshPA.Files[0]

```

```

2375     gshPA.Files[0] = fd;
2376     savenv := gshPA.Env
2377     gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
2378     gshCtx.gshellv(argv[2:])
2379     gshPA.Env = savenv
2380     gshPA.Files[0] = savfd
2381
2382     uconn.Close();
2383     file.Close();
2384 }
2385 }
2386
2387 // empty line command
2388 func (gshCtx*GshContext)xPwd(argv[]string){
2389 // execute context command, pwd + date
2390 // context notation, representation scheme, to be resumed at re-login
2391 cwd, _ := os.Getwd()
2392 switch {
2393 case isin("-a",argv):
2394     gshCtx.ShowChdirHistory(argv)
2395 case isin("-ls",argv):
2396     showFileInfo(cwd,argv)
2397 default:
2398     fmt.Printf("%s\n",cwd)
2399 case isin("-v",argv): // obsolete empty command
2400     t := time.Now()
2401     date := t.Format(time.UnixDate)
2402     exe, _ := os.Executable()
2403     host, _ := os.Hostname()
2404     fmt.Printf("PWD=\"%s\"",cwd)
2405     fmt.Printf(" HOST=\"%s\"",host)
2406     fmt.Printf(" DATE=\"%s\"",date)
2407     fmt.Printf(" TIME=\"%s\"",t.String())
2408     fmt.Printf(" PID=\"%d\"",os.Getpid())
2409     fmt.Printf(" EXE=\"%s\"",exe)
2410     fmt.Printf("\n")
2411 }
2412 }
2413
2414 // <a name="history">History</a>
2415 // these should be browsed and edited by HTTP browser
2416 // show the time of command with -t and direcotry with -ls
2417 // openfile-history, sort by -a -m -c
2418 // sort by elapsed time by -t -s
2419 // search by "more" like interface
2420 // edit history
2421 // sort history, and wc or uniq
2422 // CPU and other resource consumptions
2423 // limit showing range (by time or so)
2424 // export / import history
2425 func (gshCtx *GshContext)xHistory(argv []string){
2426     atWorkDirX := -1
2427     if 1 < len(argv) && strBegins(argv[1],"@") {
2428         atWorkDirX,_ = strconv.Atoi(argv[1][1:])
2429     }
2430     //fmt.Printf("--D-- showHistory(%v)\n",argv)
2431     for i, v := range gshCtx.CommandHistory {
2432         // exclude commands not to be listed by default
2433         // internal commands may be suppressed by default
2434         if v.CmdLine == "" && !isin("-a",argv) {
2435             continue;
2436         }
2437         if 0 <= atWorkDirX {
2438             if v.WorkDirX != atWorkDirX {
2439                 continue
2440             }
2441         }
2442         if !isin("-n",argv){ // like "fc"
2443             fmt.Printf("!%-2d ",i)
2444         }
2445         if isin("-v",argv){
2446             fmt.Println(v) // should be with it date
2447         }else{
2448             if isin("-l",argv) || isin("-l0",argv) {
2449                 elps := v.EndAt.Sub(v.StartAt);
2450                 start := v.StartAt.Format(time.Stamp)
2451                 fmt.Printf("@%d ",v.WorkDirX)
2452                 fmt.Printf("[%v] %11v/t ",start,elps)
2453             }
2454             if isin("-l",argv) && !isin("-l0",argv){
2455                 fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
2456             }
2457             if isin("-at",argv) { // isin("-ls",argv){
2458                 dhi := v.WorkDirX // workdir history index
2459                 fmt.Printf("@%d %s\t",dhi,v.WorkDir)
2460                 // show the FileInfo of the output command??
2461             }
2462             fmt.Printf("%s",v.CmdLine)
2463             fmt.Printf("\n")
2464         }
2465     }
2466 }
2467 // !n - history index
2468 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2469     if gline[0] == '!' {
2470         hix, err := strconv.Atoi(gline[1:])
2471         if err != nil {
2472             fmt.Printf("--E-- (%s : range)\n",hix)
2473             return "", false, true
2474         }
2475         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2476             fmt.Printf("--E-- (%d : out of range)\n",hix)
2477             return "", false, true
2478         }
2479         return gshCtx.CommandHistory[hix].CmdLine, false, false
2480     }
2481     // search
2482     //for i, v := range gshCtx.CommandHistory {
2483     //}
2484     return gline, false, false
2485 }
2486 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2487     if 0 <= hix && hix < len(gsh.CommandHistory) {
2488         return gsh.CommandHistory[hix].CmdLine,true
2489     }
2490     return "",false
2491 }
2492
2493 // temporary adding to PATH environment
2494 // cd name -lib for LD_LIBRARY_PATH
2495 // chdir with directory history (date + full-path)
2496 // -s for sort option (by visit date or so)
2497 func (gsh*GshContext)ShowChdirHistory(i int,v GChdirHistory, argv []string){
2498     fmt.Printf("!%-2d ",v.CmdIndex) // the first command at this WorkDir
2499     fmt.Printf("@%d ",i)

```

```

2500     fmt.Printf("%v ", v.MovedAt.Format(time.Stamp))
2501     showFileInfo(v.Dir, argv)
2502 }
2503 func (gsh*GshContext)ShowChdirHistory(argv []string){
2504     for i, v := range gsh.CkdirHistory {
2505         gsh.ShowChdirHistory1(i, v, argv)
2506     }
2507 }
2508 func skipOpts(argv[]string)(int){
2509     for i, v := range argv {
2510         if strBegins(v, "-") {
2511             }else{
2512                 return i
2513             }
2514         }
2515     return -1
2516 }
2517 func (gshCtx*GshContext)xChdir(argv []string){
2518     cdhist := gshCtx.CkdirHistory
2519     if isin("?", argv) || isin("-t", argv) || isin("-a", argv) {
2520         gshCtx.ShowChdirHistory(argv)
2521         return
2522     }
2523     pwd, _ := os.Getwd()
2524     dir := ""
2525     if len(argv) <= 1 {
2526         dir = toFullPath("-")
2527     }else{
2528         i := skipOpts(argv[1:])
2529         if i < 0 {
2530             dir = toFullPath("-")
2531         }else{
2532             dir = argv[1+i]
2533         }
2534     }
2535     if strBegins(dir, "@") {
2536         if dir == "@0" { // obsolete
2537             dir = gshCtx.StartDir
2538         }else
2539         if dir == "@1" {
2540             index := len(cdhist) - 1
2541             if 0 < index { index -= 1 }
2542             dir = cdhist[index].Dir
2543         }else{
2544             index, err := strconv.Atoi(dir[1:])
2545             if err != nil {
2546                 fmt.Printf("--E-- xChdir(%v)\n", err)
2547                 dir = "?"
2548             }else
2549             if len(gshCtx.CkdirHistory) <= index {
2550                 fmt.Printf("--E-- xChdir(history range error)\n")
2551                 dir = "?"
2552             }else{
2553                 dir = cdhist[index].Dir
2554             }
2555         }
2556     }
2557     if dir != "?" {
2558         err := os.Chdir(dir)
2559         if err != nil {
2560             fmt.Printf("--E-- xChdir(%s)(%v)\n", argv[1], err)
2561         }else{
2562             cwd, _ := os.Getwd()
2563             if cwd != pwd {
2564                 hist1 := GChdirHistory { }
2565                 hist1.Dir = cwd
2566                 hist1.MovedAt = time.Now()
2567                 hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2568                 gshCtx.CkdirHistory = append(cdhist, hist1)
2569                 if !isin("-s", argv){
2570                     //cwd, _ := os.Getwd()
2571                     //fmt.Printf("%s\n", cwd)
2572                     ix := len(gshCtx.CkdirHistory)-1
2573                     gshCtx.ShowChdirHistory1(ix, hist1, argv)
2574                 }
2575             }
2576         }
2577     }
2578     if isin("-ls", argv){
2579         cwd, _ := os.Getwd()
2580         showFileInfo(cwd, argv);
2581     }
2582 }
2583 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2584     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2585 }
2586 func RusageSubv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2587     TimeValSub(&ru1[0].Utime, &ru2[0].Utime)
2588     TimeValSub(&ru1[0].Stime, &ru2[0].Stime)
2589     TimeValSub(&ru1[1].Utime, &ru2[1].Utime)
2590     TimeValSub(&ru1[1].Stime, &ru2[1].Stime)
2591     return ru1
2592 }
2593 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2594     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2595     return tvs
2596 }
2597 /*
2598 func RusageAddv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2599     TimeValAdd(ru1[0].Utime, ru2[0].Utime)
2600     TimeValAdd(ru1[0].Stime, ru2[0].Stime)
2601     TimeValAdd(ru1[1].Utime, ru2[1].Utime)
2602     TimeValAdd(ru1[1].Stime, ru2[1].Stime)
2603     return ru1
2604 }
2605 */
2606
2607 // <a name="rusage">Resource Usage</a>
2608 func sRusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2609     // ru[0] self , ru[1] children
2610     ut := TimeValAdd(ru[0].Utime, ru[1].Utime)
2611     st := TimeValAdd(ru[0].Stime, ru[1].Stime)
2612     uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2613     su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2614     tu := uu + su
2615     ret := fmt.Sprintf("%v/sum", abftime(tu))
2616     ret += fmt.Sprintf(", %v/usr", abftime(uu))
2617     ret += fmt.Sprintf(", %v/sys", abftime(su))
2618     return ret
2619 }
2620 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2621     ut := TimeValAdd(ru[0].Utime, ru[1].Utime)
2622     st := TimeValAdd(ru[0].Stime, ru[1].Stime)
2623     fmt.Printf("%d.%06ds/u ", ut.Sec, ut.Usec) //ru[1].Utime.Sec, ru[1].Utime.Usec)
2624     fmt.Printf("%d.%06ds/s ", st.Sec, st.Usec) //ru[1].Stime.Sec, ru[1].Stime.Usec)

```

```

2625     return ""
2626 }
2627 func Getrusage()([2]syscall.Rusage){
2628     var ruv = [2]syscall.Rusage{}
2629     syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2630     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2631     return ruv
2632 }
2633 func showRusage(what string,argv []string, ru *syscall.Rusage){
2634     fmt.Printf("%s: ",what);
2635     fmt.Printf("User=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2636     fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2637     fmt.Printf(" Rss=%vB",ru.Maxrss)
2638     if isin("-l",argv){
2639         fmt.Printf(" MinFlt=%v",ru.Minflt)
2640         fmt.Printf(" MajFlt=%v",ru.Majflt)
2641         fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2642         fmt.Printf(" IdRSS=%vB",ru.Idrss)
2643         fmt.Printf(" Nswap=%vB",ru.Nswap)
2644         fmt.Printf(" Read=%v",ru.Inblock)
2645         fmt.Printf(" Write=%v",ru.Oublock)
2646     }
2647     fmt.Printf(" Snd=%v",ru.Msgsnd)
2648     fmt.Printf(" Rcv=%v",ru.Msgrcv)
2649     //if isin("-l",argv){
2650         fmt.Printf(" Sig=%v",ru.Nsignals)
2651     //}
2652     fmt.Printf("\n");
2653 }
2654 func (gshCtx *GshContext)xTime(argv []string)(bool){
2655     if 2 <= len(argv){
2656         gshCtx.LastRusage = syscall.Rusage{}
2657         rusagev1 := Getrusage()
2658         fin := gshCtx.gshellv(argv[1:])
2659         rusagev2 := Getrusage()
2660         showRusage(argv[1],argv,&gshCtx.LastRusage)
2661         rusagev := RusageSubv(rusagev2,rusagev1)
2662         showRusage("self",argv,&rusagev[0])
2663         showRusage("chld",argv,&rusagev[1])
2664         return fin
2665     }else{
2666         rusage:= syscall.Rusage {}
2667         syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2668         showRusage("self",argv, &rusage)
2669         syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2670         showRusage("chld",argv, &rusage)
2671         return false
2672     }
2673 }
2674 func (gshCtx *GshContext)xJobs(argv []string){
2675     fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
2676     for ji, pid := range gshCtx.BackGroundJobs {
2677         //wstat := syscall.WaitStatus {0}
2678         rusage := syscall.Rusage {}
2679         //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2680         wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2681         if err != nil {
2682             fmt.Printf("--E-- %d [%d] (%v)\n",ji,pid,err)
2683         }else{
2684             fmt.Printf("%d [%d] (%d)\n",ji,pid,wpid)
2685             showRusage("chld",argv,&rusage)
2686         }
2687     }
2688 }
2689 func (gsh*GshContext)inBackground(argv []string)(bool){
2690     if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2691     gsh.BackGround = true // set background option
2692     xfin := false
2693     xfin = gsh.gshellv(argv)
2694     gsh.BackGround = false
2695     return xfin
2696 }
2697 // -o file without command means just opening it and refer by #N
2698 // should be listed by "files" command
2699 func (gshCtx*GshContext)xOpen(argv []string){
2700     var pv = []int{-1,-1}
2701     err := syscall.Pipe(pv)
2702     fmt.Printf("--I-- pipe()=[%d,%d](%v)\n",pv[0],pv[1],err)
2703 }
2704 func (gshCtx*GshContext)fromPipe(argv []string){
2705 }
2706 func (gshCtx*GshContext)xClose(argv []string){
2707 }
2708 }
2709 // <a name="redirect">redirect</a>
2710 func (gshCtx*GshContext)redirect(argv []string)(bool){
2711     if len(argv) < 2 {
2712         return false
2713     }
2714 }
2715 cmd := argv[0]
2716 fname := argv[1]
2717 var file *os.File = nil
2718 }
2719 fdix := 0
2720 mode := os.O_RDONLY
2721 }
2722 switch {
2723 case cmd == "-i" || cmd == "<":
2724     fdix = 0
2725     mode = os.O_RDONLY
2726 case cmd == "-o" || cmd == ">":
2727     fdix = 1
2728     mode = os.O_RDWR | os.O_CREATE
2729 case cmd == "-a" || cmd == ">>":
2730     fdix = 1
2731     mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2732 }
2733 if fname[0] == '#' {
2734     fd, err := strconv.Atoi(fname[1:])
2735     if err != nil {
2736         fmt.Printf("--E-- (%v)\n",err)
2737         return false
2738     }
2739     file = os.NewFile(uintptr(fd),"MaybePipe")
2740 }else{
2741     xfile, err := os.OpenFile(argv[1], mode, 0600)
2742     if err != nil {
2743         fmt.Printf("--E-- (%s)\n",err)
2744         return false
2745     }
2746     file = xfile
2747 }
2748 gshPA := gshCtx.gshPA
2749 savfd := gshPA.Files[fdix]

```

```

2750 gshPA.Files[fdix] = file.Fd()
2751 fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2752 gshCtx.gshellv(argv[2:])
2753 gshPA.Files[fdix] = savfd
2754
2755 return false
2756 }
2757
2758 //fmt.Fprintf(res, "GShell Status: %g", html.EscapeString(req.URL.Path))
2759 func httpHandler(res http.ResponseWriter, req *http.Request){
2760 path := req.URL.Path
2761 fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2762 {
2763 gshCtxBuf, _ := setupGshContext()
2764 gshCtx := *gshCtxBuf
2765 fmt.Printf("--I-- %s\n",path[1:])
2766 gshCtx.tgshelll(path[1:])
2767 }
2768 fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
2769 }
2770 func (gshCtx *GshContext) httpServer(argv []string){
2771 http.HandleFunc("/", httpHandler)
2772 accport := "localhost:9999"
2773 fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2774 http.ListenAndServe(accport,nil)
2775 }
2776 func (gshCtx *GshContext)xGo(argv[]string){
2777 go gshCtx.gshellv(argv[1:]);
2778 }
2779 func (gshCtx *GshContext) xPs(argv[]string){}
2780 }
2781
2782 // <a name="plugin">Plugin</a>
2783 // plugin [-ls [names]] to list plugins
2784 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code
2785 func (gshCtx *GshContext) whichPlugin(name string,argv[]string)(pi *PluginInfo){
2786 pi = nil
2787 for _,p := range gshCtx.PluginFuncs {
2788 if p.Name == name && pi == nil {
2789 pi = *p
2790 }
2791 if !isin("-s",argv){
2792 //fmt.Printf("%v %v ",i,p)
2793 if isin("-ls",argv){
2794 showFileInfo(p.Path,argv)
2795 }else{
2796 fmt.Printf("%s\n",p.Name)
2797 }
2798 }
2799 }
2800 return pi
2801 }
2802 func (gshCtx *GshContext) xPlugin(argv[]string) (error) {
2803 if len(argv) == 0 || argv[0] == "-ls" {
2804 gshCtx.whichPlugin("",argv)
2805 return nil
2806 }
2807 name := argv[0]
2808 Pin := gshCtx.whichPlugin(name,[]string{"-s"})
2809 if Pin != nil {
2810 os.Args = argv // should be recovered?
2811 Pin.Addr.(func())()
2812 return nil
2813 }
2814 sofile := toFullpath(argv[0] + ".so") // or find it by which($PATH)
2815
2816 p_err := plugin.Open(sofile)
2817 if err != nil {
2818 fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2819 return err
2820 }
2821 fname := "Main"
2822 f_err := p.Lookup(fname)
2823 if( err != nil){
2824 fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2825 return err
2826 }
2827 pin := PluginInfo {p,f,name,sofile}
2828 gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2829 fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2830
2831 //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2832 os.Args = argv
2833 f.(func())()
2834 return err
2835 }
2836 func (gshCtx*GshContext)Args(argv[]string){
2837 for i,v := range os.Args {
2838 fmt.Printf("[%v] %v\n",i,v)
2839 }
2840 }
2841 func (gshCtx *GshContext) showVersion(argv[]string){
2842 if isin("-l",argv) {
2843 fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2844 }else{
2845 fmt.Printf("%v",VERSION);
2846 }
2847 if isin("-a",argv) {
2848 fmt.Printf(" %s",AUTHOR)
2849 }
2850 if !isin("-n",argv) {
2851 fmt.Printf("\n")
2852 }
2853 }
2854
2855 // <a name="scanf">Scanf</a> // string decomposer
2856 // scanf [format] [input]
2857 func scanv(sstr string)(strv[]string){
2858 strv = strings.Split(sstr, " ")
2859 return strv
2860 }
2861 func scanUntil(src,end string)(rstr string,leng int){
2862 idx := strings.Index(src,end)
2863 if 0 <= idx {
2864 rstr = src[0:idx]
2865 return rstr,idx+len(end)
2866 }
2867 return src,0
2868 }
2869
2870 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting
2871 func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
2872 //vint,err := strconv.Atoi(vstr)
2873 var ival int64 = 0
2874 n := 0

```

```

2875     err := error(nil)
2876     if strBegins(vstr, "_") {
2877         vx, _ := strconv.Atoi(vstr[1:])
2878         if vx < len(gsh.iValues) {
2879             vstr = gsh.iValues[vx]
2880         }else{
2881             }
2882     }
2883     // should use Eval()
2884     if strBegins(vstr, "0x") {
2885         n, err = fmt.Sscanf(vstr[2:], "%x", &ival)
2886     }else{
2887         n, err = fmt.Sscanf(vstr, "%d", &ival)
2888     //fmt.Printf("--D-- n=%d err=(%v) (%s)=%v\n", n, err, vstr, ival)
2889     }
2890     if n == 1 && err == nil {
2891         //fmt.Printf("--D-- formatn(%v) ival(%v)\n", fmts, ival)
2892         fmt.Printf("%"+fmts, ival)
2893     }else{
2894         if isin("-bn", optv){
2895             fmt.Printf("%"+fmts, filepath.Base(vstr))
2896         }else{
2897             fmt.Printf("%"+fmts, vstr)
2898         }
2899     }
2900 }
2901 func (gsh*GshContext)printfv(fmts, div string, argv[jstring], optv[jstring], list[jstring]){
2902     //fmt.Printf("%d", len(list))
2903     //curfmt := "v"
2904     outlen := 0
2905     curfmt := gsh.iFormat
2906
2907     if 0 < len(fmts) {
2908         for xi := 0; xi < len(fmts); xi++ {
2909             fch := fmts[xi]
2910             if fch == '%' {
2911                 if xi+1 < len(fmts) {
2912                     curfmt = string(fmts[xi+1])
2913                 }
2914                 xi += 1
2915                 if xi+1 < len(fmts) && fmts[xi+1] == '(' {
2916                     vals, leng := scanUntil(fmts[xi+2:], ")")
2917                     //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n", curfmt, vals, leng)
2918                     gsh.printVal(curfmt, vals, optv)
2919                     xi += 2+leng-1
2920                     outlen += 1
2921                 }
2922                 continue
2923             }
2924             if fch == ' ' {
2925                 hi, leng := scanInt(fmts[xi+1:])
2926                 if 0 < leng {
2927                     if hi < len(gsh.iValues) {
2928                         gsh.printVal(curfmt, gsh.iValues[hi], optv)
2929                         outlen += 1 // should be the real length
2930                     }else{
2931                         fmt.Printf("((out-range))")
2932                     }
2933                     xi += leng
2934                     continue;
2935                 }
2936             }
2937             fmt.Printf("%c", fch)
2938             outlen += 1
2939         }
2940     }else{
2941         //fmt.Printf("--D-- print (%s)\n")
2942         for i, v := range list {
2943             if 0 < i {
2944                 fmt.Printf(div)
2945             }
2946             gsh.printVal(curfmt, v, optv)
2947             outlen += 1
2948         }
2949     }
2950     if 0 < outlen {
2951         fmt.Printf("\n")
2952     }
2953 }
2954 }
2955 func (gsh*GshContext)Scanv(argv[jstring]){
2956     //fmt.Printf("--D-- Scanv(%v)\n", argv)
2957     if len(argv) == 1 {
2958         return
2959     }
2960     argv = argv[1:]
2961     fmts := ""
2962     if strBegins(argv[0], "-F") {
2963         fmts = argv[0]
2964         gsh.iDelimiter = fmts
2965         argv = argv[1:]
2966     }
2967     input := strings.Join(argv, " ")
2968     if fmts == "" { // simple decomposition
2969         v := scanv(input)
2970         gsh.iValues = v
2971         //fmt.Printf("%v\n", strings.Join(v, ","))
2972     }else{
2973         v := make([]string, 8)
2974         n, err := fmt.Sscanf(input, fmts, &v[0], &v[1], &v[2], &v[3])
2975         fmt.Printf("--D-- Sscanf ->(%v) n=%d err=(%v)\n", v, n, err)
2976         gsh.iValues = v
2977     }
2978 }
2979 func (gsh*GshContext)Printv(argv[jstring]){
2980     if false { //@@@
2981         fmt.Printf("%v\n", strings.Join(argv[1:], " "))
2982         return
2983     }
2984     //fmt.Printf("--D-- Printv(%v)\n", argv)
2985     //fmt.Printf("%v\n", strings.Join(gsh.iValues, ","))
2986     div := gsh.iDelimiter
2987     fmts := ""
2988     argv = argv[1:]
2989     if 0 < len(argv) {
2990         if strBegins(argv[0], "-F") {
2991             div = argv[0][2:]
2992             argv = argv[1:]
2993         }
2994     }
2995     optv := []string{}
2996     for _, v := range argv {
2997         if strBegins(v, "-"){
2998             optv = append(optv, v)

```



```

3000         argv = argv[1:]
3001     }else{
3002         break;
3003     }
3004 }
3005 if 0 < len(argv) {
3006     fmts = strings.Join(argv, " ")
3007 }
3008 gsh.printf(fmts,div,argv,optv,gsh.iValues)
3009 }
3010 func (gsh*GshContext)Basename(argv[]string){
3011     for i,v := range gsh.iValues {
3012         gsh.iValues[i] = filepath.Base(v)
3013     }
3014 }
3015 func (gsh*GshContext)Sortv(argv[]string){
3016     sv := gsh.iValues
3017     sort.Slice(sv , func(i,j int) bool {
3018         return sv[i] < sv[j]
3019     })
3020 }
3021 func (gsh*GshContext)Shiftv(argv[]string){
3022     vi := len(gsh.iValues)
3023     if 0 < vi {
3024         if isin("-r",argv) {
3025             top := gsh.iValues[0]
3026             gsh.iValues = append(gsh.iValues[1:],top)
3027         }else{
3028             gsh.iValues = gsh.iValues[1:]
3029         }
3030     }
3031 }
3032 }
3033 func (gsh*GshContext)Enq(argv[]string){
3034 }
3035 func (gsh*GshContext)Deq(argv[]string){
3036 }
3037 func (gsh*GshContext)Push(argv[]string){
3038     gsh.iValStack = append(gsh.iValStack,argv[1:])
3039     fmt.Printf("depth=%d\n",len(gsh.iValStack))
3040 }
3041 func (gsh*GshContext)Dump(argv[]string){
3042     for i,v := range gsh.iValStack {
3043         fmt.Printf("%d %v\n",i,v)
3044     }
3045 }
3046 func (gsh*GshContext)Pop(argv[]string){
3047     depth := len(gsh.iValStack)
3048     if 0 < depth {
3049         v := gsh.iValStack[depth-1]
3050         if isin("-cat",argv){
3051             gsh.iValues = append(gsh.iValues,v...)
3052         }else{
3053             gsh.iValues = v
3054         }
3055         gsh.iValStack = gsh.iValStack[0:depth-1]
3056         fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3057     }else{
3058         fmt.Printf("depth=%d\n",depth)
3059     }
3060 }
3061 }
3062 // <a name="interpreter">Command Interpreter</a>
3063 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3064     fin = false
3065 }
3066 if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3067 if len(argv) <= 0 {
3068     return false
3069 }
3070 xargv := []string{}
3071 for ai := 0; ai < len(argv); ai++ {
3072     xargv = append(xargv,subst(gshCtx,argv[ai],false))
3073 }
3074 argv = xargv
3075 if false {
3076     for ai := 0; ai < len(argv); ai++ {
3077         fmt.Printf("[%d] %s [%d]T\n",
3078             ai,argv[ai],len(argv[ai]),argv[ai])
3079     }
3080 }
3081 cmd := argv[0]
3082 if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
3083 switch { // https://tour.golang.org/flowcontrol/11
3084 case cmd == "":
3085     gshCtx.xPwd([]string{}); // empty command
3086 case cmd == "-x":
3087     gshCtx.CmdTrace = ! gshCtx.CmdTrace
3088 case cmd == "-xt":
3089     gshCtx.CmdTime = ! gshCtx.CmdTime
3090 case cmd == "-ot":
3091     gshCtx.sconnect(true, argv)
3092 case cmd == "-ou":
3093     gshCtx.sconnect(false, argv)
3094 case cmd == "-it":
3095     gshCtx.saccept(true , argv)
3096 case cmd == "-iu":
3097     gshCtx.saccept(false, argv)
3098 case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3099     gshCtx.redirect(argv)
3100 case cmd == "|":
3101     gshCtx.fromPipe(argv)
3102 case cmd == "args":
3103     gshCtx.Args(argv)
3104 case cmd == "bg" || cmd == "-bg":
3105     rfin := gshCtx.inBackground(argv[1:])
3106     return rfin
3107 case cmd == "-bn":
3108     gshCtx.Basename(argv)
3109 case cmd == "call":
3110     _ := gshCtx.excommand(false,argv[1:])
3111 case cmd == "cd" || cmd == "chdir":
3112     gshCtx.xChdir(argv);
3113 case cmd == "-cksum":
3114     gshCtx.xFind(argv)
3115 case cmd == "-sum":
3116     gshCtx.xFind(argv)
3117 case cmd == "close":
3118     gshCtx.xClose(argv)
3119 case cmd == "gcp":
3120     gshCtx.FileCopy(argv)
3121 case cmd == "dec" || cmd == "decode":
3122     gshCtx.Dec(argv)
3123 case cmd == "#define":
3124     case cmd == "dic":

```

```

3125     xDic(argv)
3126 case cmd == "dump":
3127     gshCtx.Dump(argv)
3128 case cmd == "echo":
3129     echo(argv,true)
3130 case cmd == "enc" || cmd == "encode":
3131     gshCtx.Enc(argv)
3132 case cmd == "env":
3133     env(argv)
3134 case cmd == "eval":
3135     xEval(argv[1:],true)
3136 case cmd == "exec":
3137     _ = gshCtx.excommand(true,argv[1:])
3138     // should not return here
3139 case cmd == "exit" || cmd == "quit":
3140     // write Result code EXIT to 3>
3141     return true
3142 case cmd == "fds":
3143     // dump the attributes of fds (of other process)
3144 case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3145     gshCtx.xFind(argv[1:])
3146 case cmd == "fu":
3147     gshCtx.xFind(argv[1:])
3148 case cmd == "fork":
3149     // mainly for a server
3150 case cmd == "-gen":
3151     gshCtx.gen(argv)
3152 case cmd == "-go":
3153     gshCtx.xGo(argv)
3154 case cmd == "-grep":
3155     gshCtx.xFind(argv)
3156 case cmd == "gdeg":
3157     gshCtx.Deg(argv)
3158 case cmd == "genq":
3159     gshCtx.Enq(argv)
3160 case cmd == "gpq":
3161     gshCtx.Pop(argv)
3162 case cmd == "gpush":
3163     gshCtx.Push(argv)
3164 case cmd == "history" || cmd == "hi": // hi should be alias
3165     gshCtx.xHistory(argv)
3166 case cmd == "jobs":
3167     gshCtx.xJobs(argv)
3168 case cmd == "lnsp":
3169     gshCtx.Splitline(argv)
3170 case cmd == "-ls":
3171     gshCtx.xFind(argv)
3172 case cmd == "nop":
3173     // do nothing
3174 case cmd == "pipe":
3175     gshCtx.xOpen(argv)
3176 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3177     gshCtx.xPlugin(argv[1:])
3178 case cmd == "print" || cmd == "-pr":
3179     // output internal slice // also sprintf should be
3180     gshCtx.Printv(argv)
3181 case cmd == "ps":
3182     gshCtx.xPs(argv)
3183 case cmd == "pstitle":
3184     // to be gsh.title
3185 case cmd == "rexe" || cmd == "rexd":
3186     gshCtx.RexecServer(argv)
3187 case cmd == "rexc" || cmd == "rex":
3188     gshCtx.RexecClient(argv)
3189 case cmd == "repeat" || cmd == "rep": // repeat cond command
3190     gshCtx.repeat(argv)
3191 case cmd == "scan":
3192     // scan input (or so in fscanf) to internal slice (like Files or map)
3193     gshCtx.Scanv(argv)
3194 case cmd == "set":
3195     // set name ...
3196 case cmd == "serv":
3197     gshCtx.httpServer(argv)
3198 case cmd == "shift":
3199     gshCtx.Shiftv(argv)
3200 case cmd == "sleep":
3201     gshCtx.sleep(argv)
3202 case cmd == "-sort":
3203     gshCtx.Sortv(argv)
3204
3205 case cmd == "j" || cmd == "join":
3206     gshCtx.RJoin(argv)
3207 case cmd == "a" || cmd == "alpa":
3208     gshCtx.Rexec(argv)
3209 case cmd == "jcd" || cmd == "jchdir":
3210     gshCtx.Rchdir(argv)
3211 case cmd == "jget":
3212     gshCtx.Rget(argv)
3213 case cmd == "jls":
3214     gshCtx.Rls(argv)
3215 case cmd == "jput":
3216     gshCtx.Rput(argv)
3217 case cmd == "jpwd":
3218     gshCtx.Rpwd(argv)
3219
3220 case cmd == "time":
3221     fin = gshCtx.xTime(argv)
3222 case cmd == "pwd":
3223     gshCtx.xPwd(argv);
3224 case cmd == "ver" || cmd == "-ver" || cmd == "version":
3225     gshCtx.showVersion(argv)
3226 case cmd == "where":
3227     // data file or so?
3228 case cmd == "which":
3229     which("PATH",argv);
3230 default:
3231     if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3232         gshCtx.xPlugin(argv)
3233     }else{
3234         notfound,_ := gshCtx.excommand(false,argv)
3235         if notfound {
3236             fmt.Printf("--E-- command not found (%v)\n",cmd)
3237         }
3238     }
3239 }
3240 return fin
3241 }
3242
3243 func (gsh*GshContext)gshelll(gline string) (rfin bool) {
3244     argv := strings.Split(string(gline)," ")
3245     fin := gsh.gshellv(argv)
3246     return fin
3247 }
3248 func (gsh*GshContext)tgshelll(gline string)(xfin bool){
3249     start := time.Now()

```

```

3250 fin := gsh.gshelll(gline)
3251 end := time.Now()
3252 elps := end.Sub(start);
3253 if gsh.CmdTime {
3254     fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (&d.%09ds)\n",
3255         elps/1000000000, elps%1000000000)
3256 }
3257 return fin
3258 }
3259 func Ttyid() (int) {
3260     fi, err := os.Stdin.Stat()
3261     if err != nil {
3262         return 0;
3263     }
3264     //fmt.Printf("Stdin: %v Dev=%d\n",
3265     //    fi.Mode(), fi.Mode() & os.ModeDevice)
3266     if (fi.Mode() & os.ModeDevice) != 0 {
3267         stat := syscall.Stat_t{};
3268         err := syscall.Fstat(0, &stat)
3269         if err != nil {
3270             //fmt.Printf("--I-- Stdin: (%v)\n", err)
3271         } else {
3272             //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3273             //    stat.Rdev & 0xFF, stat.Rdev);
3274             //fmt.Printf("--I-- Stdin: tty%d\n", stat.Rdev & 0xFF);
3275             return int(stat.Rdev & 0xFF)
3276         }
3277     }
3278     return 0
3279 }
3280 func (gshCtx *GshContext) ttyfile() string {
3281     //fmt.Printf("--I-- GSH_HOME=%s\n", gshCtx.GshHomeDir)
3282     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3283         fmt.Sprintf("%02d", gshCtx.TerminalId)
3284     //strconv.Itoa(gshCtx.TerminalId)
3285     //fmt.Printf("--I-- ttyfile=%s\n", ttyfile)
3286     return ttyfile
3287 }
3288 func (gshCtx *GshContext) ttyline() (*os.File) {
3289     file, err := os.OpenFile(gshCtx.ttyfile(), os.O_RDWR|os.O_CREATE|os.O_TRUNC, 0600)
3290     if err != nil {
3291         fmt.Printf("--F-- cannot open %s (%s)\n", gshCtx.ttyfile(), err)
3292         return file;
3293     }
3294     return file
3295 }
3296 func (gshCtx *GshContext) getline(hix int, skipping bool, prevline string) (string) {
3297     if (skipping) {
3298         reader := bufio.NewReaderSize(os.Stdin, LINESIZE)
3299         line, _, _ := reader.ReadLine()
3300         return string(line)
3301     } else {
3302         if true {
3303             return xgetline(hix, prevline, gshCtx)
3304         }
3305         /*
3306         else
3307         if( with_exgetline && gshCtx.GetLine != "" ){
3308             //var xhix int64 = int64(hix); // cast
3309             newenv := os.Environ()
3310             newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix), 10) )
3311             tty := gshCtx.ttyline()
3312             tty.WriteString(prevline)
3313             Pa := os.ProcAttr {
3314                 "", // start dir
3315                 newenv, //os.Environ(),
3316                 []*os.File{os.Stdin, os.Stdout, os.Stderr, tty},
3317                 nil,
3318             }
3319             //fmt.Printf("--I-- getline=%s // %s\n", gsh_getlinev[0], gshCtx.GetLine)
3320             proc, err := os.StartProcess(gsh_getlinev[0], []string{"getline", "getline"}, &Pa)
3321             if err != nil {
3322                 fmt.Printf("--F-- getline process error (%v)\n", err)
3323                 // for ; ; { }
3324                 return "exit (getline program failed)"
3325             }
3326             //stat, err := proc.Wait()
3327             proc.Wait()
3328             buff := make([]byte, LINESIZE)
3329             count, err := tty.Read(buff)
3330             //_, err := tty.Read(buff)
3331             //fmt.Printf("--D-- getline (%d)\n", count)
3332             if err != nil {
3333                 if ! (count == 0) { // && err.String() == "EOF" } {
3334                     fmt.Printf("--E-- getline error (%s)\n", err)
3335                 }
3336             } else {
3337                 //fmt.Printf("--I-- getline OK \"%s\"\n", buff)
3338             }
3339             tty.Close()
3340             gline := string(buff[0:count])
3341             return gline
3342         } else
3343         {
3344             // if isatty {
3345             //     fmt.Printf("%d", hix)
3346             //     fmt.Print(PROMPT)
3347             // }
3348             reader := bufio.NewReaderSize(os.Stdin, LINESIZE)
3349             line, _, _ := reader.ReadLine()
3350             return string(line)
3351         }
3352     }
3353 }
3354 }
3355 }
3356 //== begin ===== getline
3357 /*
3358 * getline.c
3359 * 2020-0819 extracted from dog.c
3360 * getline.go
3361 * 2020-0822 ported to Go
3362 */
3363 /*
3364 package main // getline main
3365 import (
3366     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3367     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3368     "os" // <a href="https://golang.org/pkg/os/">os</a>
3369     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3370     "bytes" // <a href="https://golang.org/pkg/bytes/">bytes</a>
3371     "os/exec" // <a href="https://golang.org/pkg/os/">os</a>
3372 )
3373 */
3374

```

```

3375 // C language compatibility functions
3376 var errno = 0
3377 var stdin *os.File = os.Stdin
3378 var stdout *os.File = os.Stdout
3379 var stderr *os.File = os.Stderr
3380 var EOF = -1
3381 var NULL = 0
3382 type FILE os.File
3383 type StrBuff []byte
3384 var NULL_FP *os.File = nil
3385 var NULLSP = 0
3386 //var LINESIZE = 1024
3387
3388 func system(cmdstr string)(int){
3389     PA := syscall.ProcAttr {
3390         "" // the starting directory
3391         os.Environ(),
3392         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3393         nil,
3394     }
3395     argv := strings.Split(cmdstr, " ")
3396     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3397     if( err != nil ){
3398         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3399     }
3400     syscall.Wait4(pid,nil,0,nil)
3401
3402     /*
3403     argv := strings.Split(cmdstr, " ")
3404     fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3405     //cmd := exec.Command(argv[0],...)
3406     cmd := exec.Command(argv[0],argv[1],argv[2])
3407     cmd.Stdin = strings.NewReader("output of system")
3408     var out bytes.Buffer
3409     cmd.Stdout = &out
3410     var serr bytes.Buffer
3411     cmd.Stderr = &serr
3412     err := cmd.Run()
3413     if err != nil {
3414         fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)
3415         fmt.Printf("ERR:%s\n",serr.String())
3416     }else{
3417         fmt.Printf("%s",out.String())
3418     }
3419     */
3420     return 0
3421 }
3422 func atoi(str string)(ret int){
3423     ret,err := fmt.Sscanf(str,"%d",ret)
3424     if err == nil {
3425         return ret
3426     }else{
3427         // should set errno
3428         return 0
3429     }
3430 }
3431 func getenv(name string)(string){
3432     val,got := os.LookupEnv(name)
3433     if got {
3434         return val
3435     }else{
3436         return "?"
3437     }
3438 }
3439 func strcpy(dst StrBuff, src string){
3440     var i int
3441     srcb := []byte(src)
3442     for i = 0; i < len(src) && srcb[i] != 0; i++ {
3443         dst[i] = srcb[i]
3444     }
3445     dst[i] = 0
3446 }
3447 func xstrcpy(dst StrBuff, src StrBuff){
3448     dst = src
3449 }
3450 func strcat(dst StrBuff, src StrBuff){
3451     dst = append(dst,src...)
3452 }
3453 func strdup(str StrBuff)(string){
3454     return string(str[0:strlen(str)])
3455 }
3456 func strlen(str string)(int){
3457     return len(str)
3458 }
3459 func strlen(str StrBuff)(int){
3460     var i int
3461     for i = 0; i < len(str) && str[i] != 0; i++ {
3462     }
3463     return i
3464 }
3465 func sizeof(data StrBuff)(int){
3466     return len(data)
3467 }
3468 func isatty(fd int)(ret int){
3469     return 1
3470 }
3471
3472 func fopen(file string,mode string)(fp*os.File){
3473     if mode == "r" {
3474         fp,err := os.Open(file)
3475         if( err != nil ){
3476             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
3477             return NULL_FP;
3478         }
3479         return fp;
3480     }else{
3481         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3482         if( err != nil ){
3483             return NULL_FP;
3484         }
3485         return fp;
3486     }
3487 }
3488 func fclose(fp*os.File){
3489     fp.Close()
3490 }
3491 func fflush(fp *os.File)(int){
3492     return 0
3493 }
3494 func fgetc(fp*os.File)(int){
3495     var buf [1]byte
3496     _,err := fp.Read(buf[0:1])
3497     if( err != nil ){
3498         return EOF;
3499     }else{

```

```

3500     return int(buf[0])
3501 }
3502 }
3503 func sfgets(str*string, size int, fp*os.File)(int){
3504     buf := make(StrBuff,size)
3505     var ch int
3506     var i int
3507     for i = 0; i < len(buf)-1; i++ {
3508         ch = fgetc(fp)
3509         //fprintf(stderr,"--fgets %d/%d %X\n",i,len(buf),ch)
3510         if( ch == EOF ){
3511             break;
3512         }
3513         buf[i] = byte(ch);
3514         if( ch == '\n' ){
3515             break;
3516         }
3517     }
3518     buf[i] = 0
3519     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3520     return i
3521 }
3522 func fgets(buf StrBuff, size int, fp*os.File)(int){
3523     var ch int
3524     var i int
3525     for i = 0; i < len(buf)-1; i++ {
3526         ch = fgetc(fp)
3527         //fprintf(stderr,"--fgets %d/%d %X\n",i,len(buf),ch)
3528         if( ch == EOF ){
3529             break;
3530         }
3531         buf[i] = byte(ch);
3532         if( ch == '\n' ){
3533             break;
3534         }
3535     }
3536     buf[i] = 0
3537     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3538     return i
3539 }
3540 func fputc(ch int , fp*os.File)(int){
3541     var buf [1]byte
3542     buf[0] = byte(ch)
3543     fp.Write(buf[0:1])
3544     return 0
3545 }
3546 func fputs(buf StrBuff, fp*os.File)(int){
3547     fp.Write(buf)
3548     return 0
3549 }
3550 func xfputss(str string, fp*os.File)(int){
3551     return fputs([]byte(str),fp)
3552 }
3553 func sscanf(str StrBuff,fmts string, params ...interface{})(int){
3554     fmt.Sscanf(string(str[0:strlen(str)]),fmts,params...)
3555     return 0
3556 }
3557 func fprintf(fp*os.File,fmts string, params ...interface{})(int){
3558     fmt.Fprintf(fp,fmts,params...)
3559     return 0
3560 }
3561 }
3562 // <a name="IME">Command Line IME</a>
3563 //----- MyIME
3564 var MyIMEVER = "MyIME/0.0.2";
3565 type RomKana struct {
3566     dic string // dictionary ID
3567     pat string // input pattern
3568     out string // output pattern
3569     hit int64 // count of hit and used
3570 }
3571 var dicents = 0
3572 var romkana [1024]RomKana
3573 var Romkan []RomKana
3574
3575 func isinDic(str string)(int){
3576     for i,v := range Romkan {
3577         if v.pat == str {
3578             return i
3579         }
3580     }
3581     return -1
3582 }
3583 const (
3584     DIC_COM_LOAD = "im"
3585     DIC_COM_DUMP = "s"
3586     DIC_COM_LIST = "ls"
3587     DIC_COM_ENA = "en"
3588     DIC_COM_DIS = "di"
3589 )
3590 func helpDic(argv []string){
3591     out := stderr
3592     cmd := ""
3593     if 0 < len(argv) { cmd = argv[0] }
3594     fprintf(out,"--- %v Usage\n",cmd)
3595     fprintf(out,"... Commands\n")
3596     fprintf(out,"... %v %<-3v [dicName] [dicURL] -- Import dictionary\n",cmd,DIC_COM_LOAD)
3597     fprintf(out,"... %v %<-3v [pattern] -- Search in dictionary\n",cmd,DIC_COM_DUMP)
3598     fprintf(out,"... %v %<-3v [dicName] -- List dictionaries\n",cmd,DIC_COM_LIST)
3599     fprintf(out,"... %v %<-3v [dicName] -- Disable dictionaries\n",cmd,DIC_COM_DIS)
3600     fprintf(out,"... %v %<-3v [dicName] -- Enable dictionaries\n",cmd,DIC_COM_ENA)
3601     fprintf(out,"... Keys ... %v\n","ESC can be used for '\\')
3602     fprintf(out,"... \\c -- Reverse the case of the last character\n",)
3603     fprintf(out,"... \\i -- Replace input with translated text\n",)
3604     fprintf(out,"... \\j -- On/Off translation mode\n",)
3605     fprintf(out,"... \\l -- Force Lower Case\n",)
3606     fprintf(out,"... \\u -- Force Upper Case (software CapsLock)\n",)
3607     fprintf(out,"... \\v -- Show translation actions\n",)
3608     fprintf(out,"... \\x -- Replace the last input character with it Hexa-Decimal\n",)
3609 }
3610 func xDic(argv[]string){
3611     if len(argv) <= 1 {
3612         helpDic(argv)
3613         return
3614     }
3615     argv = argv[1:]
3616     var debug = false
3617     var dump = false
3618     cmd := argv[0]
3619     argv = argv[1:]
3620     opt := ""
3621     arg := ""
3622
3623     if 0 < len(argv) {
3624         arg1 := argv[0]

```

```

3625     if arg1[0] == '-' {
3626         switch arg1 {
3627             default:
3628                 fmt.Printf("--Ed-- Unknown option(%v)\n",arg1)
3629                 return
3630             case "-v":
3631                 debug = true
3632             case "-d":
3633                 debug = true
3634         }
3635         opt = arg1
3636         argv = argv[1:]
3637     }
3638 }
3639
3640 dicName := ""
3641 dicURL := ""
3642 if 0 < len(argv) {
3643     arg = argv[0]
3644     argv = argv[1:]
3645 }
3646 if false {
3647     fprintf(stderr, "--Dd-- com(%v) opt(%v) arg(%v)\n", cmd, opt, arg)
3648 }
3649 if cmd == DIC_COM_LOAD {
3650     switch arg {
3651         default:
3652             dicName = "WorldDic"
3653             dicURL = WorldDic
3654             fprintf(stderr, "--Id-- default dictionary \"%v\"\n", dicName);
3655         case "jkl":
3656             dicName = "JKLJaDic"
3657             dicURL = JA_JKLDic
3658     }
3659     if debug {
3660         fprintf(stderr, "--Id-- %v URL=%v\n", dicName, dicURL);
3661     }
3662     dicv := strings.Split(dicURL, ",")
3663     if debug {
3664         fprintf(stderr, "--Id-- %v encoded data...\n", dicName)
3665         fprintf(stderr, "Type: %v\n", dicv[0])
3666         fprintf(stderr, "Body: %v\n", dicv[1])
3667         fprintf(stderr, "\n")
3668     }
3669     body, _ := base64.StdEncoding.DecodeString(dicv[1])
3670     if debug {
3671         fprintf(stderr, "--Id-- WorldDic %v text...\n", dicName)
3672         fprintf(stderr, "%v\n", string(body))
3673     }
3674     entv := strings.Split(string(body), "\n");
3675     fprintf(stderr, "--Id-- %v scan...\n", dicName);
3676     var added int = 0
3677     var dup int = 0
3678     for i, v := range entv {
3679         var pat string
3680         var out string
3681         fmt.Sscanf(v, "%s %s", &pat, &out)
3682         if len(pat) <= 0 {
3683             }else{
3684                 if 0 <= isinDic(pat) {
3685                     dup += 1
3686                     continue
3687                 }
3688                 romkana[dicents] = RomKana{dicName, pat, out, 0}
3689                 dicents += 1
3690                 added += 1
3691                 Romkan = append(Romkan, RomKana{dicName, pat, out, 0})
3692                 if debug {
3693                     fmt.Printf("[%3v]:[%2v]%-8v [%2v]%-8v\n",
3694                         i, len(pat), pat, len(out), out)
3695                 }
3696             }
3697         }
3698     fprintf(stderr, "--Id-- %v scan... %v added, %v dup. / %v total\n",
3699         dicName, added, dup, len(Romkan));
3700     // should sort by pattern length for complete match, for performance
3701     if debug {
3702         arg = "" // search pattern
3703         dump = true
3704     }
3705 }
3706 if cmd == DIC_COM_DUMP || dump {
3707     fprintf(stderr, "--Id-- %v dump... %v entries:\n", dicName, len(Romkan));
3708     var match = 0
3709     for i := 0; i < len(Romkan); i++ {
3710         dic := Romkan[i].dic
3711         pat := Romkan[i].pat
3712         out := Romkan[i].out
3713         if arg == "" || 0 <= strings.Index(pat, arg) || 0 <= strings.Index(out, arg) {
3714             fmt.Printf("\\\\%v\t%v [%2v]%-8v [%2v]%-8v\n",
3715                 i, dic, len(pat), pat, len(out), out)
3716             match += 1
3717         }
3718     }
3719     fprintf(stderr, "--Id-- %v matched %v / %v entries:\n", arg, match, len(Romkan));
3720 }
3721 }
3722 func loadDefaultDic(dic int){
3723     if( 0 < len(Romkan) ){
3724         return
3725     }
3726     //fprintf(stderr, "\r\n")
3727     xDic([]string{"dic", DIC_COM_LOAD});
3728     fprintf(stderr, "--Id-- Conguraturations!! WorldDic is now activated.\r\n")
3729     fprintf(stderr, "--Id-- enter \"dic\" command for help.\r\n")
3730 }
3731 func readDic()(int){
3732     /*
3733     var rk *os.File;
3734     var dic = "MyIME-dic.txt";
3735     //rk = fopen("romkana.txt", "r");
3736     //rk = fopen("JK-JA-morse-dic.txt", "r");
3737     rk = fopen(dic, "r");
3738     if( rk == NULL FP ){
3739         if( true ){
3740             fprintf(stderr, "--s-- Could not load %s\n", MyIMEVER, dic);
3741         }
3742         return -1;
3743     }
3744     if( true ){
3745         var di int;
3746         var line = make(StrBuff, 1024);
3747         var pat string
3748         var out string
3749         for di = 0; di < 1024; di++ {

```

```

3750         if( fgets(line,sizeof(line),rk) == NULLSP ){
3751             break;
3752         }
3753         fmt.Sscanf(string(line[0:strlen(line)]),"%s %s",&pat,&out);
3754         //sscanf(line,"%s %[^\r\n]",&pat,&out);
3755         romkana[di].pat = pat;
3756         romkana[di].out = out;
3757         //fprintf(stderr,"--Dd- %s\n",pat,out)
3758     }
3759     dicents += di
3760     if( false ){
3761         fprintf(stderr,"--s-- loaded romkana.txt [%d]\n",MyIMEVER,di);
3762         for di = 0; di < dicents; di++ {
3763             fprintf(stderr,
3764                 "%s %s\n",romkana[di].pat,romkana[di].out);
3765         }
3766     }
3767 }
3768 fclose(rk);
3769
3770 //romkana[dicents].pat = "//ddump"
3771 //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3772 */
3773 return 0;
3774 }
3775 func matchlen(stri string, pati string)(int){
3776     if strBegins(stri,pati) {
3777         return len(pati)
3778     }else{
3779         return 0
3780     }
3781 }
3782 func convs(src string)(string){
3783     var si int;
3784     var sx = len(src);
3785     var di int;
3786     var mi int;
3787     var dstb []byte
3788
3789     for si = 0; si < sx; { // search max. match from the position
3790         if strBegins(src[si:], "%x/") {
3791             // %x/integer/ // s/a/b/
3792             ix := strings.Index(src[si+3:], "/")
3793             if 0 < ix {
3794                 var iv int = 0
3795                 //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3796                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3797                 sval := fmt.Sprintf("%x", iv)
3798                 bval := []byte(sval)
3799                 dstb = append(dstb, bval...)
3800                 si = si+3+ix+1
3801                 continue
3802             }
3803         }
3804         if strBegins(src[si:], "%d/") {
3805             // %d/integer/ // s/a/b/
3806             ix := strings.Index(src[si+3:], "/")
3807             if 0 < ix {
3808                 var iv int = 0
3809                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3810                 sval := fmt.Sprintf("%d", iv)
3811                 bval := []byte(sval)
3812                 dstb = append(dstb, bval...)
3813                 si = si+3+ix+1
3814                 continue
3815             }
3816         }
3817         if strBegins(src[si:], "%t") {
3818             now := time.Now()
3819             if true {
3820                 date := now.Format(time.Stamp)
3821                 dstb = append(dstb, []byte(date)...)
3822                 si = si+3
3823             }
3824             continue
3825         }
3826         var maxlen int = 0;
3827         var len int;
3828         mi = -1;
3829         for di = 0; di < dicents; di++ {
3830             len = matchlen(src[si:], romkana[di].pat);
3831             if( maxlen < len ){
3832                 maxlen = len;
3833                 mi = di;
3834             }
3835         }
3836         if( 0 < maxlen ){
3837             out := romkana[mi].out;
3838             dstb = append(dstb, []byte(out)...);
3839             si += maxlen;
3840         }else{
3841             dstb = append(dstb, src[si])
3842             si += 1;
3843         }
3844     }
3845     return string(dstb)
3846 }
3847 func trans(src string)(int){
3848     dst := convs(src);
3849     xfprintf(stderr, dst);
3850     return 0;
3851 }
3852
3853 //----- LINEEDIT
3854 // "?" at the top of the line means searching history
3855
3856 const (
3857     GO_UP = 201
3858     GO_DOWN = 202
3859     GO_RIGHT = 203
3860     GO_LEFT = 204
3861     DEL_RIGHT = 205
3862     EV_TIMEOUT = 206
3863 )
3864
3865 // should return number of octets ready to be read immediately
3866 //fprintf(stderr, "\n--Select(%v %v)\n", err, r.Bits[0])
3867
3868 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
3869 // 2020-0827 GShell-0.2.3
3870 func FpollIn1(fp *os.File, usec int)(int){
3871     rdrv := syscall.FdSet {}
3872     fd := fp.Fd()
3873     bank := fd/32
3874     mask := int32(1 << fd)

```

```

3875   rdv.Bits[bank] = mask
3876   t := syscall.NsecToTimeval(int64(usec*1000))
3877   //n,err := syscall.Select(1,&rdv,nil,nil,&t) // spec. mismatch
3878   err := syscall.Select(1,&rdv,nil,nil,&t)
3879   if err == nil {
3880       if (rdv.Bits[bank] & mask) != 0 {
3881           return 1
3882       }else{
3883           return 0
3884       }
3885   }else{
3886       return -1
3887   }
3888 }
3889 func fgetcTimeout(fp *os.File,usec int)(int){
3890     ready := FpollIn1(fp,usec)
3891     if ready <= 0 {
3892         return EV_TIMEOUT
3893     }
3894     var buf [1]byte
3895     _,err := fp.Read(buf[0:1])
3896     if( err != nil ){
3897         return EOF;
3898     }else{
3899         return int(buf[0])
3900     }
3901 }
3902
3903 var TtyMaxCol = 72
3904 var EscTimeout = (100*1000)
3905 var (
3906     MODE_ShowMode bool
3907     romkanmode bool
3908     MODE_CapsLock bool // software CapsLock
3909     MODE_LowerLock bool // force lower-case character lock
3910     MODE_Vinsert int // visible insert mode, should be like "I" icon in X Window
3911     MODE_ViTrace bool // output newline before translation
3912 )
3913 type IInput struct {
3914     lno int
3915     lastlno int
3916     pch []int // input queue
3917     prompt string
3918     line string
3919     right string
3920     inJmode bool
3921     pinJmode bool
3922     waitingMeta string // waiting meta character
3923     lastCmd string
3924 }
3925 func (iin*IInput)Getc(timeoutUs int)(int){
3926     ch1 := EOF
3927     ch2 := EOF
3928     ch3 := EOF
3929     if( 0 < len(iin.pch) ){ // deQ
3930         ch1 = iin.pch[0]
3931         iin.pch = iin.pch[1:]
3932     }else{
3933         ch1 = fgetcTimeout(stdin,timeoutUs);
3934     }
3935     if( ch1 == 033 ){ // escape sequence
3936         ch2 = fgetcTimeout(stdin,EscTimeout);
3937         if( ch2 == EV_TIMEOUT ){
3938             }else{
3939             ch3 = fgetcTimeout(stdin,EscTimeout);
3940             if( ch3 == EV_TIMEOUT ){
3941                 iin.pch = append(iin.pch,ch2) // enQ
3942             }else{
3943                 switch( ch2 ){
3944                     default:
3945                         iin.pch = append(iin.pch,ch2) // enQ
3946                         iin.pch = append(iin.pch,ch3) // enQ
3947                     case '[':
3948                         switch( ch3 ){
3949                             case 'A': ch1 = GO_UP; // ^
3950                             case 'B': ch1 = GO_DOWN; // v
3951                             case 'C': ch1 = GO_RIGHT; // >
3952                             case 'D': ch1 = GO_LEFT; // <
3953                             case '3':
3954                                 ch4 := fgetcTimeout(stdin,EscTimeout);
3955                                 if( ch4 == '-' ){
3956                                     //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
3957                                     ch1 = DEL_RIGHT
3958                                 }
3959                             case '\\':
3960                                 //ch4 := fgetcTimeout(stdin,EscTimeout);
3961                                 //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
3962                                 switch( ch3 ){
3963                                     case '-': ch1 = DEL_RIGHT
3964                                 }
3965                             }
3966                     }
3967             }
3968         }
3969     }
3970     return ch1
3971 }
3972 func (inn*IInput)clearline(){
3973     var i int
3974     fprintf(stderr,"\r");
3975     // should be ANSI ESC sequence
3976     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
3977         fputc(' ',os.Stderr);
3978     }
3979     fprintf(stderr,"\r");
3980 }
3981 func (iin*IInput)Redraw(){
3982     redraw(iin,iin.lno,iin.line,iin.right)
3983 }
3984 func redraw(iin *IInput,lno int,line string,right string){
3985     inMeta := false
3986     showMode := ""
3987     showMeta := "" // visible Meta mode on the cursor position
3988     showLno := fmt.Sprintf("%d!", lno)
3989     InsertMark := "" // in visible insert mode
3990
3991     if 0 < len(iin.right) {
3992         InsertMark = " "
3993     }
3994
3995     if( 0 < len(iin.waitingMeta) ){
3996         inMeta = true
3997         if iin.waitingMeta[0] != 033 {
3998             showMeta = iin.waitingMeta
3999         }

```



```

4000 }
4001 if( romkanmode ){
4002 //romkanmark = " *";
4003 }else{
4004 //romkanmark = "";
4005 }
4006 if MODE_ShowMode {
4007 romkan := "---"
4008 inmeta := "-"
4009 inveri := ""
4010 if MODE_CapsLock {
4011 inmeta = "A"
4012 }
4013 if MODE_LowerLock {
4014 inmeta = "a"
4015 }
4016 if MODE_ViTrace {
4017 inveri = "v"
4018 }
4019 if romkanmode {
4020 romkan = "\343\201\202"
4021 if MODE_CapsLock {
4022 inmeta = "R"
4023 }else{
4024 inmeta = "r"
4025 }
4026 }
4027 if inMeta {
4028 inmeta = "\\\"
4029 }
4030 showMode = "["+romkan+inmeta+inveri+"]";
4031 }
4032 Pre := "\r" + showMode + showLino
4033 Output := ""
4034 Left := ""
4035 Right := ""
4036 if romkanmode {
4037 Left = convs(line)
4038 Right = InsertMark+convs(right)
4039 }else{
4040 Left = line
4041 Right = InsertMark+right
4042 }
4043 Output = Pre+Left
4044 if MODE_ViTrace {
4045 Output += iin.LastCmd
4046 }
4047 Output += showMeta+Right
4048 for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4049 Output += " "
4050 // should be ANSI ESC sequence
4051 // not necessary just after newline
4052 }
4053 Output += Pre+Left+showMeta // to set the cursor to the current input position
4054 fprintf(stderr,"%s",Output)
4055 }
4056 if MODE_ViTrace {
4057 if 0 < len(iin.LastCmd) {
4058 iin.LastCmd = ""
4059 fprintf(stderr,"\r\n")
4060 }
4061 }
4062 }
4063 func delHeadChar(str string)(rline string,head string){
4064 ,clen := utf8.DecodeRune([]byte(str))
4065 head = string(str[0:clen])
4066 return str[clen:],head
4067 }
4068 func delTailChar(str string)(rline string, last string){
4069 var i = 0
4070 var clen = 0
4071 for {
4072 ,siz := utf8.DecodeRune([]byte(str)[i:])
4073 if siz <= 0 { break }
4074 clen = siz
4075 i += siz
4076 }
4077 last = str[len(str)-clen:]
4078 return str[0:len(str)-clen],last
4079 }
4080 }
4081 // 3> for output and history
4082 // 4> for keylog?
4083 // <a name="getline">Command Line Editor</a>
4084 func xgetline(lno int, prevline string, gsh*GshContext)(string){
4085 var iin IInput
4086 iin.lastlno = lno
4087 iin.lno = lno
4088 }
4089 if( isatty(0) == 0 ){
4090 if( sfgets(&iin.line,LINESIZE,stdin) == NULL ){
4091 iin.line = "exit\n";
4092 }else{
4093 }
4094 return iin.line
4095 }
4096 if( true ){
4097 //var pts string;
4098 //pts = ptsname(0);
4099 //pts = ttyname(0);
4100 //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
4101 }
4102 if( false ){
4103 fprintf(stderr,"! ");
4104 fflush(stderr);
4105 sfgets(&iin.line,LINESIZE,stdin);
4106 return iin.line
4107 }
4108 system("/bin/stty -echo -icanon");
4109 xline := iin.xgetline1(prevline,gsh)
4110 system("/bin/stty echo sane");
4111 return xline
4112 }
4113 func (iin*IInput)Translate(cmdch int){
4114 romkanmode = !romkanmode;
4115 if MODE_ViTrace {
4116 fprintf(stderr,"%v\r\n",string(cmdch));
4117 }else
4118 if( cmdch == 'J' ){
4119 fprintf(stderr,"J\r\n");
4120 iin.inJmode = true
4121 }
4122 iin.Redraw();
4123 loadDefaultDic(cmdch);
4124 iin.Redraw();

```

```

4125 }
4126 func (iin*IInput)Replace(cmdch int){
4127     iin.LastCmd = fmt.Sprintf("\%v",string(cmdch))
4128     iin.Redraw();
4129     loadDefaultDic(cmdch);
4130     dst := convs(iin.line+iin.right);
4131     iin.line = dst
4132     iin.right = ""
4133     if( cmdch == 'I' ){
4134         fprintf(stderr,"I\r\n");
4135         iin.inJmode = true
4136     }
4137     iin.Redraw();
4138 }
4139 func (iin*IInput)xgetline(prevline string, gsh*GshContext)(string){
4140     var ch int;
4141     iin.Redraw();
4142     for {
4143         iin.pinJmode = iin.inJmode
4144         iin.inJmode = false
4145
4146         ch = iin.Getc(1000*1000)
4147         //fprintf(stderr,"A[%02X]\n",ch);
4148         if( ch == '\\ ' || ch == 033 ){
4149             MODE_ShowMode = true
4150             metach := ch
4151             iin.waitingMeta = string(ch)
4152             iin.Redraw();
4153             // set cursor //fprintf(stderr,"???\b\b\b")
4154             ch = fgetcTimeout(stdin,2000*1000)
4155             // reset cursor
4156             iin.waitingMeta = ""
4157
4158             cmdch := ch
4159             if( ch == EV_TIMEOUT ){
4160                 if metach == 033 {
4161                     continue
4162                 }
4163                 ch = metach
4164             }else
4165             if( ch == 'j' || ch == 'J' ){
4166                 iin.Translate(cmdch);
4167                 continue
4168             }else
4169             if( ch == 'i' || ch == 'I' ){
4170                 iin.Replace(cmdch);
4171                 continue
4172             }else
4173             if( ch == 'l' || ch == 'L' ){
4174                 MODE_LowerLock = IMODE_LowerLock
4175                 MODE_CapsLock = false
4176                 if MODE_ViTrace {
4177                     fprintf(stderr,"%v\r\n",string(cmdch));
4178                 }
4179                 iin.Redraw();
4180                 continue
4181             }else
4182             if( ch == 'u' || ch == 'U' ){
4183                 MODE_CapsLock = IMODE_CapsLock
4184                 MODE_LowerLock = false
4185                 if MODE_ViTrace {
4186                     fprintf(stderr,"%v\r\n",string(cmdch));
4187                 }
4188                 iin.Redraw();
4189                 continue
4190             }else
4191             if( ch == 'v' || ch == 'V' ){
4192                 MODE_ViTrace = IMODE_ViTrace
4193                 if MODE_ViTrace {
4194                     fprintf(stderr,"%v\r\n",string(cmdch));
4195                 }
4196                 iin.Redraw();
4197                 continue
4198             }else
4199             if( ch == 'c' || ch == 'C' ){
4200                 if 0 < len(iin.line) {
4201                     xline,tail := delTailChar(iin.line)
4202                     if len([]byte(tail)) == 1 {
4203                         ch = int(tail[0])
4204                         if( 'a' <= ch && ch <= 'z' ){
4205                             ch = ch + 'A'-'a'
4206                         }else
4207                         if( 'A' <= ch && ch <= 'Z' ){
4208                             ch = ch + 'a'-'A'
4209                         }
4210                         iin.line = xline + string(ch)
4211                     }
4212                 }
4213                 if MODE_ViTrace {
4214                     fprintf(stderr,"%v\r\n",string(cmdch));
4215                 }
4216                 iin.Redraw();
4217                 continue
4218             }else{
4219                 iin.pch = append(iin.pch,ch) // push
4220                 ch = '\\ '
4221             }
4222         }
4223         switch( ch ){
4224             case 'P'-0x40: ch = GO_UP
4225             case 'N'-0x40: ch = GO_DOWN
4226             case 'B'-0x40: ch = GO_LEFT
4227             case 'F'-0x40: ch = GO_RIGHT
4228         }
4229         //fprintf(stderr,"B[%02X]\n",ch);
4230         switch( ch ){
4231             case 0:
4232                 continue;
4233
4234             case '\t':
4235                 iin.Replace('j');
4236                 continue
4237             case 'X'-0x40:
4238                 iin.Replace('j');
4239                 continue
4240
4241             case EV_TIMEOUT:
4242                 iin.Redraw();
4243                 if iin.pinJmode {
4244                     fprintf(stderr,"\\J\r\n")
4245                     iin.inJmode = true
4246                 }
4247                 continue
4248             case GO_UP:
4249                 if iin.lno == 1 {

```

```

4250         continue
4251     }
4252     cmd_ok := gsh.cmdStringInHistory(iin.lno-1)
4253     if ok {
4254         iin.line = cmd
4255         iin.right = ""
4256         iin.lno = iin.lno - 1
4257     }
4258     iin.Redraw();
4259     continue
4260 case GO_DOWN:
4261     cmd_ok := gsh.cmdStringInHistory(iin.lno+1)
4262     if ok {
4263         iin.line = cmd
4264         iin.right = ""
4265         iin.lno = iin.lno + 1
4266     }else{
4267         iin.line = ""
4268         iin.right = ""
4269         if iin.lno == iin.lastlno-1 {
4270             iin.lno = iin.lno + 1
4271         }
4272     }
4273     iin.Redraw();
4274     continue
4275 case GO_LEFT:
4276     if 0 < len(iin.line) {
4277         xline,tail := delTailChar(iin.line)
4278         iin.line = xline
4279         iin.right = tail + iin.right
4280     }
4281     iin.Redraw();
4282     continue;
4283 case GO_RIGHT:
4284     if( 0 < len(iin.right) && iin.right[0] != 0 ){
4285         xright,head := delHeadChar(iin.right)
4286         iin.right = xright
4287         iin.line += head
4288     }
4289     iin.Redraw();
4290     continue;
4291 case EOF:
4292     goto EXIT;
4293 case 'R'-0x40: // replace
4294     dst := convs(iin.line+iin.right);
4295     iin.line = dst
4296     iin.right = ""
4297     iin.Redraw();
4298     continue;
4299 case 'T'-0x40: // just show the result
4300     readDic();
4301     romkanmode = !romkanmode;
4302     iin.Redraw();
4303     continue;
4304 case 'L'-0x40:
4305     iin.Redraw();
4306     continue
4307 case 'K'-0x40:
4308     iin.right = ""
4309     iin.Redraw();
4310     continue
4311 case 'E'-0x40:
4312     iin.line += iin.right
4313     iin.right = ""
4314     iin.Redraw();
4315     continue
4316 case 'A'-0x40:
4317     iin.right = iin.line + iin.right
4318     iin.line = ""
4319     iin.Redraw();
4320     continue
4321 case 'U'-0x40:
4322     iin.line = ""
4323     iin.right = ""
4324     iin.clearline();
4325     iin.Redraw();
4326     continue;
4327 case DEL_RIGHT:
4328     if( 0 < len(iin.right) ){
4329         iin.right,_ = delHeadChar(iin.right)
4330         iin.Redraw();
4331     }
4332     continue;
4333 case 0x7F: // BS? not DEL
4334     if( 0 < len(iin.line) ){
4335         iin.line,_ = delTailChar(iin.line)
4336         iin.Redraw();
4337     }
4338     /*
4339     else
4340     if( 0 < len(iin.right) ){
4341         iin.right,_ = delHeadChar(iin.right)
4342         iin.Redraw();
4343     }
4344     */
4345     continue;
4346 case 'H'-0x40:
4347     if( 0 < len(iin.line) ){
4348         iin.line,_ = delTailChar(iin.line)
4349         iin.Redraw();
4350     }
4351     continue;
4352 }
4353 if( ch == '\n' || ch == '\r' ){
4354     iin.line += iin.right;
4355     iin.right = ""
4356     iin.Redraw();
4357     fputc(ch,stderr);
4358     break;
4359 }
4360 if MODE_CapsLock {
4361     if 'a' <= ch && ch <= 'z' {
4362         ch = ch+'A'-'a'
4363     }
4364 }
4365 if MODE_LowerLock {
4366     if 'A' <= ch && ch <= 'Z' {
4367         ch = ch+'a'-'A'
4368     }
4369 }
4370 iin.line += string(ch);
4371 iin.Redraw();
4372 }
4373 EXIT:
4374     return iin.line + iin.right;

```

```

4375 }
4376
4377 func getline_main(){
4378     line := xgetline(0,"",nil)
4379     fprintf(stderr,"%s\n",line);
4380 /*
4381     dp = strpbrk(line,"\r\n");
4382     if( dp != NULL ){
4383         *dp = 0;
4384     }
4385
4386     if( 0 ){
4387         fprintf(stderr,"\n(%d)\n",int(strlen(line)));
4388     }
4389     if( lseek(3,0,0) == 0 ){
4390         if( romkammode ){
4391             var buf [8*1024]byte;
4392             convs(line,buf);
4393             strcpy(line,buf);
4394         }
4395         write(3,line,strlen(line));
4396         ftruncate(3,lseek(3,0,SEEK_CUR));
4397         //fprintf(stderr,"outsize=%d\n", (int)lseek(3,0,SEEK_END));
4398         lseek(3,0,SEEK_SET);
4399         close(3);
4400     }else{
4401         fprintf(stderr,"\r\n gotline: ");
4402         trans(line);
4403         //printf("%s\n",line);
4404         printf("\n");
4405     }
4406 */
4407 }
4408 //== end ====== getline
4409
4410 //
4411 // $USERHOME/.gsh/
4412 //   gsh-rc.txt, or gsh-configure.txt
4413 //   gsh-history.txt
4414 //   gsh-aliases.txt // should be conditional?
4415 //
4416 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
4417     homedir,found := userHomeDir()
4418     if !found {
4419         fmt.Printf("--E-- You have no UserHomeDir\n")
4420         return true
4421     }
4422     gshhome := homedir + "/" + GSH_HOME
4423     _, err2 := os.Stat(gshhome)
4424     if err2 != nil {
4425         err3 := os.Mkdir(gshhome,0700)
4426         if err3 != nil {
4427             fmt.Printf("--E-- Could not Create %s (%s)\n",
4428                 gshhome,err3)
4429             return true
4430         }
4431         fmt.Printf("--I-- Created %s\n",gshhome)
4432     }
4433     gshCtx.GshHomeDir = gshhome
4434     return false
4435 }
4436 func setupGshContext()(GshContext,bool){
4437     gshPA := syscall.ProcAttr {
4438         "", // the staring directory
4439         os.Environ(), // environ[]
4440         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
4441         nil, // OS specific
4442     }
4443     cwd, _ := os.Getwd()
4444     gshCtx := GshContext {
4445         cwd, // StartDir
4446         "", // GetLine
4447         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
4448         gshPA,
4449         []GCommandHistory{}, //something for invokation?
4450         GCommandHistory{}, // CmdCurrent
4451         false,
4452         []int{},
4453         syscall.Rusage{},
4454         "", // GshHomeDir
4455         Ttyid(),
4456         false,
4457         false,
4458         []PluginInfo{},
4459         []string{},
4460         "",
4461         "v",
4462         ValueStack{},
4463         GServer("", ""), // LastServer
4464         "", // RSERV
4465         cwd, // RWD
4466         CheckSum{},
4467     }
4468     err := gshCtx.gshSetupHomedir()
4469     return gshCtx, err
4470 }
4471 func (gsh *GshContext)gshellh(gline string)(bool){
4472     ghist := gsh.CmdCurrent
4473     ghist.WorkDir, _ = os.Getwd()
4474     ghist.WorkDirX = len(gsh.ChdirHistory)-1
4475     //fmt.Printf("--D--ChdirHistory(%d)\n",len(gsh.ChdirHistory))
4476     ghist.StartAt = time.Now()
4477     rusagev1 := Getrusagev()
4478     gsh.CmdCurrent.FoundFile = []string{}
4479     fin := gsh.tgshellh(gline)
4480     rusagev2 := Getrusagev()
4481     ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
4482     ghist.EndAt = time.Now()
4483     ghist.CmdLine = gline
4484     ghist.FoundFile = gsh.CmdCurrent.FoundFile
4485
4486     /* record it but not show in list by default
4487     if len(gline) == 0 {
4488         continue
4489     }
4490     if gline == "hi" || gline == "history" { // don't record it
4491         continue
4492     }
4493     */
4494     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
4495     return fin
4496 }
4497 // <a name="main">Main loop</a>
4498 func script(gshCtxGiven *GshContext) (_ GshContext) {
4499     gshCtxBuf,err0 := setupGshContext()

```

```

4500     if err0 {
4501         return gshCtxBuf;
4502     }
4503     gshCtx := &gshCtxBuf
4504
4505     //fmt.Printf("--I-- GSH_HOME=%s\n", gshCtx.GshHomeDir)
4506     //resmap()
4507
4508     /*
4509     if false {
4510         gsh_getline, with_exgetline :=
4511             which("PATH", []string{"which", "gsh-getline", "-s"})
4512         if with_exgetline {
4513             gsh_getlinev[0] = toFullpath(gsh_getline[0])
4514             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
4515         }else{
4516             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
4517         }
4518     }
4519     */
4520
4521     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
4522     gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist0)
4523
4524     prevline := ""
4525     skipping := false
4526     for hix := len(gshCtx.CommandHistory); ; {
4527         gline := gshCtx.getline(hix, skipping, prevline)
4528         if skipping {
4529             if strings.Index(gline, "fi") == 0 {
4530                 fmt.Printf("fi\n");
4531                 skipping = false;
4532             }else{
4533                 //fmt.Printf("%s\n", gline);
4534             }
4535             continue
4536         }
4537         if strings.Index(gline, "if") == 0 {
4538             //fmt.Printf("--D-- if start: %s\n", gline);
4539             skipping = true;
4540             continue
4541         }
4542         if false {
4543             os.Stdout.Write([]byte("gotline:"))
4544             os.Stdout.Write([]byte(gline))
4545             os.Stdout.Write([]byte("\n"))
4546         }
4547         gline = strsubst(gshCtx, gline, true)
4548         if false {
4549             fmt.Printf("fmt.Printf %%v - %v\n", gline)
4550             fmt.Printf("fmt.Printf %%s - %s\n", gline)
4551             fmt.Printf("fmt.Printf %%x - %x\n", gline)
4552             fmt.Printf("fmt.Printf %%U - %U\n", gline)
4553             fmt.Printf("Stout.Write -")
4554             os.Stdout.Write([]byte(gline))
4555             fmt.Printf("\n")
4556         }
4557         /*
4558         // should be cared in substitution ?
4559         if 0 < len(gline) && gline[0] == '!' {
4560             xgline, set, err := searchHistory(gshCtx, gline)
4561             if err {
4562                 continue
4563             }
4564             if set {
4565                 // set the line in command line editor
4566             }
4567             gline = xgline
4568         }
4569         */
4570         fin := gshCtx.gshelllh(gline)
4571         if fin {
4572             break;
4573         }
4574         prevline = gline;
4575         hix++;
4576     }
4577     return *gshCtx
4578 }
4579 func main() {
4580     gshCtxBuf := GshContext{}
4581     gsh := &gshCtxBuf
4582     argv := os.Args
4583     if 1 < len(argv) {
4584         if isin("version", argv){
4585             gsh.showVersion(argv)
4586             return
4587         }
4588         comx := isinX("-c", argv)
4589         if 0 < comx {
4590             gshCtxBuf, err := setupGshContext()
4591             gsh := &gshCtxBuf
4592             if !err {
4593                 gsh.gshellv(argv[comx+1:])
4594             }
4595             return
4596         }
4597     }
4598     if 1 < len(argv) && isin("-s", argv) {
4599     }else{
4600         gsh.showVersion(append(argv, []string{"-l", "-a"}...))
4601     }
4602     script(nil)
4603     //gshCtx := script(nil)
4604     //gshellh(gshCtx, "time")
4605 }
4606 //</div></details>
4607 //<details id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
4608 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
4609 // - merged histories of multiple parallel gsh sessions
4610 // - alias as a function or macro
4611 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
4612 // - retrieval PATH of files by its type
4613 // - gsh as an IME with completion using history and file names as dictionaies
4614 // - gsh a scheduler in precise time of within a millisecond
4615 // - all commands have its subucomand after "---" symbol
4616 // - filename expansion by "-find" command
4617 // - history of ext code and output of each commoand
4618 // - "script" output for each command by pty-tee or telnet-tee
4619 // - $BULLETIN command in PATH to show the priority
4620 // - "?" symbol in the command (not as in arguments) shows help request
4621 // - searching command with wild card like: which ssh-*
4622 // - longformat prompt after long idle time (should dismiss by BS)
4623 // - customizing by building plugin and dynamically linking it
4624 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias

```

```

4625 // - "!" symbol should be used for negation, don't wast it just for job control
4626 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
4627 // - making canonical form of command at the start adding quotation or white spaces
4628 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
4629 // - name? or name! might be useful
4630 // - htar format - packing directory contents into a single html file using data scheme
4631 // - filepath substitution should be done by each command, especially in case of builtins
4632 // - @# substitution for the history of working directory, and @spec for more generic ones
4633 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
4634 // - GSH_PATH for plugins
4635 // - standard command output: list of data with name, size, resource usage, modified time
4636 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
4637 // - wc word-count, grep match line count, ...
4638 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
4639 // - tailf-filename like tail -f filename, repeat close and open before read
4640 // - max. size and max. duration and timeout of (generated) data transfer
4641 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
4642 // - IME "?" at the top of the command line means searching history
4643 // - IME %d/0x10000/ %x/ffff/
4644 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
4645 // - gsh in WebAssembly
4646 // - gsh as a HTTP server of online-manual
4647 //---END--- ("-")/ITS more</div></details>
4648
4649 //<span class="gsh-golang-data">
4650 var WorldDic = //<span id="gsh-world-dic">
4651 "data:text/dic;base64,"+
4652 "Ly8gTK1JTUUvMk4wLjEg6L6e5pu4ICgyMDIwLTA4MThkQkpzZWthaSDkuJbnlyWka28g44GT"+
4653 "Cm5uIOOCkwpuaSDjgasKY2hpIOOBopQaSDjgaEKaEG44GvCnNlIOOBmwrYSDjgYsKaSDj"+
4654 "gYQK";
4655 //</span>
4656 var JA_KJLDic = //<span id="gsh-ja-jkl-dic">
4657 "data:text/dic;base64,"+
4658 "Ly92ZmJscU15SU1FamRyY2ptb3JzZWpKQWpKS0woMjYyMGowODE5KSheLV4pL1NhdG94SVRT"+
4659 "CmtqampRbGtqa2tsa2psIOS4lueVJApqamtqamwJ44GCmtqbAnjgYQKa2tqbAnjgYkamtq"+
4660 "amwJ44GIcmtqa2trbAnjgYKa2pra2wJ44GLCmpamtrbAnjgY0Ka2trawmJ44GPCmpampramps"+
4661 "CeOBkOpqampqBAnjgZMKamtqa2psCeOB1Opqamtqa2wJ44GXcmqamtqbAnjgZkKa2pqamts"+
4662 "CeOBmwpqampRbAnjgZ0KamtscCeOBnwpRa2prbAnjgaEKa2pqa2wJ44GKcmtqa2pqbAnjgaYK"+
4663 "a2tqa2tsCeOBgApramtsCeOBgqpa2prbAnjgasKa2tra2wJ44Gscmpqa2psCeOBpOpra2pq"+
4664 "banjga4Kamtra2wJ44GvCmpqa2tqbAnjgIKampra2wJ44G1CmtscCeOBuApra2tsCeOBuwpq"+
4665 "a2tqbAnjgBaKa2qa2psCeOBvwpqBAnjgoAKamtra2psCeOCgOpqa2tqa2wJ44KCCmtqamwJ"+
4666 "44KECmpRa2pqbAnjgYkampsCeOCiApra2tsCeOC1OpqamtsCeOCiOpqa2pqa2wJ44KLCmpq"+
4667 "amwJ44KCCmtqa2psCeOCjOpqa2psCeOCjwpramramwJ44KCCmtqamtrbAnjgpEKa2pqa2wJ"+
4668 "44KSCmtqa2prbAnjgpMKa2pqa2psCeODvApra2wJ44KbCmtramprbAnjgpwKa2pramtqbAnj"+
4669 "gIEK";
4670 //</span>
4671 //</span>
4672 /*
4673 <details id="references"><summary>References</summary><div class="gsh-src">
4674 <p>
4675 <a href="https://golang.org">The Go Programming Language</a>
4676 <iframe src="https://golang.org" width="100%" height="300"></iframe>
4677
4678 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
4679 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
4680 CSS
4681 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/Selectors">Selectors</a>
4682 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
4683 HTTP
4684 JavaScript:
4685 ..
4686 </p>
4687 </div></details>
4688 */
4689 /*
4690 <details id="html-src" onclick="frame_open();"><summary>Raw Source</summary><div>
4691
4692 <!-- h2>The full of this HTML including the Go code is here.</h2 -->
4693 <details id="gsh-whole-view"><summary>Whole file</summary>
4694 <a name="whole-src-view"></a>
4695 <span id="src-frame"></span><!-- a window to show source code -->
4696 </details>
4697
4698 <details id="gsh-style-frame" onclick="fill_CSSView()"><summary>CSS part</summary>
4699 <a name="style-src-view"></a>
4700 <span id="gsh-style-view"></span>
4701 </details>
4702
4703 <details id="gsh-script-frame" onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
4704 <a name="script-src-view"></a>
4705 <span id="gsh-script-view"></span>
4706 </details>
4707
4708 <details id="gsh-data-frame" onclick="fill_DataView()"><summary>Builtin data part</summary>
4709 <a name="gsh-data-frame"></a>
4710 <span id="gsh-data-view"></span>
4711 </details>
4712 </div></details>
4713 */
4714 */
4715 /*
4716 <div id="gsh-footer" style=""></div><!-- ----- END-OF-VISIBLE-PART ----- -->
4717
4718
4719 <style id="gsh-style-def">
4720 //body {display:none;}
4721 .gsh-link{color:green;}
4722 #gsh {border-width:1;margin:0;padding:0;}
4723 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
4724 #gsh header{height:100px;}
4725 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
4726 #gsh-menu{font-size:14pt;color:#f88;}
4727 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
4728 #gsh note{color:#000;font-size:10pt;}
4729 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
4730 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
4731 #gsh summary{font-size:16pt;color:#fff;background-color:#8af,height:30px;}
4732 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
4733 #gsh a{color:#24a;}
4734 #gsh a[name]{color:#24a;font-size:16pt;}
4735 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
4736 #gsh .gsh-src{background-color:#faffff;color:#223;}
4737 #gsh-src-src{spellcheck:false}
4738 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
4739 #src-frame-textarea{background-color:#faffff;color:#223;}
4740 .gsh-code {white-space:pre;font-family:monospace !import;}
4741 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
4742 .gsh-golang-data {display:none;}
4743 #gsh-winId {color:#000;font-size:14pt;}
4744
4745 #gsh-statement {font-size:11pt;background-color:#fff;font-family:Georgia;}
4746 #gsh-statement {color:#000;background-color:#fff !import;}
4747 #gsh-statement h2{color:#000;background-color:#fff !import;}
4748 #gsh-statement details{color:#000;background-color:#fff;font-family:Georgia;}
4749 #gsh-statement p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}

```

```

4750 #gsh-statement address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
4751
4752 @media print {
4753 #gsh_pre{font-size:11pt !important;}
4754 }
4755 </style>
4756
4757 <!--
4758 // Logo image should be drawn by JavaScript from a meta-font.
4759 // CSS seems not follow line-splitted URL
4760 -->
4761 <script id="gsh-data">
4762 //GshLogo="QR-ITS-more.jp.png"
4763 GshLogo="data:image/png;base64,\
4764 iVBORw0KGgoAAAANSUHEGAAQAQAAB/CAYAAADv3f4AAAAAXNSR0IArs4c6QAAAHh1WELm\
4765 TU0AKgAAAABAAEAAUAAABAAAPgEBAUAAABAAABAAQgEAAUAAABAAABAAIAAIpdaQAAAAB\
4766 AAAATgAAAAAABAAAAAQAABAAQAAABAAQAAQADAAAAQAABAAcAgAAAAQAABAAQAAQAAQAAQAA\
4767 AAAAQAABAAH8AAAAAYx1BhgAAAlwSfLzAAALeWAAcMBAJgcGAAAF3RJREfUEAhtnQUuFfNWZ\
4768 x+tt7ukZ3iCgg0/jY60sb8WgMzAvn7uG4+biSTR7YnQXQdQPCkCj2aWnLd2MS1rkeUaPnoCdu\
4769 4iuXw7jriY5Z0DGMf2VqIBeISggCoIMma+mu+vu//ZMD9Uldau6a2Ubyv91Gkrq3vdx6/q\
4770 fnXdx8tBA85IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4771 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4772 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4773 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4774 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4775 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4776 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4777 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4778 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4779 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4780 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4781 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4782 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4783 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4784 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4785 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4786 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4787 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4788 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4789 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4790 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4791 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4792 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4793 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4794 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4795 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4796 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4797 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4798 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4799 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4800 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4801 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4802 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4803 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4804 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4805 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4806 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4807 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4808 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4809 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4810 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4811 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4812 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4813 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4814 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4815 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4816 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4817 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4818 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4819 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4820 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4821 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4822 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4823 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4824 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4825 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4826 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4827 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4828 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4829 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4830 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4831 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4832 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4833 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4834 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4835 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4836 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4837 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4838 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4839 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4840 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4841 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4842 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4843 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4844 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4845 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4846 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4847 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4848 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4849 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4850 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4851 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4852 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4853 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4854 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4855 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4856 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4857 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4858 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4859 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4860 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4861 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4862 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4863 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4864 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4865 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4866 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4867 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4868 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4869 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4870 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4871 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4872 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4873 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4874 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\

```



```

5000 }
5001 }
5002 function html_pure(e){
5003     if( e.innerHTML == "Pure" ){
5004         document.getElementById('gsh').style.display=true
5005         //document.style.display = false
5006         e.innerHTML = "Unpure"
5007     }else{
5008         document.getElementById('gsh').style.display=false
5009         //document.style.display = true
5010         e.innerHTML = "Pure"
5011     }
5012 }
5013
5014 var bannerIsStopping = false
5015 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5016 function shiftBG(){
5017     bannerIsStopping = !bannerIsStopping
5018     bannerStyle.backgroundColor = "0 0";
5019 }
5020 // status should be inherited on Window Fork(), so use the status in DOM
5021 function html_stop(e,toggle){
5022     if( toggle ){
5023         if( e.innerHTML == "Stop" ){
5024             bannerIsStopping = true
5025             e.innerHTML = "Start"
5026         }else{
5027             bannerIsStopping = false
5028             e.innerHTML = "Stop"
5029         }
5030     }else{
5031         // update JavaScript variable from DOM status
5032         if( e.innerHTML == "Stop" ){ // shown if it's running
5033             bannerIsStopping = false
5034         }else{
5035             bannerIsStopping = true
5036         }
5037     }
5038 }
5039 html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
5040 //html_stop(bannerElem(),false) // onInit.
5041
5042 //https://www.w3schools.com/jsref/met_win_setinterval.asp
5043 function shiftBanner(){
5044     var now = new Date().getTime();
5045     //console.log("now="+now%10)
5046     if( !bannerIsStopping ){
5047         bannerStyle.backgroundColor = ((now/10)%100000)+" 0";
5048     }
5049 }
5050 setInterval(shiftBanner,10); // onInit.
5051
5052 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
5053 // from embedded html to standalone page
5054 var MyChildren = 0
5055 function html_fork(){
5056     MyChildren += 1
5057     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5058     newwin = window.open("",WinId,"");
5059     src = document.getElementById('gsh');
5060     newwin.document.write("<"+"html">\n");
5061     newwin.document.write("<"+"span id=\"gsh\">");
5062     newwin.document.write(src.innerHTML);
5063     newwin.document.write("<"+"span">"+"/html">\n"); // gsh span
5064     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5065     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5066     newwin.document.close();
5067     newwin.focus();
5068 }
5069 function html_close(){
5070     window.close()
5071 }
5072 function win_jump(win){
5073     //win = window.top;
5074     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
5075     if( win == null ){
5076         console.log("jump to window.opener("+win+") (Error)\n")
5077     }else{
5078         console.log("jump to window.opener("+win+")\n")
5079         win.focus();
5080     }
5081 }
5082
5083 // source code view
5084 function frame_close(){
5085     srcframe = document.getElementById("src-frame");
5086     srcframe.innerHTML = "";
5087     //srcframe.style.cols = 1;
5088     srcframe.style.rows = 1;
5089     srcframe.style.height = 0;
5090     srcframe.style.display = false;
5091     src = document.getElementById("src-frame-textarea");
5092     src.innerHTML = ""
5093     //src.cols = 0
5094     src.rows = 0
5095     src.display = false
5096     //alert("--closed--")
5097 }
5098 //<!-- | <span onclick="html_view();">Source</span> -->
5099 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
5100 //<!--| <span>Download</span> -->
5101 function frame_open(){
5102     oldsrc = document.getElementById("GENSRC");
5103     if( oldsrc != null ){
5104         //alert("--I--(erasing old text)")
5105         oldsrc.innerHTML = "";
5106         return
5107     }else{
5108         //alert("--I--(no old text)")
5109     }
5110     banner = document.getElementById('gsh-banner').style.backgroundImage;
5111     footer = document.getElementById('gsh-footer').style.backgroundImage;
5112     document.getElementById('gsh-banner').style.backgroundImage = "";
5113     document.getElementById('gsh-banner').style.backgroundColor = "";
5114     document.getElementById('gsh-footer').style.backgroundImage = "";
5115
5116     src = document.getElementById("gsh");
5117     srcframe = document.getElementById("src-frame");
5118     srcframe.innerHTML = ""
5119     + "<"+"cite id=\"GENSRC\">\n"
5120     + "<"+"style>\n"
5121     + "#GENSRC textarea{tab-size:4;}\n"
5122     + "#GENSRC textarea{-o-tab-size:4;}\n"
5123     + "#GENSRC textarea{-moz-tab-size:4;}\n"
5124     + "#GENSRC textarea{spellcheck:false;}\n"

```

```
5125 + "</+>style>\n"
5126 + "<+>textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">"
5127 + "<+>html>\n" // lost preamble text
5128 + "<+>span id="gsh">" // lost preamble text
5129 + src.innerHTML
5130 + "<+>/span><+>/html>\n" // lost trail text
5131 + "<+>textarea>\n"
5132 + "<+>cite>!-- GENSRC -->\n";
5133
5134 //srcframe.style.cols = 80;
5135 //srcframe.style.rows = 80;
5136
5137 document.getElementById('gsh-banner').style.backgroundImage = banner;
5138 document.getElementById('gsh-footer').style.backgroundImage = footer;
5139 }
5140 function fill_CSSView(){
5141 part = document.getElementById('gsh-style-def')
5142 view = document.getElementById('gsh-style-view')
5143 view.innerHTML = ""
5144 + "<+>textarea cols=100 rows=20 class="gsh-code">"
5145 + part.innerHTML
5146 + "<+>/textarea>"
5147 }
5148 function fill_JavaScriptView(){
5149 jspart = document.getElementById('gsh-script')
5150 view = document.getElementById('gsh-script-view')
5151 view.innerHTML = ""
5152 + "<+>textarea cols=100 rows=20 class="gsh-code">"
5153 + jspart.innerHTML
5154 + "<+>/textarea>"
5155 }
5156 function fill_DataView(){
5157 part = document.getElementById('gsh-data')
5158 view = document.getElementById('gsh-data-view')
5159 view.innerHTML = ""
5160 + "<+>textarea cols=100 rows=20 class="gsh-code">"
5161 + part.innerHTML
5162 + "<+>/textarea>"
5163 }
5164 function jumpto_StyleView(){
5165 jsview = document.getElementById('html-src')
5166 jsview.open = true
5167 jsview = document.getElementById('gsh-style-frame')
5168 jsview.open = true
5169 fill_CSSView()
5170 }
5171 function jumpto_JavaScriptView(){
5172 jsview = document.getElementById('html-src')
5173 jsview.open = true
5174 jsview = document.getElementById('gsh-script-frame')
5175 jsview.open = true
5176 fill_JavaScriptView()
5177 }
5178 function jumpto_DataView(){
5179 jsview = document.getElementById('html-src')
5180 jsview.open = true
5181 jsview = document.getElementById('gsh-data-frame')
5182 jsview.open = true
5183 fill_DataView()
5184 }
5185 function jumpto_WholeView(){
5186 jsview = document.getElementById('html-src')
5187 jsview.open = true
5188 jsview = document.getElementById('gsh-whole-view')
5189 jsview.open = true
5190 frame_open()
5191 }
5192 function html_view(){
5193 html_stop();
5194
5195 banner = document.getElementById('gsh-banner').style.backgroundImage;
5196 footer = document.getElementById('gsh-footer').style.backgroundImage;
5197 document.getElementById('gsh-banner').style.backgroundImage = "";
5198 document.getElementById('gsh-banner').style.backgroundPosition = "";
5199 document.getElementById('gsh-footer').style.backgroundImage = "";
5200
5201 //srcwin = window.open("", "CodeView2", "");
5202 srcwin = window.open("", "", "");
5203 srcwin.document.write("<span id="gsh">\n");
5204
5205 src = document.getElementById("gsh");
5206 srcwin.document.write("<+>style>\n");
5207 srcwin.document.write("textarea{tab-size:4;}\n");
5208 srcwin.document.write("textarea{-o-tab-size:4;}\n");
5209 srcwin.document.write("textarea{-moz-tab-size:4;}\n");
5210 srcwin.document.write("</style>\n");
5211 srcwin.document.write("<h2>\n");
5212 srcwin.document.write("<+>span onclick="window.close();>Close</span> | \n");
5213 //srcwin.document.write("<+>span onclick="html_stop();>Run</span>\n");
5214 srcwin.document.write("</h2>\n");
5215 srcwin.document.write("<textarea id="gsh-src-src" cols=100 rows=60>");
5216 srcwin.document.write("<+>html>\n");
5217 srcwin.document.write("<+>span id="gsh">");
5218 srcwin.document.write(src.innerHTML);
5219 srcwin.document.write("<+>/span><+>/html>\n");
5220 srcwin.document.write("<+>textarea>\n");
5221
5222 document.getElementById('gsh-banner').style.backgroundImage = banner;
5223 document.getElementById('gsh-footer').style.backgroundImage = footer
5224
5225 sty = document.getElementById("gsh-style-def");
5226 srcwin.document.write("<+>style>\n");
5227 srcwin.document.write(sty.innerHTML);
5228 srcwin.document.write("<+>/style>\n");
5229
5230 run = document.getElementById("gsh-script");
5231 srcwin.document.write("<+>script>\n");
5232 srcwin.document.write(run.innerHTML);
5233 srcwin.document.write("<+>/script>\n");
5234
5235 srcwin.document.write("<+>/span><+>/html>\n"); // gsh span
5236 srcwin.document.close();
5237 srcwin.focus();
5238 }
5239 </script>
5240 -->
5241 *//<br></span></details></html>
5242
```