

```

1 //<html><details open><summary>GShell-0.2.8-HtmlArchive</summary>
2 /*<span id="gsh">
3 <meta charset="UTF-8">
4 <meta name="viewport" content="width=device-width, initial-scale=1.0">
5 <link rel="icon" id="gsh-iconurl" href=""/><!-- place holder -->
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 0.2.8 // 2020-09-01 // SatoxITS</note></div>
9 </header>
10 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
11 <p>
12 <note>
13 It is a shell for myself, by myself, of myself. --SatoxITS(^-^ )
14 </note>
15 </p>
16 <span id="gsh-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" class="gsh-document"><summary>Statement</summary><p>
27 <h3>Fun to create a shell</h3>
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 open id="gsh-features" class="gsh-document"><summary>Features</summary><p>
54 </p>
55 <h3>Vi compatible command line editor</h3>
56 <p>
57 The command line of GShell can be edited with commands compatible with
58 <a href="https://www.washington.edu/computing/unix/vi.html">vi</a>.
59 As in vi, you can enter <i><b>command mode</b></i> by <b>ESC</b> key,
60 then move around in the history by <b>code>j k / ? n N</code>,
61 or within the current line by <b>code>l h f w b o $ %</code> or so.
62 </p>
63 </details>
64 */
65 /*
66 <details id="gsh-gindex">
67 <summary>Index</summary><div class="gsh-src">
68 Documents
69 <span class="gsh-link" onclick="jumpto_JavaScriptView();">Command summary</span>
70 Go lang part<span class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
71 Package structures
72 <a href="#import">import</a>
73 <a href="#struct">struct</a>
74 Main functions
75 <a href="#comexpansion">str-expansion</a> // macro processor
76 <a href="#finder">finder</a> // builtin find + du
77 <a href="#grep">grep</a> // builtin grep + wc + cksun + ...
78 <a href="#plugin">plugin</a> // plugin commands
79 <a href="#ex-commands">system</a> // external commands
80 <a href="#builtin">builtin</a> // builtin commands
81 <a href="#network">network</a> // socket handler
82 <a href="#remote-sh">remote-sh</a> // remote shell
83 <a href="#redirect">redirect</a> // StdIn/Out redirection
84 <a href="#history">history</a> // command history
85 <a href="#rusage">rusage</a> // resource usage
86 <a href="#encode">encode</a> // encode / decode
87 <a href="#IME">IME</a> // command line IME
88 <a href="#getline">getline</a> // line editor
89 <a href="#scanf">scanf</a> // string decomposer
90 <a href="#interpreter">interpreter</a> // command interpreter
91 <a href="#main">main</a>
92 </span>
93 JavaScript part
94 <a href="#script-src-view" class="gsh-link" onclick="jumpto_JavaScriptView();">Source</a>
95 <a href="#gsh-data-frame" class="gsh-link" onclick="jumpto_DataView();">Builtin data</a>
96 CSS part
97 <a href="#style-src-view" class="gsh-link" onclick="jumpto_StyleView();">Source</a>
98 References
99 <a href="#" class="gsh-link" onclick="jumpto_WholeView();">Internal</a>
100 <a href="#gsh-reference" class="gsh-link" onclick="jumpto_ReferenceView();">External</a>
101 Whole parts
102 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Source</a>
103 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Download</a>
104 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Dump</a>
105
106 </div>
107 </details>
108 */
109 //<details id="gsh-gocode">
110 //<summary>Go Source</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
111 // gsh - Go lang based Shell
112 // (c) 2020 ITS more Co., Ltd.
113 // 2020-0807 created by SatoxITS (sato@its-more.jp)
114
115 package main // gsh main
116 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
117 import (
118 "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
119 "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
120 "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
121 "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
122 "time" // <a href="https://golang.org/pkg/time/">time</a>
123 "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
124 "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>

```

```

125 "os" // <a href="https://golang.org/pkg/os/">os</a>
126 "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
127 "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
128 "net" // <a href="https://golang.org/pkg/net/">net</a>
129 "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
130 // "html" // <a href="https://golang.org/pkg/html/">html</a>
131 "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
132 "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
133 "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
134 "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
135 "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
136 // "gshdata" // gshell's logo and source code
137 "hash/crc32" // <a href="https://golang.org/pkg/unicode/hash/crc32/">crc32</a>
138 )
139 const (
140     NAME = "gsh"
141     VERSION = "0.2.8"
142     DATE = "2020-09-01"
143     AUTHOR = "SatoxITS(^-^)/"
144 )
145 var (
146     GSH_HOME = ".gsh" // under home directory
147     GSH_PORT = 9999
148     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
149     PROMPT = ">"
150     LINESIZE = (8*1024)
151     PATHSEP = ";" // should be ";" in Windows
152     DIRSEP = "/" // canbe \ in Windows
153 )
154
155 // -xX logging control
156 // --a-- all
157 // --i-- info.
158 // --D-- debug
159 // --T-- time and resource usage
160 // --W-- warning
161 // --E-- error
162 // --F-- fatal error
163 // --Xn- network
164
165 // <a name="struct">Structures</a>
166 type GCommandHistory struct {
167     StartAt time.Time // command line execution started at
168     EndAt time.Time // command line execution ended at
169     ResCode int // exit code of (external command)
170     CmdError error // error string
171     OutData *os.File // output of the command
172     FoundFile []string // output - result of unfind
173     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
174     CmdId int // maybe with identified with arguments or impact
175     // redirection commands should not be the CmdId
176     WorkDir string // working directory at start
177     WorkDirX int // index in ChdirHistory
178     CmdLine string // command line
179 }
180 type GChdirHistory struct {
181     Dir string
182     MovedAt time.Time
183     CmdIndex int
184 }
185 type CmdMode struct {
186     Background bool
187 }
188 type Event struct {
189     when time.Time
190     event int
191     evarg int64
192     CmdIndex int
193 }
194 var CmdIndex int
195 var Events []Event
196 type PluginInfo struct {
197     Spec *plugin.Plugin
198     Addr plugin.Symbol
199     Name string // maybe relative
200     Path string // this is in Plugin but hidden
201 }
202 type GServer struct {
203     host string
204     port string
205 }
206
207 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
208 const ( // SumType
209     SUM_ITEMS = 0x000001 // items count
210     SUM_SIZE = 0x000002 // data length (simply added)
211     SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
212     SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
213     // also envelope attributes like time stamp can be a part of digest
214     // hashed value of sizes or mod-date of files will be useful to detect changes
215
216     SUM_WORDS = 0x000010 // word count is a kind of digest
217     SUM_LINES = 0x000020 // line count is a kind of digest
218     SUM_SUM64 = 0x000040 // simple add of bytes, useful for human too
219
220     SUM_SUM32_BITS = 0x000100 // the number of true bits
221     SUM_SUM32_2BYTE = 0x000200 // 16bits words
222     SUM_SUM32_4BYTE = 0x000400 // 32bits words
223     SUM_SUM32_8BYTE = 0x000800 // 64bits words
224
225     SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
226     SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
227     SUM_UNIXFILE = 0x004000
228     SUM_CRCIEEE = 0x008000
229 )
230 type CheckSum struct {
231     Files int64 // the number of files (or data)
232     Size int64 // content size
233     Words int64 // word count
234     Lines int64 // line count
235     SumType int
236     Sum64 uint64
237     Crc32Table crc32.Table
238     Crc32Val uint32
239     Sum16 int
240     Ctime time.Time
241     Atime time.Time
242     Mtime time.Time
243     Start time.Time
244     Done time.Time
245     RusgAtStart [2]syscall.Rusage
246     RusgAtEnd [2]syscall.Rusage
247 }
248 type ValueStack [][]string
249 type GshContext struct {

```

```

250 StartDir    string // the current directory at the start
251 GetLine    string // gsh-getline command as a input line editor
252 ChdirHistory []ChdirHistory // the 1st entry is wd at the start
253 gshPA      syscall.ProcAttr
254 CommandHistory []CommandHistory
255 CmdCurrent  GCommandHistory
256 BackGround  bool
257 BackGroundJobs []int
258 LastRusage  syscall.Rusage
259 GshHomeDir  string
260 TerminalId  int
261 CmdTrace    bool // should be [map]
262 CmdTime     bool // should be [map]
263 PluginFuncs []PluginInfo
264 iValues     []string
265 iDelimiter  string // field separator of print out
266 iFormat     string // default print format (of integer)
267 iValStack  ValueStack
268 LastServer  GServer
269 RSERVER     string // [gsh://host[:port]]
270 RWD         string // remote (target, there) working directory
271 lastChecksum CheckSum
272 }
273
274 func nsleep(ns time.Duration){
275     time.Sleep(ns)
276 }
277 func usleep(ns time.Duration){
278     nsleep(ns*1000)
279 }
280 func msleep(ns time.Duration){
281     nsleep(ns*1000000)
282 }
283 func sleep(ns time.Duration){
284     nsleep(ns*1000000000)
285 }
286
287 func strBegins(str, pat string)(bool){
288     if len(pat) <= len(str){
289         yes := str[0:len(pat)] == pat
290         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat, yes)
291         return yes
292     }
293     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
294     return false
295 }
296 func isin(what string, list []string) bool {
297     for _, v := range list {
298         if v == what {
299             return true
300         }
301     }
302     return false
303 }
304 func isinX(what string, list[]string)(int){
305     for i,v := range list {
306         if v == what {
307             return i
308         }
309     }
310     return -1
311 }
312
313 func env(opts []string) {
314     env := os.Environ()
315     if isin("-s", opts){
316         sort.Slice(env, func(i,j int) bool {
317             return env[i] < env[j]
318         })
319     }
320     for _, v := range env {
321         fmt.Printf("%v\n",v)
322     }
323 }
324
325 // - rewriting should be context dependent
326 // - should postpone until the real point of evaluation
327 // - should rewrite only known notation of symbol
328 func scanInt(str string)(val int, leng int){
329     leng = -1
330     for i,ch := range str {
331         if '0' <= ch && ch <= '9' {
332             leng = i+1
333         }else{
334             break
335         }
336     }
337     if 0 < leng {
338         ival, _ := strconv.Atoi(str[0:leng])
339         return ival,leng
340     }else{
341         return 0,0
342     }
343 }
344 func substHistory(gshCtx *GshContext, str string, i int, rstr string)(leng int, rst string){
345     if len(str[i+1:]) == 0 {
346         return 0, rstr
347     }
348     hi := 0
349     histlen := len(gshCtx.CommandHistory)
350     if str[i+1] == '|' {
351         hi = histlen - 1
352         leng = 1
353     }else{
354         hi, leng = scanInt(str[i+1:])
355         if leng == 0 {
356             return 0, rstr
357         }
358         if hi < 0 {
359             hi = histlen + hi
360         }
361     }
362     if 0 <= hi && hi < histlen {
363         var ext byte
364         if 1 < len(str[i+leng:]) {
365             ext = str[i+leng:][1]
366         }
367         //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
368         if ext == 'f' {
369             leng += 1
370             xlist := []string{}
371             list := gshCtx.CommandHistory[hi].FoundFile
372             for _,v := range list {
373                 //list[i] = escapeWhiteSP(v)
374                 xlist = append(xlist,escapeWhiteSP(v))

```

```

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

```

```

500 if isin("-l",opts) || isin("-ls",opts) {
501     fi, err := os.Stat(path)
502     if err != nil {
503         fmt.Printf("----- ((%v))",err)
504     }else{
505         mod := fi.ModTime()
506         date := mod.Format(time.Stamp)
507         fmt.Printf("%v %v %s ",fi.Mode(),fi.Size(),date)
508     }
509 }
510 fmt.Printf("%s",path)
511 if isin("-sp",opts) {
512     fmt.Printf(" ")
513 }else
514 if ! isin("-n",opts) {
515     fmt.Printf("\n")
516 }
517 }
518 func userHomeDir()(string,bool){
519     /*
520     homedir,_ = os.UserHomeDir() // not implemented in older Golang
521     */
522     homedir,found := os.LookupEnv("HOME")
523     //fmt.Printf("--I-- HOME=%v\n",homedir,found)
524     if !found {
525         return "/tmp",found
526     }
527     return homedir,found
528 }
529 }
530 func toFullpath(path string) (fullpath string) {
531     if path[0] == '/' {
532         return path
533     }
534     pathv := strings.Split(path,DIRSEP)
535     switch {
536     case pathv[0] == ".":
537         pathv[0],_ = os.Getwd()
538     case pathv[0] == "..": // all ones should be interpreted
539         cwd,_ := os.Getwd()
540         ppathv := strings.Split(cwd,DIRSEP)
541         pathv[0] = strings.Join(ppathv,DIRSEP)
542     case pathv[0] == "-":
543         pathv[0],_ = userHomeDir()
544     default:
545         cwd,_ := os.Getwd()
546         pathv[0] = cwd + DIRSEP + pathv[0]
547     }
548     return strings.Join(pathv,DIRSEP)
549 }
550 }
551 func IsRegFile(path string)(bool){
552     fi, err := os.Stat(path)
553     if err == nil {
554         fm := fi.Mode()
555         return fm.IsRegular();
556     }
557     return false
558 }
559 }
560 // <a name="encode">Encode / Decode</a>
561 // <a href="https://golang.org/pkg/encoding/base64/#example_NewEncoder">Encoder</a>
562 func (gshCtx *GshContext)Enc(argv[]string){
563     file := os.Stdin
564     buff := make([]byte,LINESIZE)
565     li := 0
566     encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
567     for li = 0; ; li++ {
568         count, err := file.Read(buff)
569         if count <= 0 {
570             break
571         }
572         if err != nil {
573             break
574         }
575         encoder.Write(buff[0:count])
576     }
577     encoder.Close()
578 }
579 func (gshCtx *GshContext)Dec(argv[]string){
580     decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
581     li := 0
582     buff := make([]byte,LINESIZE)
583     for li = 0; ; li++ {
584         count, err := decoder.Read(buff)
585         if count <= 0 {
586             break
587         }
588         if err != nil {
589             break
590         }
591         os.Stdout.Write(buff[0:count])
592     }
593 }
594 // lnspl [N] [-crlf][-C \\\]
595 func (gshCtx *GshContext)SplitLine(argv[]string){
596     reader := bufio.NewReaderSize(os.Stdin,64*1024)
597     ni := 0
598     toi := 0
599     for ni = 0; ; ni++ {
600         line, err := reader.ReadString('\n')
601         if len(line) <= 0 {
602             if err != nil {
603                 fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d (%v)\n",ni,toi,err)
604                 break
605             }
606         }
607         off := 0
608         ilen := len(line)
609         remlen := len(line)
610         for oi := 0; 0 < remlen; oi++ {
611             olen := remlen
612             addnl := false
613             if 72 < olen {
614                 olen = 72
615                 addnl = true
616             }
617             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d\n",
618                 toi,ni,oi,off,olen,remlen,ilen)
619             toi += 1
620             os.Stdout.Write([]byte(line[0:olen]))
621             if addnl {
622                 //os.Stdout.Write([]byte("\r\n"))
623                 os.Stdout.Write([]byte("\n"))
624                 os.Stdout.Write([]byte("\n"))

```

```

625     }
626     line = line[olen:]
627     off += olen
628     remlen -= olen
629 }
630 }
631 fmt.Fprintf(os.Stderr, "--I-- lnspp %d to %d\n", ni, toi)
632 }
633 }
634 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
635 // 1 0000 0100 1100 0001 0001 1101 1011 0111
636 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
637 var CRC32IEEE uint32 = uint32(0xEDB88320)
638 func byteCRC32add(crc uint32, str []byte, len uint64)(uint32){
639     var i uint64
640     for i = 0; i < len; i++ {
641         var oct = str[i]
642         for bi := 0; bi < 8; bi++ {
643             ovf1 := (crc & 0x80000000) != 0
644             ovf2 := (oct & 0x80) != 0
645             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
646             oct <<= 1
647             crc <<= 1
648             if ovf { crc ^= CRC32UNIX }
649         }
650     }
651     return crc;
652 }
653 func byteCRC32end(crc uint32, len uint64)(uint32){
654     var slen = make([]byte,4)
655     var li = 0
656     for li = 0; li < 4; {
657         slen[li] = byte(len)
658         li += 1
659         len >>= 8
660         if( len == 0 ){
661             break
662         }
663     }
664     crc = byteCRC32add(crc, slen, uint64(li))
665     crc ^= 0xFFFFFFFF
666     return crc
667 }
668 func byteCRC32(str []byte, len uint64)(crc uint32){
669     crc = byteCRC32add(0, str, len)
670     crc = byteCRC32end(crc, len)
671     return crc
672 }
673 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
674     var slen = make([]byte,4)
675     var li = 0
676     for li = 0; li < 4; {
677         slen[li] = byte(len & 0xFF)
678         li += 1
679         len >>= 8
680         if( len == 0 ){
681             break
682         }
683     }
684     crc = crc32.Update(crc, table, slen)
685     crc ^= 0xFFFFFFFF
686     return crc
687 }
688 }
689 func (gsh*GshContext)xChecksum(path string, argv []string, sum*Checksum)(int64){
690     if isin("-type/f", argv) && !IsRegFile(path){
691         return 0
692     }
693     if isin("-type/d", argv) && IsRegFile(path){
694         return 0
695     }
696     file, err := os.OpenFile(path, os.O_RDONLY, 0)
697     if err != nil {
698         fmt.Printf("--E-- cksum %v (%v)\n", path, err)
699         return -1
700     }
701     defer file.Close()
702     if gsh.CmdTrace { fmt.Printf("--I-- cksum %v %v\n", path, argv) }
703 }
704 bi := 0
705 var buff = make([]byte, 32*1024)
706 var total int64 = 0
707 var initTime = time.Time{}
708 if sum.Start == initTime {
709     sum.Start = time.Now()
710 }
711 for bi = 0; ; bi++ {
712     count, err := file.Read(buff)
713     if count <= 0 || err != nil {
714         break
715     }
716     if (sum.SumType & SUM_SUM64) != 0 {
717         s := sum.Sum64
718         for _, c := range buff[0:count] {
719             s += uint64(c)
720         }
721         sum.Sum64 = s
722     }
723     if (sum.SumType & SUM_UNIXFILE) != 0 {
724         sum.Crc32Val = byteCRC32add(sum.Crc32Val, buff, uint64(count))
725     }
726     if (sum.SumType & SUM_CRCIEEE) != 0 {
727         sum.Crc32Val = crc32.Update(sum.Crc32Val, &sum.Crc32Table, buff[0:count])
728     }
729     // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
730     if (sum.SumType & SUM_SUM16_BSD) != 0 {
731         s := sum.Sum16
732         for _, c := range buff[0:count] {
733             s = (s >> 1) + ((s & 1) << 15)
734             s += int(c)
735             s &= 0xFFFF
736             //fmt.Printf("BSDsum: %d[%d] %d\n", sum.Size+int64(i), i, s)
737         }
738         sum.Sum16 = s
739     }
740     if (sum.SumType & SUM_SUM16_SYSV) != 0 {
741         for bj := 0; bj < count; bj++ {
742             sum.Sum16 += int(buff[bj])
743         }
744     }
745     total += int64(count)
746 }
747 sum.Done = time.Now()
748 sum.Files += 1
749 sum.Size += total

```

```

750     if !isin("-s",argv) {
751         fmt.Printf("%v ",total)
752     }
753     return 0
754 }
755
756 // <a name="grep">grep</a>
757 // "lines", "lin" or "lmp" for "(text) line processor" or "scanner"
758 // a*,lab,c,... sequential combination of patterns
759 // what "LIMP" is should be definable
760 // generic line-by-line processing
761 // grep [-v]
762 // cat -n -v
763 // uniq [-c]
764 // tail -f
765 // sed s/x/y/ or awk
766 // grep with line count like wc
767 // rewrite contents if specified
768 func (gsh*GshContext)xGrep(path string, rexpv[]string)(int){
769     file, err := os.OpenFile(path,os.O_RDONLY,0)
770     if err != nil {
771         fmt.Printf("--E-- grep %v (%v)\n",path,err)
772         return -1
773     }
774     defer file.Close()
775     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path, rexpv) }
776     //reader := bufio.NewReaderSize(file,LINESIZE)
777     reader := bufio.NewReaderSize(file,80)
778     li := 0
779     found := 0
780     for li = 0; ; li++ {
781         line, err := reader.ReadString('\n')
782         if len(line) <= 0 {
783             break
784         }
785         if 150 < len(line) {
786             // maybe binary
787             break;
788         }
789         if err != nil {
790             break
791         }
792         if 0 <= strings.Index(string(line),rexpv[0]) {
793             found += 1
794             fmt.Printf("%s:%d: %s",path,li,line)
795         }
796     }
797     //fmt.Printf("total %d lines %s\n",li,path)
798     //if(0 < found){ fmt.Printf("(found %d lines %s)\n",found,path); }
799     return found
800 }
801
802 // <a name="finder">Finder</a>
803 // finding files with it name and contents
804 // file names are ORED
805 // show the content with %x fmt list
806 // ls -R
807 // tar command by adding output
808 type fileSum struct {
809     Err int64 // access error or so
810     Size int64 // content size
811     DupSize int64 // content size from hard links
812     Blocks int64 // number of blocks (of 512 bytes)
813     DupBlocks int64 // Blocks pointed from hard links
814     HLinks int64 // hard links
815     Words int64
816     Lines int64
817     Files int64
818     Dirs int64 // the num. of directories
819     SymLink int64
820     Flats int64 // the num. of flat files
821     MaxDepth int64
822     MaxNamlen int64 // max. name length
823     nextRepo time.Time
824 }
825 func showFusage(dir string,fusage *fileSum){
826     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
827     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
828
829     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
830         dir,
831         fusage.Files,
832         fusage.Dirs,
833         fusage.SymLink,
834         fusage.HLinks,
835         float64(fusage.Size)/1000000.0,bsume);
836 }
837 const (
838     S_IFMT = 0170000
839     S_IFCHR = 0020000
840     S_IFDIR = 0040000
841     S_IFREG = 0100000
842     S_IFLNK = 0120000
843     S_IFSOCK = 0140000
844 )
845 func cumPinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
846     now := time.Now()
847     if time.Second <= now.Sub(fsum.nextRepo) {
848         if !fsum.nextRepo.IsZero(){
849             tstamp := now.Format(time.Stamp)
850             showFusage(tstamp, fsum)
851         }
852         fsum.nextRepo = now.Add(time.Second)
853     }
854     if staterr != nil {
855         fsum.Err += 1
856         return fsum
857     }
858     fsum.Files += 1
859     if l <= fstat.Nlink {
860         // must count only once...
861         // at least ignore ones in the same directory
862         //if finfo.Mode().IsRegular() {
863         if (fstat.Mode & S_IFMT) == S_IFREG {
864             fsum.HLinks += 1
865             fsum.DupBlocks += int64(fstat.Blocks)
866             //fmt.Printf("---Dup HardLink %v %s\n",fstat.Nlink,path)
867         }
868     }
869     //fsum.Size += finfo.Size()
870     fsum.Size += fstat.Size
871     fsum.Blocks += int64(fstat.Blocks)
872     //if verb { fmt.Printf("(%dBlk) %s",fstat.Blocks/2,path) }
873     if isin("-ls",argv){
874         //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }

```

```

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

```



```

1000     return total
1001 }
1002 //filev,err := ioutil.ReadDir(dir)
1003 //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
1004 /*
1005 if err != nil {
1006     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
1007     return total
1008 }
1009 */
1010 if depth == 0 {
1011     total = cumPinfo(total,dir,staterr,dstat,argv,true)
1012     if !nols && !isin("-s",argv) && (!isin("-du",argv) || !isin("-a",argv)) {
1013         showFileInfo(dir,argv)
1014     }
1015 }
1016 // it it is not a directory, just scan it and finish
1017
1018 for ei := 0; ; ei++ {
1019     entv,rderr := dirfile.Readdirnames(8*1024)
1020     if len(entv) == 0 || rderr != nil {
1021         //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
1022         break
1023     }
1024     if 0 < ei {
1025         fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
1026     }
1027     total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npats,argv)
1028 }
1029 if !isin("-du",argv) {
1030     // if in "du" mode
1031     fmt.Printf("%d\t%s\n",total.Blocks-prev.Blocks)/2,dir)
1032 }
1033 return total
1034 }
1035
1036 // {ufind|fu|ls} [Files] [// Names] [-- Expressions]
1037 // Files is "." by default
1038 // Names is "*" by default
1039 // Expressions is "-print" by default for "ufind", or -du for "fu" command
1040 func (gsh*GshContext)xFind(argv[]string){
1041     if 0 < len(argv) && strBegins(argv[0],"?"){
1042         showFound(gsh,argv)
1043         return
1044     }
1045     if !isin("-cksum",argv) || !isin("-sum",argv) {
1046         gsh.lastCheckSum = CheckSum{}
1047         if !isin("-sum",argv) && !isin("-add",argv) {
1048             gsh.lastCheckSum.SumType |= SUM_SUM64
1049         }else
1050         if !isin("-sum",argv) && !isin("-size",argv) {
1051             gsh.lastCheckSum.SumType |= SUM_SIZE
1052         }else
1053         if !isin("-sum",argv) && !isin("-bsd",argv) {
1054             gsh.lastCheckSum.SumType |= SUM_SUM16_BSD
1055         }else
1056         if !isin("-sum",argv) && !isin("-sysv",argv) {
1057             gsh.lastCheckSum.SumType |= SUM_SUM16_SYSV
1058         }else
1059         if !isin("-sum",argv) {
1060             gsh.lastCheckSum.SumType |= SUM_SUM64
1061         }
1062         if !isin("-unix",argv) {
1063             gsh.lastCheckSum.SumType |= SUM_UNIXFILE
1064             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32UNIX)
1065         }
1066         if !isin("-ieee",argv){
1067             gsh.lastCheckSum.SumType |= SUM_CRCIEEE
1068             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32IEEE)
1069         }
1070         gsh.lastCheckSum.RusgAtStart = Getrusagev()
1071     }
1072     var total = fileSum{}
1073     npats := []string{}
1074     for _,v := range argv {
1075         if 0 < len(v) && v[0] != '-' {
1076             npats = append(npats,v)
1077         }
1078         if v == "/" { break }
1079         if v == "--" { break }
1080         if v == "-grep" { break }
1081         if v == "-ls" { break }
1082     }
1083     if len(npats) == 0 {
1084         npats = []string{"*"}
1085     }
1086     cwd := "."
1087     // if to be fullpath :: cwd, _ := os.Getwd()
1088     if len(npats) == 0 { npats = []string{"*"} }
1089     fusage := gsh.xxFind(0,&total,cwd,npats,argv)
1090     if gsh.lastCheckSum.SumType != 0 {
1091         var sumi uint64 = 0
1092         sum := &gsh.lastCheckSum
1093         if (sum.SumType & SUM_SIZE) != 0 {
1094             sumi = uint64(sum.Size)
1095         }
1096         if (sum.SumType & SUM_SUM64) != 0 {
1097             sumi = sum.Sum64
1098         }
1099         if (sum.SumType & SUM_SUM16_SYSV) != 0 {
1100             s := uint32(sum.Sum16)
1101             r := (s & 0xFFFF) + ((s & 0xFFFFFFFF) >> 16)
1102             s = (r & 0xFFFF) + (r >> 16)
1103             sum.Crc32Val = uint32(s)
1104             sumi = uint64(s)
1105         }
1106         if (sum.SumType & SUM_SUM16_BSD) != 0 {
1107             sum.Crc32Val = uint32(sum.Sum16)
1108             sumi = uint64(sum.Sum16)
1109         }
1110         if (sum.SumType & SUM_UNIXFILE) != 0 {
1111             sum.Crc32Val = byteCRC32end(sum.Crc32Val,uint64(sum.Size))
1112             sumi = uint64(byteCRC32end(sum.Crc32Val,uint64(sum.Size)))
1113         }
1114         if 1 < sum.Files {
1115             fmt.Printf("%v %v // %v / %v files, %v/file\r\n",
1116                 sumi,sum.Size,
1117                 abssize(sum.Size),sum.Files,
1118                 abssize(sum.Size/sum.Files))
1119         }else{
1120             fmt.Printf("%v %v %v\n",
1121                 sumi,sum.Size,npats[0])
1122         }
1123     }
1124     if !isin("-grep",argv) {

```

```

1125     showFusage("total", fusage)
1126 }
1127 if !isin("-s", argv){
1128     hits := len(gsh.CmdCurrent.FoundFile)
1129     if 0 < hits {
1130         fmt.Printf("--I-- %d files hits // can be refered with !&df\n",
1131             hits, len(gsh.CommandHistory))
1132     }
1133 }
1134 if gsh.lastCheckSum.SumType != 0 {
1135     if isin("-ru", argv){
1136         sum := gsh.lastCheckSum
1137         sum.Done = time.Now()
1138         gsh.lastCheckSum.RusgAtEnd = Getrusagev()
1139         elps := sum.Done.Sub(sum.Start)
1140         fmt.Printf("--cksum-size: %v (%v) / %v files, %v/file\r\n",
1141             sum.Size, absSize(sum.Size), sum.Files, absSize(sum.Size)/sum.Files)
1142         nanos := int64(elps)
1143         fmt.Printf("--cksum-time: %v/total, %v/file, %1f files/s, %v\r\n",
1144             abftime(nanos),
1145             abftime(nanos/sum.Files),
1146             (float64(sum.Files)*1000000000.0)/float64(nanos),
1147             absSpeed(sum.Size, nanos))
1148         diff := RusageSubv(sum.RusgAtEnd, sum.RusgAtStart)
1149         fmt.Printf("--cksum-rusg: %v\n", sRusagef("", argv, diff))
1150     }
1151 }
1152 return
1153 }
1154
1155 func showFiles(files []string){
1156     sp := ""
1157     for i, file := range files {
1158         if 0 < i { sp = " " } else { sp = "" }
1159         fmt.Printf(sp+"%s", escapeWhiteSP(file))
1160     }
1161 }
1162 func showFound(gshCtx *GshContext, argv []string){
1163     for i, v := range gshCtx.CommandHistory {
1164         if 0 < len(v.FoundFile) {
1165             fmt.Printf("%d (%d) ", i, len(v.FoundFile))
1166             if isin("-ls", argv){
1167                 fmt.Printf("\n")
1168                 for file := range v.FoundFile {
1169                     fmt.Printf(" ") //sub number?
1170                     showFileInfo(file, argv)
1171                 }
1172             }else{
1173                 showFiles(v.FoundFile)
1174                 fmt.Printf("\n")
1175             }
1176         }
1177     }
1178 }
1179
1180 func showMatchFile(filev []os.FileInfo, npat, dir string, argv []string)(string, bool){
1181     fname := ""
1182     found := false
1183     for _, v := range filev {
1184         match, _ := filepath.Match(npat, (v.Name()))
1185         if match {
1186             fname = v.Name()
1187             found = true
1188             //fmt.Printf("[%d] %s\n", i, v.Name())
1189             showIfExecutable(fname, dir, argv)
1190         }
1191     }
1192     return fname, found
1193 }
1194 func showIfExecutable(name, dir string, argv []string)(ffullpath string, ffound bool){
1195     var fullpath string
1196     if strBegins(name, DIRSEP){
1197         fullpath = name
1198     }else{
1199         fullpath = dir + DIRSEP + name
1200     }
1201     fi, err := os.Stat(fullpath)
1202     if err != nil {
1203         fullpath = dir + DIRSEP + name + ".go"
1204         fi, err = os.Stat(fullpath)
1205     }
1206     if err == nil {
1207         fm := fi.Mode()
1208         if fm.IsRegular() {
1209             // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1210             if syscall.Access(fullpath, 5) == nil {
1211                 ffullpath = fullpath
1212                 ffound = true
1213                 if !isin("-s", argv) {
1214                     showFileInfo(fullpath, argv)
1215                 }
1216             }
1217         }
1218     }
1219     return ffullpath, ffound
1220 }
1221 func which(list string, argv []string) (fullpathv []string, itis bool){
1222     if len(argv) <= 1 {
1223         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1224         return []string(""), false
1225     }
1226     path := argv[1]
1227     if strBegins(path, "/") {
1228         // should check if excecutable?
1229         _, exOK := showIfExecutable(path, "/", argv)
1230         fmt.Printf("--D-- %v exOK=%v\n", path, exOK)
1231         return []string{path}, exOK
1232     }
1233     pathenv, efound := os.LookupEnv(list)
1234     if !efound {
1235         fmt.Printf("--E-- which: no \"%s\" environment\n", list)
1236         return []string(""), false
1237     }
1238     showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
1239     dirv := strings.Split(pathenv, PATHSEP)
1240     ffound := false
1241     ffullpath := path
1242     for _, dir := range dirv {
1243         if 0 <= strings.Index(path, "*") { // by wild-card
1244             list, _ := ioutil.ReadDir(dir)
1245             ffullpath, ffound = showMatchFile(list, path, dir, argv)
1246         }else{
1247             ffullpath, ffound = showIfExecutable(path, dir, argv)
1248         }
1249         //if ffound && !isin("-a", argv) {

```

```

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

```

```

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

```

```

1500         }else{
1501             rusage := syscall.Rusage {}
1502             syscall.Wait4(pid,nil,0,&rusage);
1503             gsh.LastRusage = rusage
1504             gsh.CmdCurrent.Rusagev[1] = rusage
1505         }
1506     }
1507 }
1508 return false,false
1509 }
1510
1511 // <a name="builtin">Builtin Commands</a>
1512 func (gshCtx *GshContext) sleep(argv []string) {
1513     if len(argv) < 2 {
1514         fmt.Printf("Sleep 100ms, 100us, 100ns, ...\\n")
1515         return
1516     }
1517     duration := argv[1];
1518     d, err := time.ParseDuration(duration)
1519     if err != nil {
1520         d, err = time.ParseDuration(duration+"s")
1521         if err != nil {
1522             fmt.Printf("duration ? %s (%s)\\n",duration,err)
1523             return
1524         }
1525     }
1526     //fmt.Printf("Sleep %v\\n",duration)
1527     time.Sleep(d)
1528     if 0 < len(argv[2:]) {
1529         gshCtx.gshellv(argv[2:])
1530     }
1531 }
1532 func (gshCtx *GshContext)repeat(argv []string) {
1533     if len(argv) < 2 {
1534         return
1535     }
1536     start0 := time.Now()
1537     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1538         if 0 < len(argv[2:]) {
1539             //start := time.Now()
1540             gshCtx.gshellv(argv[2:])
1541             end := time.Now()
1542             elps := end.Sub(start0);
1543             if( 1000000000 < elps ){
1544                 fmt.Printf("repeat#%d %v)\\n",ri,elps);
1545             }
1546         }
1547     }
1548 }
1549
1550 func (gshCtx *GshContext)gen(argv []string) {
1551     gshPA := gshCtx.gshPA
1552     if len(argv) < 2 {
1553         fmt.Printf("Usage: %s N\\n",argv[0])
1554         return
1555     }
1556     // should br repeated by "repeat" command
1557     count, _ := strconv.Atoi(argv[1])
1558     fd := gshPA.Files[1] // Stdout
1559     file := os.NewFile(fd,"internalStdOut")
1560     fmt.Printf("--I-- Gen. Count=%d to [%d)\\n",count,file.Fd())
1561     //buf := []byte{}
1562     outdata := "0123 5678 0123 5678 0123 5678 0123 5678 \\r"
1563     for gi := 0; gi < count; gi++ {
1564         file.WriteString(outdata)
1565     }
1566     //file.WriteString("\\n")
1567     fmt.Printf("\\n(%d B)\\n",count*len(outdata));
1568     //file.Close()
1569 }
1570
1571 // <a name="rexec">Remote Execution</a> // 2020-0820
1572 func Elapsed(from time.Time)(string){
1573     elps := time.Now().Sub(from)
1574     if 1000000000 < elps {
1575         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1576     }
1577     if 1000000 < elps {
1578         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1579     }
1580     return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1581 }
1582 }
1583 func abftime(nanos int64)(string){
1584     if 1000000000 < nanos {
1585         return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/1000000)
1586     }
1587     if 1000000 < nanos {
1588         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1589     }
1590     return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1591 }
1592 }
1593 func abssize(size int64)(string){
1594     fsize := float64(size)
1595     if 1024*1024*1024 < size {
1596         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1597     }
1598     if 1024*1024 < size {
1599         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1600     }
1601     return fmt.Sprintf("%.3fKiB",fsize/1024)
1602 }
1603 }
1604 func abszize(size int64)(string){
1605     fsize := float64(size)
1606     if 1024*1024*1024 < size {
1607         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1608     }
1609     if 1024*1024 < size {
1610         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1611     }
1612     return fmt.Sprintf("%.3fKiB",fsize/1024)
1613 }
1614 }
1615 func abbspeed(totalB int64,ns int64)(string){
1616     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1617     if 1000 <= MBs {
1618         return fmt.Sprintf("%.3fGB/s",MBs/1000)
1619     }
1620     if 1 <= MBs {
1621         return fmt.Sprintf("%.3fMB/s",MBs)
1622     }
1623     return fmt.Sprintf("%.3fKB/s",MBs*1000)
1624 }

```

```

1625 }
1626 func abspeed(totalB int64,ns time.Duration)(string){
1627     Mbs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1628     if 1000 <= Mbs {
1629         return fmt.Sprintf("%6.3fGBps",Mbs/1000)
1630     }
1631     if 1 <= Mbs {
1632         return fmt.Sprintf("%6.3fMBps",Mbs)
1633     }else{
1634         return fmt.Sprintf("%6.3fKBps",Mbs*1000)
1635     }
1636 }
1637 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1638     Start := time.Now()
1639     Buff := make([]byte,bsiz)
1640     var total int64 = 0
1641     var rem int64 = size
1642     nio := 0
1643     Prev := time.Now()
1644     var PrevSize int64 = 0
1645
1646     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1647         what,absize(total),size,nio)
1648
1649     for i:= 0; ; i++ {
1650         var len = bsiz
1651         if int(rem) < len {
1652             len = int(rem)
1653         }
1654         Now := time.Now()
1655         Elps := Now.Sub(Prev);
1656         if 1000000000 < Now.Sub(Prev) {
1657             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1658                 what,absize(total),size,nio,
1659                 abspeed((total-PrevSize),Elps))
1660             Prev = Now;
1661             PrevSize = total
1662         }
1663         rlen := len
1664         if in != nil {
1665             // should watch the disconnection of out
1666             rcc,err := in.Read(buff[0:rlen])
1667             if err != nil {
1668                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1669                     what,rcc,err,in.Name())
1670                 break
1671             }
1672             rlen = rcc
1673             if string(buff[0:10]) == "(SoftEOF " {
1674                 var ecc int64 = 0
1675                 fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1676                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/&v\n",
1677                     what,ecc,total)
1678                 if ecc == total {
1679                     break
1680                 }
1681             }
1682         }
1683
1684         wlen := rlen
1685         if out != nil {
1686             wcc,err := out.Write(buff[0:rlen])
1687             if err != nil {
1688                 fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1689                     what,wcc,err,out.Name())
1690                 break
1691             }
1692             wlen = wcc
1693         }
1694         if wlen < rlen {
1695             fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1696                 what,wlen,rlen)
1697             break;
1698         }
1699
1700         nio += 1
1701         total += int64(rlen)
1702         rem -= int64(rlen)
1703         if rem <= 0 {
1704             break
1705         }
1706     }
1707     Done := time.Now()
1708     Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1709     TotalMB := float64(total)/1000000 //MB
1710     MBps := TotalMB / Elps
1711     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %6.3fMB/s\n",
1712         what,total,size,nio,absize(total),MBps)
1713     return total
1714 }
1715 func tcpPush(clnt *os.File){
1716     // shrink socket buffer and recover
1717     usleep(100);
1718 }
1719 func (gsh*GshContext)RexecServer(argv []string){
1720     debug := true
1721     Start0 := time.Now()
1722     Start := Start0
1723     // if local == ":" { local = "0.0.0.0:9999" }
1724     local := "0.0.0.0:9999"
1725
1726     if 0 < len(argv) {
1727         if argv[0] == "-s" {
1728             debug = false
1729             argv = argv[1:]
1730         }
1731     }
1732     if 0 < len(argv) {
1733         argv = argv[1:]
1734     }
1735     port, err := net.ResolveTCPAddr("tcp",local);
1736     if err != nil {
1737         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1738         return
1739     }
1740     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1741     sconn, err := net.ListenTCP("tcp", port)
1742     if err != nil {
1743         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1744         return
1745     }
1746
1747     reqbuf := make([]byte,LINESIZE)
1748     res := ""
1749     for {

```

```

1750     fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1751     aconn, err := sconn.AcceptTCP()
1752     Start = time.Now()
1753     if err != nil {
1754         fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1755         return
1756     }
1757     clnt, _ := aconn.File()
1758     fd := clnt.Fd()
1759     ar := aconn.RemoteAddr()
1760     if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1761         local,fd,ar) }
1762     res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1763     fmt.Fprintf(clnt,"%s",res)
1764     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1765     count, err := clnt.Read(reqbuf)
1766     if err != nil {
1767         fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1768             count,err,string(reqbuf))
1769     }
1770     req := string(reqbuf[:count])
1771     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1772     reqv := strings.Split(string(req),"r")
1773     cmdv := gshScanArg(reqv[0],0)
1774     //cmdv := strings.Split(reqv[0]," ")
1775     switch cmdv[0] {
1776     case "HELO":
1777         res = fmt.Sprintf("250 %v",req)
1778     case "GET":
1779         // download {remotefile|-zN} [localfile]
1780         var dsize int64 = 32*1024*1024
1781         var bsize int = 64*1024
1782         var fname string = ""
1783         var in *os.File = nil
1784         var pseudoEOF = false
1785         if 1 < len(cmdv) {
1786             fname = cmdv[1]
1787             if strBegins(fname,"-z") {
1788                 fmt.Sscanf(fname[2:], "%d",&dsize)
1789             }else{
1790                 if strBegins(fname,"{") {
1791                     xin,xout,err := gsh.Popen(fname,"r")
1792                     if err {
1793                         }else{
1794                             xout.Close()
1795                             defer xin.Close()
1796                             in = xin
1797                             dsize = MaxStreamSize
1798                             pseudoEOF = true
1799                         }
1800                 }else{
1801                     xin,err := os.Open(fname)
1802                     if err != nil {
1803                         fmt.Printf("--En- GET (%v)\n",err)
1804                     }else{
1805                         defer xin.Close()
1806                         in = xin
1807                         fi,_ := xin.Stat()
1808                         dsize = fi.Size()
1809                     }
1810                 }
1811             }
1812             //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsize,bsize)
1813             res = fmt.Sprintf("200 %v\r\n",dsize)
1814             fmt.Fprintf(clnt,"%v",res)
1815             tcpPush(clnt); // should be separated as line in receiver
1816             fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1817             wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
1818             if pseudoEOF {
1819                 in.Close() // pipe from the command
1820                 // show end of stream data (its size) by OOB?
1821                 SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1822                 fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1823             }
1824             tcpPush(clnt); // to let SoftEOF data apper at the top of received data
1825             fmt.Fprintf(clnt,"%v\r\n",SoftEOF)
1826             tcpPush(clnt); // to let SoftEOF alone in a packet (separate with 200 OK)
1827             // with client generated random?
1828             //fmt.Printf("--In- L: close %v (%v)\n",in.Fd(),in.Name())
1829         }
1830         res = fmt.Sprintf("200 GET done\r\n")
1831     case "PUT":
1832         // upload {srcfile|-zN} [dstfile]
1833         var dsize int64 = 32*1024*1024
1834         var bsize int = 64*1024
1835         var fname string = ""
1836         var out *os.File = nil
1837         if 1 < len(cmdv) { // localfile
1838             fmt.Sscanf(cmdv[1],"%d",&dsize)
1839         }
1840         if 2 < len(cmdv) {
1841             fname = cmdv[2]
1842             if fname == "-" {
1843                 // nul dev
1844             }else{
1845                 if strBegins(fname,"{") {
1846                     xin,xout,err := gsh.Popen(fname,"w")
1847                     if err {
1848                         }else{
1849                             xin.Close()
1850                             defer xout.Close()
1851                             out = xout
1852                         }
1853                     }else{
1854                         // should write to temporary file
1855                         // should suppress ^C on tty
1856                         xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1857                         //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1858                         if err != nil {
1859                             fmt.Printf("--En- PUT (%v)\n",err)
1860                         }else{
1861                             out = xout
1862                         }
1863                     }
1864                 }
1865                 fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1866                     fname,local,err)
1867             }
1868             fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1869             fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1870             fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1871             fileRelay("RecvPUT",clnt,out,dsize,bsize)
1872             res = fmt.Sprintf("200 PUT done\r\n")
1873         default:
1874             res = fmt.Sprintf("400 What? %v",req)
1875     }

```

```

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

```



```

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

```

```

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

```

```

2250     if 1 < len(argv) {
2251         local = argv[1]
2252         remote = local // base name
2253     }
2254     if 2 < len(argv) {
2255         remote = argv[2]
2256     }
2257     fmt.Printf("--I-- jput from=%v to=%v\n",local,gsh.Trelpath(remote))
2258 }
2259 func (gsh*GshContext)Rget(argv[]string){
2260     var remote string = ""
2261     var local string = ""
2262     if 1 < len(argv) {
2263         remote = argv[1]
2264         local = remote // base name
2265     }
2266     if 2 < len(argv) {
2267         local = argv[2]
2268     }
2269     fmt.Printf("--I-- jget from=%v to=%v\n",gsh.Trelpath(remote),local)
2270 }
2271
2272 // <a name="network">network</a>
2273 // -s, -si, -so // bi-directional, source, sync (maybe socket)
2274 func (gshCtx*GshContext)sconnect(inTCP bool, argv []string) {
2275     gshPA := gshCtx.gshPA
2276     if len(argv) < 2 {
2277         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
2278         return
2279     }
2280     remote := argv[1]
2281     if remote == ":" { remote = "0.0.0.0:9999" }
2282
2283     if inTCP { // TCP
2284         dport, err := net.ResolveTCPAddr("tcp",remote);
2285         if err != nil {
2286             fmt.Printf("Address error: %s (%s)\n",remote,err)
2287             return
2288         }
2289         conn, err := net.DialTCP("tcp",nil,dport)
2290         if err != nil {
2291             fmt.Printf("Connection error: %s (%s)\n",remote,err)
2292             return
2293         }
2294         file, _ := conn.File();
2295         fd := file.Fd()
2296         fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
2297
2298         savfd := gshPA.Files[1]
2299         gshPA.Files[1] = fd;
2300         gshCtx.gshelly(argv[2:])
2301         gshPA.Files[1] = savfd
2302         file.Close()
2303         conn.Close()
2304     }else{
2305         //dport, err := net.ResolveUDPAddr("udp4",remote);
2306         dport, err := net.ResolveUDPAddr("udp",remote);
2307         if err != nil {
2308             fmt.Printf("Address error: %s (%s)\n",remote,err)
2309             return
2310         }
2311         //conn, err := net.DialUDP("udp4",nil,dport)
2312         conn, err := net.DialUDP("udp",nil,dport)
2313         if err != nil {
2314             fmt.Printf("Connection error: %s (%s)\n",remote,err)
2315             return
2316         }
2317         file, _ := conn.File();
2318         fd := file.Fd()
2319
2320         ar := conn.RemoteAddr()
2321         //al := conn.LocalAddr()
2322         fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
2323             remote,ar.String(),fd)
2324
2325         savfd := gshPA.Files[1]
2326         gshPA.Files[1] = fd;
2327         gshCtx.gshelly(argv[2:])
2328         gshPA.Files[1] = savfd
2329         file.Close()
2330         conn.Close()
2331     }
2332 }
2333 func (gshCtx*GshContext)saccept(inTCP bool, argv []string) {
2334     gshPA := gshCtx.gshPA
2335     if len(argv) < 2 {
2336         fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
2337         return
2338     }
2339     local := argv[1]
2340     if local == ":" { local = "0.0.0.0:9999" }
2341     if inTCP { // TCP
2342         port, err := net.ResolveTCPAddr("tcp",local);
2343         if err != nil {
2344             fmt.Printf("Address error: %s (%s)\n",local,err)
2345             return
2346         }
2347         //fmt.Printf("Listen at %s...\n",local);
2348         sconn, err := net.ListenTCP("tcp", port)
2349         if err != nil {
2350             fmt.Printf("Listen error: %s (%s)\n",local,err)
2351             return
2352         }
2353         //fmt.Printf("Accepting at %s...\n",local);
2354         aconn, err := sconn.AcceptTCP()
2355         if err != nil {
2356             fmt.Printf("Accept error: %s (%s)\n",local,err)
2357             return
2358         }
2359         file, _ := aconn.File()
2360         fd := file.Fd()
2361         fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
2362
2363         savfd := gshPA.Files[0]
2364         gshPA.Files[0] = fd;
2365         gshCtx.gshelly(argv[2:])
2366         gshPA.Files[0] = savfd
2367
2368         sconn.Close();
2369         aconn.Close();
2370         file.Close();
2371     }else{
2372         //port, err := net.ResolveUDPAddr("udp4",local);
2373         port, err := net.ResolveUDPAddr("udp",local);
2374         if err != nil {

```

```

2375         fmt.Printf("Address error: %s (%s)\n",local,err)
2376         return
2377     }
2378     fmt.Printf("Listen UDP at %s...\n",local);
2379     //uconn, err := net.ListenUDP("udp4", port)
2380     uconn, err := net.ListenUDP("udp", port)
2381     if err != nil {
2382         fmt.Printf("Listen error: %s (%s)\n",local,err)
2383         return
2384     }
2385     file, _ := uconn.File()
2386     fd := file.Fd()
2387     ar := uconn.RemoteAddr()
2388     remote := ""
2389     if ar != nil { remote = ar.String() }
2390     if remote == "" { remote = "?" }
2391
2392     // not yet received
2393     //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")
2394
2395     savfd := gshPA.Files[0]
2396     gshPA.Files[0] = fd;
2397     savenv := gshPA.Env
2398     gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
2399     gshCtx.gshenv(argv[2:])
2400     gshPA.Env = savenv
2401     gshPA.Files[0] = savfd
2402
2403     uconn.Close();
2404     file.Close();
2405 }
2406 }
2407
2408 // empty line command
2409 func (gshCtx*GshContext)xPwd(argv[]string){
2410     // execute context command, pwd + date
2411     // context notation, representation scheme, to be resumed at re-login
2412     cwd, _ := os.Getwd()
2413     switch {
2414     case isin("-a",argv):
2415         gshCtx.ShowChdirHistory(argv)
2416     case isin("-ls",argv):
2417         showFileInfo(cwd,argv)
2418     default:
2419         fmt.Printf("%s\n",cwd)
2420     case isin("-v",argv): // obsolete empty command
2421         t := time.Now()
2422         date := t.Format(time.UnixDate)
2423         exe, _ := os.Executable()
2424         host, _ := os.Hostname()
2425         fmt.Printf("PWD=\"%s\" ",cwd)
2426         fmt.Printf("HOST=\"%s\" ",host)
2427         fmt.Printf("DATE=\"%s\" ",date)
2428         fmt.Printf("TIME=\"%s\" ",t.String())
2429         fmt.Printf("PID=\"%d\" ",os.Getpid())
2430         fmt.Printf("EXE=\"%s\" ",exe)
2431         fmt.Printf("\n")
2432     }
2433 }
2434
2435 // <a name="history">History</a>
2436 // these should be browsed and edited by HTTP browser
2437 // show the time of command with -t and direcotry with -ls
2438 // openfile-history, sort by -a -m -c
2439 // sort by elapsed time by -t -s
2440 // search by "more" like interface
2441 // edit history
2442 // sort history, and we or unig
2443 // CPU and other resource consumptions
2444 // limit showing range (by time or so)
2445 // export / import history
2446 func (gshCtx *GshContext)xHistory(argv []string){
2447     atWorkDirX := -1
2448     if 1 < len(argv) && strBegins(argv[1],"@") {
2449         atWorkDirX,_ = strconv.Atoi(argv[1][1:])
2450     }
2451     //fmt.Printf("--D-- showHistory(%v)\n",argv)
2452     for i, v := range gshCtx.CommandHistory {
2453         // exclude commands not to be listed by default
2454         // internal commands may be suppressed by default
2455         if v.CmdLine == "" && !isin("-a",argv) {
2456             continue;
2457         }
2458         if 0 <= atWorkDirX {
2459             if v.WorkDirX != atWorkDirX {
2460                 continue
2461             }
2462         }
2463         if !isin("-n",argv){ // like "fc"
2464             fmt.Printf("!%-2d ",i)
2465         }
2466         if isin("-v",argv){
2467             fmt.Println(v) // should be with it date
2468         }else{
2469             if isin("-l",argv) || isin("-l0",argv) {
2470                 elps := v.EndAt.Sub(v.StartAt);
2471                 start := v.StartAt.Format(time.Stamp)
2472                 fmt.Printf("@%d ",v.WorkDirX)
2473                 fmt.Printf("[%v] %11v/t ",start,elps)
2474             }
2475             if isin("-l",argv) && !isin("-l0",argv){
2476                 fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
2477             }
2478             if isin("-at",argv) { // !isin("-ls",argv){
2479                 dhi := v.WorkDirX // workdir history index
2480                 fmt.Printf("@%d %s\t",dhi,v.WorkDir)
2481                 // show the FileInfo of the output command??
2482             }
2483             fmt.Printf("%s",v.CmdLine)
2484             fmt.Printf("\n")
2485         }
2486     }
2487 }
2488 // !n - history index
2489 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2490     if gline[0] == '!' {
2491         hix, err := strconv.Atoi(gline[1:])
2492         if err != nil {
2493             fmt.Printf("--E-- (%s : range)\n",hix)
2494             return "", false, true
2495         }
2496         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2497             fmt.Printf("--E-- (%d : out of range)\n",hix)
2498             return "", false, true
2499         }

```

```

2500     return gshCtx.CommandHistory[hix].CmdLine, false, false
2501 }
2502 // search
2503 //for i, v := range gshCtx.CommandHistory {
2504 //}
2505 return gline, false, false
2506 }
2507 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2508     if 0 <= hix && hix < len(gsh.CommandHistory) {
2509         return gsh.CommandHistory[hix].CmdLine,true
2510     }
2511     return "",false
2512 }
2513 }
2514 // temporary adding to PATH environment
2515 // cd name -lib for LD_LIBRARY_PATH
2516 // chdir with directory history (date + full-path)
2517 // -s for sort option (by visit date or so)
2518 func (gsh*GshContext)ShowChdirHistory1(i int,v GChdirHistory, argv []string){
2519     fmt.Printf("%s-2d ",v.CmdIndex) // the first command at this WorkDir
2520     fmt.Printf("%d ", i)
2521     fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
2522     showFileInfo(v.Dir,argv)
2523 }
2524 func (gsh*GshContext)ShowChdirHistory(argv []string){
2525     for i, v := range gsh.ChdirHistory {
2526         gsh.ShowChdirHistory1(i,v,argv)
2527     }
2528 }
2529 func skipOpts(argv[]string)(int){
2530     for i,v := range argv {
2531         if strBegins(v,"-") {
2532             }else{
2533                 return i
2534             }
2535     }
2536     return -1
2537 }
2538 func (gshCtx*GshContext)xChdir(argv []string){
2539     cdhist := gshCtx.ChdirHistory
2540     if isin("?",argv) || isin("-",argv) || isin("-a",argv) {
2541         gshCtx.ShowChdirHistory(argv)
2542         return
2543     }
2544     pwd, _ := os.Getwd()
2545     dir := ""
2546     if len(argv) <= 1 {
2547         dir = toFullpath("-")
2548     }else{
2549         i := skipOpts(argv[1:])
2550         if i < 0 {
2551             dir = toFullpath("-")
2552         }else{
2553             dir = argv[1+i]
2554         }
2555     }
2556     if strBegins(dir,"@") {
2557         if dir == "@0" { // obsolete
2558             dir = gshCtx.StartDir
2559         }else
2560         if dir == "@1" {
2561             index := len(cdhist) - 1
2562             if 0 < index { index -= 1 }
2563             dir = cdhist[index].Dir
2564         }else{
2565             index, err := strconv.Atoi(dir[1:])
2566             if err != nil {
2567                 fmt.Printf("--E-- xChdir(%v)\n",err)
2568                 dir = "?"
2569             }else
2570             if len(gshCtx.ChdirHistory) <= index {
2571                 fmt.Printf("--E-- xChdir(history range error)\n")
2572                 dir = "?"
2573             }else{
2574                 dir = cdhist[index].Dir
2575             }
2576         }
2577     }
2578     if dir != "?" {
2579         err := os.Chdir(dir)
2580         if err != nil {
2581             fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)
2582         }else{
2583             cwd, _ := os.Getwd()
2584             if cwd != pwd {
2585                 hist1 := GChdirHistory { }
2586                 hist1.Dir = cwd
2587                 hist1.MovedAt = time.Now()
2588                 hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2589                 gshCtx.ChdirHistory = append(cdhist,hist1)
2590                 if lisin("-s",argv){
2591                     //cwd, _ := os.Getwd()
2592                     //fmt.Printf("%s\n",cwd)
2593                     ix := len(gshCtx.ChdirHistory)-1
2594                     gshCtx.ShowChdirHistory1(ix,hist1,argv)
2595                 }
2596             }
2597         }
2598     }
2599     if isin("-ls",argv){
2600         cwd, _ := os.Getwd()
2601         showFileInfo(cwd,argv);
2602     }
2603 }
2604 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2605     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2606 }
2607 func RusageSubv(rul, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2608     TimeValSub(&rul[0].Utime,&ru2[0].Utime)
2609     TimeValSub(&rul[0].Stime,&ru2[0].Stime)
2610     TimeValSub(&rul[1].Utime,&ru2[1].Utime)
2611     TimeValSub(&rul[1].Stime,&ru2[1].Stime)
2612     return rul
2613 }
2614 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2615     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2616     return tvs
2617 }
2618 /*
2619 func RusageAddv(rul, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2620     TimeValAdd(rul[0].Utime,ru2[0].Utime)
2621     TimeValAdd(rul[0].Stime,ru2[0].Stime)
2622     TimeValAdd(rul[1].Utime,ru2[1].Utime)
2623     TimeValAdd(rul[1].Stime,ru2[1].Stime)
2624     return rul

```

```

2625 }
2626 */
2627
2628 // <a name="rusage">Resource Usage</a>
2629 func sRusage(fmts spec string, argv []string, ru [2]syscall.Rusage)(string){
2630     // ru[0] self, ru[1] children
2631     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2632     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2633     uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2634     su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2635     tu := uu + su
2636     ret := fmt.Sprintf("%v/sum",abftime(tu))
2637     ret += fmt.Sprintf(" ", %v/usr",abftime(uu))
2638     ret += fmt.Sprintf(" ", %v/sys",abftime(su))
2639     return ret
2640 }
2641 func Rusage(fmts spec string, argv []string, ru [2]syscall.Rusage)(string){
2642     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2643     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2644     fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2645     fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2646     return ""
2647 }
2648 func Getrusagev()([2]syscall.Rusage){
2649     var ruv = [2]syscall.Rusage{}
2650     syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2651     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2652     return ruv
2653 }
2654 func showRusage(what string,argv []string, ru *syscall.Rusage){
2655     fmt.Printf("%s: ",what);
2656     fmt.Printf("User=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2657     fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2658     fmt.Printf(" Rss=%vB",ru.Maxrss)
2659     if isin("-l",argv) {
2660         fmt.Printf(" MinFlt=%v",ru.Minflt)
2661         fmt.Printf(" MajFlt=%v",ru.Majflt)
2662         fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2663         fmt.Printf(" IdRSS=%vB",ru.Idrss)
2664         fmt.Printf(" Nswap=%vB",ru.Nswap)
2665         fmt.Printf(" Read=%v",ru.Inblock)
2666         fmt.Printf(" Write=%v",ru.Oblock)
2667     }
2668     fmt.Printf(" Snd=%v",ru.Msgsnd)
2669     fmt.Printf(" Rcv=%v",ru.Msgrcv)
2670     //if isin("-l",argv) {
2671         fmt.Printf(" Sig=%v",ru.Nsignals)
2672     //}
2673     fmt.Printf("\n");
2674 }
2675 func (gshCtx *GshContext)xTime(argv []string)(bool){
2676     if 2 <= len(argv){
2677         gshCtx.LastRusage = syscall.Rusage{}
2678         rusagev1 := Getrusagev()
2679         fin := gshCtx.gshellv(argv[1:])
2680         rusagev2 := Getrusagev()
2681         showRusage(argv[1],argv,&gshCtx.LastRusage)
2682         rusagev := RusageSubv(rusagev2,rusagev1)
2683         showRusage("self",argv,&rusagev[0])
2684         showRusage("chld",argv,&rusagev[1])
2685         return fin
2686     }else{
2687         rusage:= syscall.Rusage {}
2688         syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2689         showRusage("self",argv, &rusage)
2690         syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2691         showRusage("chld",argv, &rusage)
2692         return false
2693     }
2694 }
2695 func (gshCtx *GshContext)xJobs(argv []string){
2696     fmt.Printf("%d Jobs\n",len(gshCtx.BackgroundJobs))
2697     for ji, pid := range gshCtx.BackgroundJobs {
2698         //wstat := syscall.WaitStatus {0}
2699         rusage := syscall.Rusage {}
2700         //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2701         wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2702         if err != nil {
2703             fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
2704         }else{
2705             fmt.Printf("%%d[%d](%d)\n",ji,pid,wpid)
2706             showRusage("chld",argv,&rusage)
2707         }
2708     }
2709 }
2710 func (gsh*GshContext)inBackground(argv []string)(bool){
2711     if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2712     gsh.BackGround = true // set background option
2713     xfin := false
2714     xfin = gsh.gshellv(argv)
2715     gsh.BackGround = false
2716     return xfin
2717 }
2718 // -o file without command means just opening it and refer by #N
2719 // should be listed by "files" command
2720 func (gshCtx*GshContext)xOpen(argv []string){
2721     var pv = [int{-1,-1}]
2722     err := syscall.Pipe(pv)
2723     fmt.Printf("--I-- pipe()=[%d,%d](%v)\n",pv[0],pv[1],err)
2724 }
2725 func (gshCtx*GshContext)fromPipe(argv []string){
2726 }
2727 func (gshCtx*GshContext)xClose(argv []string){
2728 }
2729
2730 // <a name="redirect">redirect</a>
2731 func (gshCtx*GshContext)redirect(argv []string)(bool){
2732     if len(argv) < 2 {
2733         return false
2734     }
2735
2736     cmd := argv[0]
2737     fname := argv[1]
2738     var file *os.File = nil
2739
2740     fdix := 0
2741     mode := os.O_RDONLY
2742
2743     switch {
2744     case cmd == "-i" || cmd == "<":
2745         fdix = 0
2746         mode = os.O_RDONLY
2747     case cmd == "-o" || cmd == ">":
2748         fdix = 1
2749         mode = os.O_RDWR | os.O_CREATE

```

```

2750 case cmd == "-a" || cmd == ">>":
2751     fdix = 1
2752     mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2753 }
2754 if fname[0] == '#' {
2755     fd, err := strconv.Atoi(fname[1:])
2756     if err != nil {
2757         fmt.Printf("--E-- (%v)\n",err)
2758         return false
2759     }
2760     file = os.NewFile(uintptr(fd),"MaybePipe")
2761 }else{
2762     xfile, err := os.OpenFile(argv[1], mode, 0600)
2763     if err != nil {
2764         fmt.Printf("--E-- (%s)\n",err)
2765         return false
2766     }
2767     file = xfile
2768 }
2769 gshPA := gshCtx.gshPA
2770 savfd := gshPA.Files[fdix]
2771 gshPA.Files[fdix] = file.Fd()
2772 fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2773 gshCtx.gshellv(argv[2:])
2774 gshPA.Files[fdix] = savfd
2775
2776 return false
2777 }
2778
2779 //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2780 func httpHandler(res http.ResponseWriter, req *http.Request){
2781     path := req.URL.Path
2782     fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2783     {
2784         gshCtxBuf, _ := setupGshContext()
2785         gshCtx := *gshCtxBuf
2786         fmt.Printf("--I-- %s\n",path[1:])
2787         gshCtx.tgshell(path[1:])
2788     }
2789     fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
2790 }
2791 func (gshCtx *GshContext) httpServer(argv []string){
2792     http.HandleFunc("/", httpHandler)
2793     accport := "localhost:9999"
2794     fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2795     http.ListenAndServe(accport,nil)
2796 }
2797 func (gshCtx *GshContext) xGo(argv []string){
2798     go gshCtx.gshellv(argv[1:]);
2799 }
2800 func (gshCtx *GshContext) xPs(argv []string){}
2801 }
2802
2803 // <a name="plugin">Plugin</a>
2804 // plugin [-ls [names]] to list plugins
2805 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code
2806 func (gshCtx *GshContext) whichPlugin(name string,argv []string)(pi *PluginInfo){
2807     pi = nil
2808     for _,p := range gshCtx.PluginFuncs {
2809         if p.Name == name && pi == nil {
2810             pi = p
2811         }
2812         if !isin("-s",argv){
2813             //fmt.Printf("%v %v ",i,p)
2814             if isin("-ls",argv){
2815                 showFileInfo(p.Path,argv)
2816             }else{
2817                 fmt.Printf("%s\n",p.Name)
2818             }
2819         }
2820     }
2821     return pi
2822 }
2823 func (gshCtx *GshContext) xPlugin(argv []string) (error) {
2824     if len(argv) == 0 || argv[0] == "-ls" {
2825         gshCtx.whichPlugin("",argv)
2826         return nil
2827     }
2828     name := argv[0]
2829     Pin := gshCtx.whichPlugin(name, []string{"-s"})
2830     if Pin != nil {
2831         os.Args = argv // should be recovered?
2832         Pin.Addr.(func())()
2833         return nil
2834     }
2835     sofile := toFullPath(argv[0] + ".so") // or find it by which($PATH)
2836
2837     p, err := plugin.Open(sofile)
2838     if err != nil {
2839         fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2840         return err
2841     }
2842     fname := "Main"
2843     f, err := p.Lookup(fname)
2844     if( err != nil ){
2845         fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2846         return err
2847     }
2848     pin := PluginInfo {p,f,name,sofile}
2849     gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2850     fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2851
2852     //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2853     os.Args = argv
2854     f.(func())()
2855     return err
2856 }
2857 func (gshCtx*GshContext)Args(argv []string){
2858     for i,v := range os.Args {
2859         fmt.Printf("[%v] %v\n",i,v)
2860     }
2861 }
2862 func (gshCtx *GshContext) showVersion(argv []string){
2863     if isin("-l",argv) {
2864         fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2865     }else{
2866         fmt.Printf("%v",VERSION);
2867     }
2868     if isin("-a",argv) {
2869         fmt.Printf(" %s",AUTHOR)
2870     }
2871     if !isin("-n",argv) {
2872         fmt.Printf("\n")
2873     }
2874 }

```

```

2875
2876 // <a name="scanf">Scanf</a> // string decomposer
2877 // scanf [format] [input]
2878 func scanf(sstr string)(strv[]string){
2879     strv = strings.Split(sstr, " ")
2880     return strv
2881 }
2882 func scanUntil(src,end string)(rstr string, leng int){
2883     idx := strings.Index(src,end)
2884     if 0 <= idx {
2885         rstr = src[0:idx]
2886         return rstr,idx+leng(end)
2887     }
2888     return src,0
2889 }
2890
2891 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting
2892 func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
2893     //vint,err := strconv.Atoi(vstr)
2894     var ival int64 = 0
2895     n := 0
2896     err := error(nil)
2897     if strBegins(vstr, ".") {
2898         vx,_ := strconv.Atoi(vstr[1:])
2899         if vx < len(gsh.iValues) {
2900             vstr = gsh.iValues[vx]
2901         }else{
2902             }
2903     }
2904     // should use Eval()
2905     if strBegins(vstr, "0x") {
2906         n,err = fmt.Sscanf(vstr[2:], "%x", &ival)
2907     }else{
2908         n,err = fmt.Sscanf(vstr, "%d", &ival)
2909     //fmt.Printf("--D-- n=%d err=(%v) {%s}=%v\n",n,err,vstr, ival)
2910     }
2911     if n == 1 && err == nil {
2912         //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
2913         fmt.Printf("%"+fmts,ival)
2914     }else{
2915         if isin("-bn",optv){
2916             fmt.Printf("%"+fmts,filepath.Base(vstr))
2917         }else{
2918             fmt.Printf("%"+fmts,vstr)
2919         }
2920     }
2921 }
2922 func (gsh*GshContext)printfv(fmts,div string,argv[]string,optv[]string,list[]string){
2923     //fmt.Printf("%d",len(list))
2924     //curfmt := "v"
2925     outlen := 0
2926     curfmt := gsh.iFormat
2927
2928     if 0 < len(fmts) {
2929         for xi := 0; xi < len(fmts); xi++ {
2930             fch := fmts[xi]
2931             if fch == '%' {
2932                 if xi+1 < len(fmts) {
2933                     curfmt = string(fmts[xi+1])
2934                 }
2935                 gsh.iFormat = curfmt
2936                 xi += 1
2937                 if xi+1 < len(fmts) && fmts[xi+1] == '(' {
2938                     vals,leng := scanUntil(fmts[xi+2:],")")
2939                     //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n",curfmt,vals,leng)
2940                     gsh.printVal(curfmt,vals,optv)
2941                     xi += 2+leng-1
2942                     outlen += 1
2943                 }
2944                 continue
2945             }
2946             if fch == '-' {
2947                 hi,leng := scanInt(fmts[xi+1:])
2948                 if 0 < leng {
2949                     if hi < len(gsh.iValues) {
2950                         gsh.printVal(curfmt,gsh.iValues[hi],optv)
2951                         outlen += 1 // should be the real length
2952                     }else{
2953                         fmt.Printf("(out-range)")
2954                     }
2955                     xi += leng
2956                     continue;
2957                 }
2958             }
2959             fmt.Printf("%c",fch)
2960             outlen += 1
2961         }
2962     }else{
2963         //fmt.Printf("--D-- print {%s}\n")
2964         for i,v := range list {
2965             if 0 < i {
2966                 fmt.Printf(div)
2967             }
2968             gsh.printVal(curfmt,v,optv)
2969             outlen += 1
2970         }
2971     }
2972     if 0 < outlen {
2973         fmt.Printf("\n")
2974     }
2975 }
2976 func (gsh*GshContext)Scanv(argv[]string){
2977     //fmt.Printf("--D-- Scnav(%v)\n",argv)
2978     if len(argv) == 1 {
2979         return
2980     }
2981     argv = argv[1:]
2982     fmts := ""
2983     if strBegins(argv[0],"-F") {
2984         fmts = argv[0]
2985         gsh.iDelimiter = fmts
2986         argv = argv[1:]
2987     }
2988     input := strings.Join(argv, " ")
2989     if fmts == "" { // simple decomposition
2990         v := scanf(input)
2991         gsh.iValues = v
2992         //fmt.Printf("%v\n",strings.Join(v, ","))
2993     }else{
2994         v := make([]string,8)
2995         n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
2996         fmt.Printf("--D-- Scnav ->(%v) n=%d err=(%v)\n",v,n,err)
2997         gsh.iValues = v
2998     }
2999 }

```



```

3000 func (gsh*GshContext)Printv(argv []string){
3001     if false { //@@@
3002         fmt.Printf("%v\n", strings.Join(argv[1:], " "))
3003         return
3004     }
3005     //fmt.Printf("--D-- Printv(%v)\n", argv)
3006     //fmt.Printf("%v\n", strings.Join(gsh.iValues, ","))
3007     div := gsh.iDelimiter
3008     fmts := ""
3009     argv = argv[1:]
3010     if 0 < len(argv) {
3011         if strBegins(argv[0], "-F") {
3012             div = argv[0][2:]
3013             argv = argv[1:]
3014         }
3015     }
3016 }
3017 optv := []string{}
3018 for _,v := range argv {
3019     if strBegins(v, "-"){
3020         optv = append(optv,v)
3021         argv = argv[1:]
3022     }else{
3023         break;
3024     }
3025 }
3026 if 0 < len(argv) {
3027     fmts = strings.Join(argv, " ")
3028 }
3029 gsh.printfv(fmts,div,argv,optv,gsh.iValues)
3030 }
3031 func (gsh*GshContext)Basename(argv []string){
3032     for i,v := range gsh.iValues {
3033         gsh.iValues[i] = filepath.Base(v)
3034     }
3035 }
3036 func (gsh*GshContext)Sortv(argv []string){
3037     sv := gsh.iValues
3038     sort.Slice(sv, func(i,j int) bool {
3039         return sv[i] < sv[j]
3040     })
3041 }
3042 func (gsh*GshContext)Shiftv(argv []string){
3043     vi := len(gsh.iValues)
3044     if 0 < vi {
3045         if isin("-r",argv) {
3046             top := gsh.iValues[0]
3047             gsh.iValues = append(gsh.iValues[1:],top)
3048         }else{
3049             gsh.iValues = gsh.iValues[1:]
3050         }
3051     }
3052 }
3053 }
3054 func (gsh*GshContext)Enq(argv []string){
3055 }
3056 func (gsh*GshContext)Deq(argv []string){
3057 }
3058 func (gsh*GshContext)Push(argv []string){
3059     gsh.iValStack = append(gsh.iValStack,argv[1:])
3060     fmt.Printf("depth=%d\n",len(gsh.iValStack))
3061 }
3062 func (gsh*GshContext)Dump(argv []string){
3063     for i,v := range gsh.iValStack {
3064         fmt.Printf("%d %v\n",i,v)
3065     }
3066 }
3067 func (gsh*GshContext)Pop(argv []string){
3068     depth := len(gsh.iValStack)
3069     if 0 < depth {
3070         v := gsh.iValStack[depth-1]
3071         if isin("-cat",argv){
3072             gsh.iValues = append(gsh.iValues,v...)
3073         }else{
3074             gsh.iValues = v
3075         }
3076         gsh.iValStack = gsh.iValStack[0:depth-1]
3077         fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3078     }else{
3079         fmt.Printf("depth=%d\n",depth)
3080     }
3081 }
3082 }
3083 // <a name="interpreter">Command Interpreter</a>
3084 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3085     fin = false
3086 }
3087 if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3088 if len(argv) <= 0 {
3089     return false
3090 }
3091 xargv := []string{}
3092 for ai := 0; ai < len(argv); ai++ {
3093     xargv = append(xargv, strsubst(gshCtx,argv[ai],false))
3094 }
3095 argv = xargv
3096 if false {
3097     for ai := 0; ai < len(argv); ai++ {
3098         fmt.Printf("[%d] %s [%d]T\n",
3099             ai,argv[ai],len(argv[ai]),argv[ai])
3100     }
3101 }
3102 cmd := argv[0]
3103 if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
3104 switch { // https://tour.golang.org/flowcontrol/11
3105 case cmd == "":
3106     gshCtx.xPwd([]string{}); // empty command
3107 case cmd == "-x":
3108     gshCtx.CmdTrace = ! gshCtx.CmdTrace
3109 case cmd == "-xt":
3110     gshCtx.CmdTime = ! gshCtx.CmdTime
3111 case cmd == "-ot":
3112     gshCtx.sconnect(true, argv)
3113 case cmd == "-ou":
3114     gshCtx.sconnect(false, argv)
3115 case cmd == "-it":
3116     gshCtx.saccept(true, argv)
3117 case cmd == "-iu":
3118     gshCtx.saccept(false, argv)
3119 case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3120     gshCtx.redirect(argv)
3121 case cmd == "|":
3122     gshCtx.fromPipe(argv)
3123 case cmd == "args":
3124     gshCtx.Args(argv)

```

```

3125 case cmd == "bg" || cmd == "-bg":
3126     rfin := gshCtx.inBackground(argv[1:])
3127     return rfin
3128 case cmd == "-bn":
3129     gshCtx.Basename(argv)
3130 case cmd == "call":
3131     _,_ = gshCtx.excommand(false,argv[1:])
3132 case cmd == "cd" || cmd == "chdir":
3133     gshCtx.xChdir(argv);
3134 case cmd == "-cksum":
3135     gshCtx.xFind(argv)
3136 case cmd == "-sum":
3137     gshCtx.xFind(argv)
3138 case cmd == "close":
3139     gshCtx.xClose(argv)
3140 case cmd == "gcp":
3141     gshCtx.FileCopy(argv)
3142 case cmd == "dec" || cmd == "decode":
3143     gshCtx.Dec(argv)
3144 case cmd == "#define":
3145 case cmd == "dic" || cmd == "d":
3146     xDic(argv)
3147 case cmd == "dump":
3148     gshCtx.Dump(argv)
3149 case cmd == "echo" || cmd == "e":
3150     echo(argv,true)
3151 case cmd == "enc" || cmd == "encode":
3152     gshCtx.Enc(argv)
3153 case cmd == "env":
3154     env(argv)
3155 case cmd == "eval":
3156     xEval(argv[1:],true)
3157 case cmd == "ev" || cmd == "events":
3158     dumpEvents(argv)
3159 case cmd == "exec":
3160     _,_ = gshCtx.excommand(true,argv[1:])
3161     // should not return here
3162 case cmd == "exit" || cmd == "quit":
3163     // write Result code EXIT to 3>
3164     return true
3165 case cmd == "fds":
3166     // dump the attributes of fds (of other process)
3167 case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3168     gshCtx.xFind(argv[1:])
3169 case cmd == "fu":
3170     gshCtx.xFind(argv[1:])
3171 case cmd == "fork":
3172     // mainly for a server
3173 case cmd == "-gen":
3174     gshCtx.gen(argv)
3175 case cmd == "-go":
3176     gshCtx.xGo(argv)
3177 case cmd == "-grep":
3178     gshCtx.xFind(argv)
3179 case cmd == "gdeg":
3180     gshCtx.Deg(argv)
3181 case cmd == "genq":
3182     gshCtx.Enq(argv)
3183 case cmd == "gpop":
3184     gshCtx.Pop(argv)
3185 case cmd == "gpush":
3186     gshCtx.Push(argv)
3187 case cmd == "history" || cmd == "hi": // hi should be alias
3188     gshCtx.xHistory(argv)
3189 case cmd == "jobs":
3190     gshCtx.xJobs(argv)
3191 case cmd == "lisp" || cmd == "nlsp":
3192     gshCtx.SplitLine(argv)
3193 case cmd == "-ls":
3194     gshCtx.xFind(argv)
3195 case cmd == "nop":
3196     // do nothing
3197 case cmd == "pipe":
3198     gshCtx.xOpen(argv)
3199 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3200     gshCtx.xPlugin(argv[1:])
3201 case cmd == "print" || cmd == "-pr":
3202     // output internal slice // also sprintf should be
3203     gshCtx.Printf(argv)
3204 case cmd == "ps":
3205     gshCtx.xPs(argv)
3206 case cmd == "pstable":
3207     // to be gsh.title
3208 case cmd == "rexe" || cmd == "rexd":
3209     gshCtx.RexecServer(argv)
3210 case cmd == "rexe" || cmd == "rex":
3211     gshCtx.RexecClient(argv)
3212 case cmd == "repeat" || cmd == "rep": // repeat cond command
3213     gshCtx.repeat(argv)
3214 case cmd == "replay":
3215     gshCtx.xReplay(argv)
3216 case cmd == "scan":
3217     // scan input (or so in fscanf) to internal slice (like Files or map)
3218     gshCtx.Scanv(argv)
3219 case cmd == "set":
3220     // set name ...
3221 case cmd == "serv":
3222     gshCtx.httpServer(argv)
3223 case cmd == "shift":
3224     gshCtx.Shiftv(argv)
3225 case cmd == "sleep":
3226     gshCtx.sleep(argv)
3227 case cmd == "-sort":
3228     gshCtx.Sortv(argv)
3229
3230 case cmd == "j" || cmd == "join":
3231     gshCtx.RJoin(argv)
3232 case cmd == "a" || cmd == "alpa":
3233     gshCtx.Rexec(argv)
3234 case cmd == "jcd" || cmd == "jchdir":
3235     gshCtx.Rchdir(argv)
3236 case cmd == "jget":
3237     gshCtx.Rget(argv)
3238 case cmd == "jls":
3239     gshCtx.Rls(argv)
3240 case cmd == "jput":
3241     gshCtx.Rput(argv)
3242 case cmd == "jpwd":
3243     gshCtx.Rpwd(argv)
3244
3245 case cmd == "time":
3246     fin = gshCtx.xTime(argv)
3247 case cmd == "ungets":
3248     if 1 < len(argv) {
3249         ungets(argv[1]+\n")

```

```

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

```

```

3375 {
3376 // if isatty {
3377     fmt.Printf("%d",hix)
3378     fmt.Print(PROMPT)
3379 // }
3380 reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3381 line, _, _ := reader.ReadLine()
3382 return string(line)
3383 }
3384 }
3385
3386 //== begin ===== getline
3387 /*
3388  * getline.c
3389  * 2020-0819 extracted from dog.c
3390  * getline.go
3391  * 2020-0822 ported to Go
3392  */
3393 /*
3394 package main // getline main
3395 import (
3396     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3397     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3398     "os" // <a href="https://golang.org/pkg/os/">os</a>
3399     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3400     //"bytes" // <a href="https://golang.org/pkg/bytes/">bytes</a>
3401     //"os/exec" // <a href="https://golang.org/pkg/os/exec/">os/exec</a>
3402 )
3403 */
3404
3405 // C language compatibility functions
3406 var errno = 0
3407 var stdin *os.File = os.Stdin
3408 var stdout *os.File = os.Stdout
3409 var stderr *os.File = os.Stderr
3410 var EOF = -1
3411 var NULL = 0
3412 type FILE os.File
3413 type StrBuff []byte
3414 var NULL_FP *os.File = nil
3415 var NULLSP = 0
3416 //var LINESIZE = 1024
3417
3418 func system(cmdstr string)(int){
3419     PA := syscall.ProcAttr {
3420         "", // the starting directory
3421         os.Environ(),
3422         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3423         nil,
3424     }
3425     argv := strings.Split(cmdstr, " ")
3426     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3427     if( err != nil ){
3428         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3429     }
3430     syscall.Wait4(pid,nil,0,nil)
3431
3432     /*
3433     argv := strings.Split(cmdstr, " ")
3434     fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3435     //cmd := exec.Command(argv[0],...)
3436     cmd := exec.Command(argv[0],argv[1],argv[2])
3437     cmd.Stdin = strings.NewReader("output of system")
3438     var out bytes.Buffer
3439     cmd.Stdout = &out
3440     var serr bytes.Buffer
3441     cmd.Stderr = &serr
3442     err := cmd.Run()
3443     if err != nil {
3444         fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)
3445         fmt.Printf("ERR:%s\n",serr.String())
3446     }else{
3447         fmt.Printf("%s",out.String())
3448     }
3449     */
3450     return 0
3451 }
3452 func atoi(str string)(ret int){
3453     ret,err := fmt.Sscanf(str,"%d",ret)
3454     if err == nil {
3455         return ret
3456     }else{
3457         // should set errno
3458         return 0
3459     }
3460 }
3461 func getenv(name string)(string){
3462     val,got := os.LookupEnv(name)
3463     if got {
3464         return val
3465     }else{
3466         return "?"
3467     }
3468 }
3469 func strcpy(dst StrBuff, src string){
3470     var i int
3471     srcb := []byte(src)
3472     for i = 0; i < len(src) && srcb[i] != 0; i++ {
3473         dst[i] = srcb[i]
3474     }
3475     dst[i] = 0
3476 }
3477 func xstrcpy(dst StrBuff, src StrBuff){
3478     dst = src
3479 }
3480 func strcat(dst StrBuff, src StrBuff){
3481     dst = append(dst,src...)
3482 }
3483 func strdup(str StrBuff)(string){
3484     return string(str[0:strlen(str)])
3485 }
3486 func sstrlen(str string)(int){
3487     return len(str)
3488 }
3489 func strlen(str StrBuff)(int){
3490     var i int
3491     for i = 0; i < len(str) && str[i] != 0; i++ {
3492     }
3493     return i
3494 }
3495 func sizeof(data StrBuff)(int){
3496     return len(data)
3497 }
3498 func isatty(fd int)(ret int){
3499     return 1

```

```

3500 }
3501
3502 func fopen(file string,mode string)(fp*os.File){
3503     if mode == "r" {
3504         fp,err := os.Open(file)
3505         if( err != nil ){
3506             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
3507             return NULL_FP;
3508         }
3509         return fp;
3510     }else{
3511         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3512         if( err != nil ){
3513             return NULL_FP;
3514         }
3515         return fp;
3516     }
3517 }
3518 func fclose(fp*os.File){
3519     fp.Close()
3520 }
3521 func fflush(fp *os.File)(int){
3522     return 0
3523 }
3524 func fgetc(fp*os.File)(int){
3525     var buf [1]byte
3526     _,err := fp.Read(buf[0:1])
3527     if( err != nil ){
3528         return EOF;
3529     }else{
3530         return int(buf[0])
3531     }
3532 }
3533 func sfgets(str*string, size int, fp*os.File)(int){
3534     buf := make(StrBuff,size)
3535     var ch int
3536     var i int
3537     for i = 0; i < len(buf)-1; i++ {
3538         ch = fgetc(fp)
3539         //fprintf(stderr,"--fgets %d/%d %X\n",i,len(buf),ch)
3540         if( ch == EOF ){
3541             break;
3542         }
3543         buf[i] = byte(ch);
3544         if( ch == '\n' ){
3545             break;
3546         }
3547     }
3548     buf[i] = 0
3549     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3550     return i
3551 }
3552 func fgets(buf StrBuff, size int, fp*os.File)(int){
3553     var ch int
3554     var i int
3555     for i = 0; i < len(buf)-1; i++ {
3556         ch = fgetc(fp)
3557         //fprintf(stderr,"--fgets %d/%d %X\n",i,len(buf),ch)
3558         if( ch == EOF ){
3559             break;
3560         }
3561         buf[i] = byte(ch);
3562         if( ch == '\n' ){
3563             break;
3564         }
3565     }
3566     buf[i] = 0
3567     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3568     return i
3569 }
3570 func fputc(ch int , fp*os.File)(int){
3571     var buf [1]byte
3572     buf[0] = byte(ch)
3573     fp.Write(buf[0:1])
3574     return 0
3575 }
3576 func fputs(buf StrBuff, fp*os.File)(int){
3577     fp.Write(buf)
3578     return 0
3579 }
3580 func xputss(str string, fp*os.File)(int){
3581     return fputs([]byte(str),fp)
3582 }
3583 func sscanf(str StrBuff,fmts string, params ...interface{})(int){
3584     fmt.Sscanf(string(str[0:strlen(str)]),fmts,params...)
3585     return 0
3586 }
3587 func fprintf(fp*os.File,fmts string, params ...interface{})(int){
3588     fmt.Fprintf(fp,fmts,params...)
3589     return 0
3590 }
3591
3592 // <a name="IME">Command Line IME</a>
3593 //----- MyIME
3594 var MyIMEVER = "MyIME/0.0.2";
3595 type RomKana struct {
3596     dic string // dictionary ID
3597     pat string // input pattern
3598     out string // output pattern
3599     hit int64 // count of hit and used
3600 }
3601 var dicents = 0
3602 var romkana [1024]RomKana
3603 var Romkan []RomKana
3604
3605 func isinDic(str string)(int){
3606     for i,v := range Romkan {
3607         if v.pat == str {
3608             return i
3609         }
3610     }
3611     return -1
3612 }
3613 const (
3614     DIC_COM_LOAD = "im"
3615     DIC_COM_DUMP = "s"
3616     DIC_COM_LIST = "ls"
3617     DIC_COM_ENA = "en"
3618     DIC_COM_DIS = "di"
3619 )
3620 func helpDic(argv []string){
3621     out := stderr
3622     cmd := ""
3623     if 0 < len(argv) { cmd = argv[0] }
3624     fprintf(out,"--- %v Usage\n",cmd)

```

```

3625     fprintf(out, "... Commands\n")
3626     fprintf(out, "...   %v %v [dicName] [dicURL ] -- Import dictionary\n", cmd, DIC_COM_LOAD)
3627     fprintf(out, "...   %v %v [pattern] -- Search in dictionary\n", cmd, DIC_COM_DUMP)
3628     fprintf(out, "...   %v %v [dicName] -- List dictionaries\n", cmd, DIC_COM_LIST)
3629     fprintf(out, "...   %v %v [dicName] -- Disable dictionaries\n", cmd, DIC_COM_DIS)
3630     fprintf(out, "...   %v %v [dicName] -- Enable dictionaries\n", cmd, DIC_COM_ENA)
3631     fprintf(out, "... Keys ... %v\n", "ESC can be used for '\\')
3632     fprintf(out, "...   %v -- Reverse the case of the last character\n",)
3633     fprintf(out, "...   %v -- Replace input with translated text\n",)
3634     fprintf(out, "...   %v -- On/Off translation mode\n",)
3635     fprintf(out, "...   %v -- Force Lower Case\n",)
3636     fprintf(out, "...   %v -- Force Upper Case (software CapsLock)\n",)
3637     fprintf(out, "...   %v -- Show translation actions\n",)
3638     fprintf(out, "...   %v -- Replace the last input character with it Hexa-Decimal\n",)
3639 }
3640 func xDic(argv[]string){
3641     if len(argv) <= 1 {
3642         helpDic(argv)
3643         return
3644     }
3645     argv = argv[1:]
3646     var debug = false
3647     var info = false
3648     var silent = false
3649     var dump = false
3650     var builtin = false
3651     cmd := argv[0]
3652     argv = argv[1:]
3653     opt := ""
3654     arg := ""
3655
3656     if 0 < len(argv) {
3657         arg1 := argv[0]
3658         if arg1[0] == '-' {
3659             switch arg1 {
3660                 default: fmt.Printf("--Ed-- Unknown option(%v)\n", arg1)
3661                     return
3662                 case "-b": builtin = true
3663                     case "-d": debug = true
3664                     case "-s": silent = true
3665                     case "-v": info = true
3666             }
3667             opt = arg1
3668             argv = argv[1:]
3669         }
3670     }
3671
3672     dicName := ""
3673     dicURL := ""
3674     if 0 < len(argv) {
3675         arg = argv[0]
3676         dicName = arg
3677         argv = argv[1:]
3678     }
3679     if 0 < len(argv) {
3680         dicURL = argv[0]
3681         argv = argv[1:]
3682     }
3683     if false {
3684         fprintf(stderr, "--Dd-- com(%v) opt(%v) arg(%v)\n", cmd, opt, arg)
3685     }
3686     if cmd == DIC_COM_LOAD {
3687         //dicType := ""
3688         dicBody := ""
3689         if !builtin && dicName != "" && dicURL == "" {
3690             f, err := os.Open(dicName)
3691             if err == nil {
3692                 dicURL = dicName
3693             } else {
3694                 f, err = os.Open(dicName+".html")
3695                 if err == nil {
3696                     dicURL = dicName+".html"
3697                 } else {
3698                     f, err = os.Open("gshdic-"+dicName+".html")
3699                     if err == nil {
3700                         dicURL = "gshdic-"+dicName+".html"
3701                     }
3702                 }
3703             }
3704             if err == nil {
3705                 var buf = make([]byte, 128*1024)
3706                 count, err := f.Read(buf)
3707                 f.Close()
3708                 if info {
3709                     fprintf(stderr, "--Id-- ReadDic(%v,%v)\n", count, err)
3710                 }
3711                 dicBody = string(buf[0:count])
3712             }
3713         }
3714         if dicBody == "" {
3715             switch arg {
3716                 default:
3717                     dicName = "WorldDic"
3718                     dicURL = WorldDic
3719                     if info {
3720                         fprintf(stderr, "--Id-- default dictionary \"%v\"\n",
3721                             dicName);
3722                     }
3723                 case "wnn":
3724                     dicName = "WnnDic"
3725                     dicURL = WnnDic
3726                 case "sumomo":
3727                     dicName = "SumomoDic"
3728                     dicURL = SumomoDic
3729                 case "sijimi":
3730                     dicName = "SijimiDic"
3731                     dicURL = SijimiDic
3732                 case "jkl":
3733                     dicName = "JKLJaDic"
3734                     dicURL = JA_JKLDic
3735             }
3736             if debug {
3737                 fprintf(stderr, "--Id-- %v URL=%v\n", dicName, dicURL);
3738             }
3739             dicv := strings.Split(dicURL, ",")
3740             if debug {
3741                 fprintf(stderr, "--Id-- %v encoded data...\n", dicName)
3742                 fprintf(stderr, "Type: %v\n", dicv[0])
3743                 fprintf(stderr, "Body: %v\n", dicv[1])
3744                 fprintf(stderr, "\n")
3745             }
3746             body, _ := base64.StdEncoding.DecodeString(dicv[1])
3747             dicBody = string(body)
3748         }
3749         if info {

```

```

3750         fmt.Printf("--Id-- %v %v\n",dicName,dicURL)
3751         fmt.Printf("%s\n",dicBody)
3752     }
3753     if debug {
3754         fprintf(stderr,"--Id-- dicName %v text...\n",dicName)
3755         fprintf(stderr,"%v\n",string(dicBody))
3756     }
3757     entv := strings.Split(dicBody,"\n");
3758     if info {
3759         fprintf(stderr,"--Id-- %v scan...\n",dicName);
3760     }
3761     var added int = 0
3762     var dup int = 0
3763     for i,v := range entv {
3764         var pat string
3765         var out string
3766         fmt.Sscanf(v,"%s %s",&pat,&out)
3767         if len(pat) <= 0 {
3768             }else{
3769             if 0 <= isinDic(pat) {
3770                 dup += 1
3771                 continue
3772             }
3773             romkana[dicents] = RomKana{dicName,pat,out,0}
3774             dicents += 1
3775             added += 1
3776             Romkan = append(Romkan,RomKana{dicName,pat,out,0})
3777             if debug {
3778                 fmt.Printf("[%3v]:[%2v]%-8v [%2v]%-8v\n",
3779                     i,len(pat),pat,len(out),out)
3780             }
3781         }
3782     }
3783     if !silent {
3784         url := dicURL
3785         if strBegins(url,"data:") {
3786             url = "builtin"
3787         }
3788         fprintf(stderr,"--Id-- %v scan... %v added, %v dup. / %v total (%v)\n",
3789             dicName,added,dup,len(Romkan),url);
3790     }
3791     // should sort by pattern length for conplete match, for performance
3792     if debug {
3793         arg = "" // search pattern
3794         dump = true
3795     }
3796 }
3797 if cmd == DIC_COM_DUMP || dump {
3798     fprintf(stderr,"--Id-- %v dump... %v entries:\n",dicName,len(Romkan));
3799     var match = 0
3800     for i := 0; i < len(Romkan); i++ {
3801         dic := Romkan[i].dic
3802         pat := Romkan[i].pat
3803         out := Romkan[i].out
3804         if arg == "" || 0 <= strings.Index(pat,arg) || 0 <= strings.Index(out,arg) {
3805             fmt.Printf("\\\\%v\t%v [%2v]%-8v [%2v]%-8v\n",
3806                 i,dic,len(pat),pat,len(out),out)
3807             match += 1
3808         }
3809     }
3810     fprintf(stderr,"--Id-- %v matched %v / %v entries:\n",arg,match,len(Romkan));
3811 }
3812 }
3813 func loadDefaultDic(dic int){
3814     if( 0 < len(Romkan) ){
3815         return
3816     }
3817     //fprintf(stderr,"%v\n")
3818     xDic([]string{"dic",DIC_COM_LOAD});
3819 }
3820 var info = false
3821 if info {
3822     fprintf(stderr,"--Id-- Conguratations!! WorldDic is now activated.\r\n")
3823     fprintf(stderr,"--Id-- enter \"dic\" command for help.\r\n")
3824 }
3825 }
3826 func readDic()(int){
3827     /*
3828     var rk *os.File;
3829     var dic = "MyIME-dic.txt";
3830     //rk = fopen("romkana.txt","r");
3831     //rk = fopen("JK-JA-morse-dic.txt","r");
3832     rk = fopen(dic,"r");
3833     if( rk == NULL FP ){
3834         if( true ){
3835             fprintf(stderr,"--%s-- Could not load %s\n",MyIMEVER,dic);
3836         }
3837         return -1;
3838     }
3839     if( true ){
3840         var di int;
3841         var line = make(StrBuff,1024);
3842         var pat string
3843         var out string
3844         for di = 0; di < 1024; di++ {
3845             if( fgets(line,sizeof(line),rk) == NULLSP ){
3846                 break;
3847             }
3848             fmt.Sscanf(string(line[0:strlen(line)]),"%s %s",&pat,&out);
3849             //sscanf(line,"%s %[\r\n]",&pat,&out);
3850             romkana[di].pat = pat;
3851             romkana[di].out = out;
3852             //fprintf(stderr,"--Dd- %-10s %s\n",pat,out)
3853         }
3854         dicents += di
3855         if( false ){
3856             fprintf(stderr,"--%s-- loaded romkana.txt [%d]\n",MyIMEVER,di);
3857             for di = 0; di < dicents; di++ {
3858                 fprintf(stderr,
3859                     "%s %s\n",romkana[di].pat,romkana[di].out);
3860             }
3861         }
3862     }
3863     fclose(rk);
3864 }
3865 //romkana[dicents].pat = "//ddump"
3866 //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3867 /*
3868     return 0;
3869 }
3870 func matchlen(stri string, pati string)(int){
3871     if strBegins(stri,pati) {
3872         return len(pati)
3873     }else{
3874         return 0

```

```

3875     }
3876 }
3877 func convs(src string)(string){
3878     var si int;
3879     var sx = len(src);
3880     var di int;
3881     var mi int;
3882     var dstb []byte
3883
3884     for si = 0; si < sx; { // search max. match from the position
3885         if strBegins(src[si:], "%x/") {
3886             // %x/integer/ // s/a/b/
3887             ix := strings.Index(src[si+3:], "/")
3888             if 0 < ix {
3889                 var iv int = 0
3890                 //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3891                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3892                 sval := fmt.Sprintf("%x", iv)
3893                 bval := []byte(sval)
3894                 dstb = append(dstb, bval...)
3895                 si = si+3+ix+1
3896                 continue
3897             }
3898         }
3899         if strBegins(src[si:], "%d/") {
3900             // %d/integer/ // s/a/b/
3901             ix := strings.Index(src[si+3:], "/")
3902             if 0 < ix {
3903                 var iv int = 0
3904                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3905                 sval := fmt.Sprintf("%d", iv)
3906                 bval := []byte(sval)
3907                 dstb = append(dstb, bval...)
3908                 si = si+3+ix+1
3909                 continue
3910             }
3911         }
3912         if strBegins(src[si:], "%t") {
3913             now := time.Now()
3914             if true {
3915                 date := now.Format(time.Stamp)
3916                 dstb = append(dstb, []byte(date)...)
3917                 si = si+3
3918             }
3919             continue
3920         }
3921         var maxlen int = 0;
3922         var len int;
3923         mi = -1;
3924         for di = 0; di < dicents; di++ {
3925             len = matchlen(src[si:], romkana[di].pat);
3926             if( maxlen < len ){
3927                 maxlen = len;
3928                 mi = di;
3929             }
3930         }
3931         if( 0 < maxlen ){
3932             out := romkana[mi].out;
3933             dstb = append(dstb, []byte(out)...);
3934             si += maxlen;
3935         }else{
3936             dstb = append(dstb, src[si])
3937             si += 1;
3938         }
3939     }
3940     return string(dstb)
3941 }
3942 func trans(src string)(int){
3943     dst := convs(src);
3944     xfprintf(stderr);
3945     return 0;
3946 }
3947
3948 //----- LINEEDIT
3949 // "?" at the top of the line means searching history
3950
3951 // should be compatilbe with Telnet
3952 const (
3953     EV_MODE      = 255
3954     EV_IDLE      = 254
3955     EV_TIMEOUT   = 253
3956
3957     GO_UP        = 252 // k
3958     GO_DOWN      = 251 // j
3959     GO_RIGHT     = 250 // l
3960     GO_LEFT      = 249 // h
3961     DEL_RIGHT    = 248 // x
3962     GO_TOPL      = 'A'-0x40 // 0
3963     GO_ENDL      = 'E'-0x40 // $
3964
3965     GO_TOPW      = 239 // b
3966     GO_ENDW      = 238 // e
3967     GO_NEXTW     = 237 // w
3968
3969     GO_FORWCH    = 229 // f
3970     GO_PAIRCH    = 228 // %
3971
3972     GO_DEL       = 219 // d
3973
3974     HI_SRCH_FW   = 209 // /
3975     HI_SRCH_BK   = 208 // ?
3976     HI_SRCH_RFW  = 207 // n
3977     HI_SRCH_RBK  = 206 // N
3978 )
3979
3980 // should return number of octets ready to be read immediately
3981 //fprintf(stderr, "\n--Select(%v %v)\n", err, r.Bits[0])
3982
3983
3984 var EventRecvFd = -1 // file descriptor
3985 var EventSendFd = -1
3986 const EventFdOffset = 1000000
3987 const NormalFdOffset = 100
3988
3989 func putEvent(event int, evarg int){
3990     if true {
3991         if EventRecvFd < 0 {
3992             var pv = []int{-1, -1}
3993             syscall.Pipe(pv)
3994             EventRecvFd = pv[0]
3995             EventSendFd = pv[1]
3996             //fmt.Printf("--De-- EventPipe created[%v, %v]\n", EventRecvFd, EventSendFd)
3997         }
3998     }else{
3999         if EventRecvFd < 0 {

```



```

4000 // the document differs from this spec
4001 // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L340
4002 sv,err := syscall.Socketpair(syscall.AF_UNIX,syscall.SOCK_STREAM,0)
4003 EventRecvFd = sv[0]
4004 EventSendFd = sv[1]
4005 if err != nil {
4006     fmt.Printf("--De-- EventSock created[%v,%v](%v)\n",
4007         EventRecvFd,EventSendFd,err)
4008 }
4009 }
4010 }
4011 var buf = []byte{ byte(event)}
4012 n,err := syscall.Write(EventSendFd,buf)
4013 if err != nil {
4014     fmt.Printf("--De-- putEvent[%v]($3v)($v %v)\n",EventSendFd,event,n,err)
4015 }
4016 }
4017 func ungets(str string){
4018     for _,ch := range str {
4019         putEvent(int(ch),0)
4020     }
4021 }
4022 func (gsh*GshContext)xReplay(argv []string){
4023     hix := 0
4024     tempo := 1.0
4025     xtempo := 1.0
4026     repeat := 1
4027
4028     for _,a := range argv { // tempo
4029         if strBegins(a,"x") {
4030             fmt.Sscanf(a[1:], "%f",&xtempo)
4031             tempo = 1 / xtempo
4032             //fprintf(stderr,"--Dr-- tempo=[%v]%v\n",a[2:],tempo);
4033         }else
4034         if strBegins(a,"r") { // repeat
4035             fmt.Sscanf(a[1:], "%v",&repeat)
4036         }else
4037         if strBegins(a,"l") {
4038             fmt.Sscanf(a[1:], "%d",&hix)
4039         }else{
4040             fmt.Sscanf(a,"%d",&hix)
4041         }
4042     }
4043     if hix == 0 || len(argv) <= 1 {
4044         hix = len(gsh.CommandHistory)-1
4045     }
4046     fmt.Printf("--Ir-- Replay(!%v x%v r%v)\n",hix,xtempo,repeat)
4047     //dumpEvents(hix)
4048     //gsh.xScanReplay(hix,false,repeat,tempo,argv)
4049     go gsh.xScanReplay(hix,true,repeat,tempo,argv)
4050 }
4051 }
4052 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4053 // 2020-0827 GShell-0.2.3
4054 func FpollIn1(fp *os.File,usec int)(uintptr){
4055     nfd := 1
4056
4057     rdv := syscall.FdSet {}
4058     fd1 := fp.Fd()
4059     bank1 := fd1/32
4060     mask1 := int32(1 << fd1)
4061     rdv.Bits[bank1] = mask1
4062
4063     fd2 := -1
4064     bank2 := -1
4065     var mask2 int32 = 0
4066
4067     if 0 <= EventRecvFd {
4068         fd2 = EventRecvFd
4069         nfd = fd2 + 1
4070         bank2 = fd2/32
4071         mask2 = int32(1 << fd2)
4072         rdv.Bits[bank2] |= mask2
4073         //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n",fd2,bank2,mask2)
4074     }
4075
4076     tout := syscall.NsecToTimeval(int64(usec*1000))
4077     //n,err := syscall.Select(nfd,&rdv,nil,nil,&tout) // spec. mismatch
4078     err := syscall.Select(nfd,&rdv,nil,nil,&tout)
4079     if err != nil {
4080         //fmt.Printf("--De-- select() err(%v)\n",err)
4081     }
4082     if err == nil {
4083         if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4084             if false {
4085                 fmt.Printf("--De-- got Event\n")
4086             }
4087             return uintptr(EventFdOffset + fd2)
4088         }else
4089         if (rdv.Bits[bank1] & mask1) != 0 {
4090             return uintptr(NormalFdOffset + fd1)
4091         }else{
4092             return 1
4093         }
4094     }else{
4095         return 0
4096     }
4097 }
4098 func fgetoTimeout1(fp *os.File,usec int)(int){
4099     READ1:
4100     readyFd := FpollIn1(fp,usec)
4101     if readyFd < 100 {
4102         return EV_TIMEOUT
4103     }
4104
4105     var buf [1]byte
4106
4107     if EventFdOffset <= readyFd {
4108         fd := int(readyFd-EventFdOffset)
4109         _,err := syscall.Read(fd,buf[0:1])
4110         if( err != nil ){
4111             return EOF;
4112         }else{
4113             if buf[0] == EV_MODE {
4114                 recvEvent(fd)
4115                 goto READ1
4116             }
4117             return int(buf[0])
4118         }
4119     }
4120
4121     _,err := fp.Read(buf[0:1])
4122     if( err != nil ){
4123         return EOF;
4124     }else{

```

```

4125     return int(buf[0])
4126 }
4127 }
4128 }
4129 func visibleChar(ch int)(string){
4130     switch {
4131     case '!' <= ch && ch <= '-':
4132         return string(ch)
4133     }
4134     switch ch {
4135     case ' ': return "\\s"
4136     case '\n': return "\\n"
4137     case '\r': return "\\r"
4138     case '\t': return "\\t"
4139     }
4140     switch ch {
4141     case 0x00: return "NUL"
4142     case 0x07: return "BEL"
4143     case 0x08: return "BS"
4144     case 0x0E: return "SO"
4145     case 0x0F: return "SI"
4146     case 0x1B: return "ESC"
4147     case 0x7F: return "DEL"
4148     }
4149     switch ch {
4150     case EV_IDLE: return fmt.Sprintf("IDLE")
4151     case EV_MODE: return fmt.Sprintf("MODE")
4152     }
4153     return fmt.Sprintf("%X",ch)
4154 }
4155 func recvEvent(fd int){
4156     var buf = make([]byte,1)
4157     _,_ = syscall.Read(fd,buf[0:1])
4158     if( buf[0] != 0 ){
4159         romkanmode = true
4160     }else{
4161         romkanmode = false
4162     }
4163 }
4164 func (gsh*GshContext)xScanReplay(hix int,replay bool,repeat int,tempo float64,argv[string]){
4165     var Start time.Time
4166     var events = []Event{}
4167     for _,e := range Events {
4168         if hix == 0 || e.CmdIndex == hix {
4169             events = append(events,e)
4170         }
4171     }
4172     elen := len(events)
4173     if 0 < elen {
4174         if events[elen-1].event == EV_IDLE {
4175             events = events[0:elen-1]
4176         }
4177     }
4178     for r := 0; r < repeat; r++ {
4179         for i,e := range events {
4180             nano := e.when.Nanosecond()
4181             micro := nano / 1000
4182             if Start.Second() == 0 {
4183                 Start = time.Now()
4184             }
4185             diff := time.Now().Sub(Start)
4186             if replay {
4187                 if e.event != EV_IDLE {
4188                     putEvent(e.event,0)
4189                     if e.event == EV_MODE { // event with arg
4190                         putEvent(int(e.evarg),0)
4191                     }
4192                 }
4193             }else{
4194                 fmt.Printf("#7.3fms #%-3v !%-3v [%v.%06d] %3v %02X %%-4v %10.3fms\n",
4195                     float64(diff)/1000000.0,
4196                     i,
4197                     e.CmdIndex,
4198                     e.when.Format(time.Stamp),micro,
4199                     e.event,e.event,visibleChar(e.event),
4200                     float64(e.evarg)/1000000.0)
4201             }
4202             if e.event == EV_IDLE {
4203                 d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4204                 //nsleep(time.Duration(e.evarg))
4205                 nsleep(d)
4206             }
4207         }
4208     }
4209 }
4210 func dumpEvents(argv[string]){
4211     hix := 0
4212     if 1 < len(argv) {
4213         fmt.Sscanf(argv[1],"%d",&hix)
4214     }
4215     for i,e := range Events {
4216         nano := e.when.Nanosecond()
4217         micro := nano / 1000
4218         //if e.event != EV_TIMEOUT {
4219         if hix == 0 || e.CmdIndex == hix {
4220             fmt.Printf("#%-3v !%-3v [%v.%06d] %3v %02X %%-4v %10.3fms\n",i,
4221                 e.CmdIndex,
4222                 e.when.Format(time.Stamp),micro,
4223                 e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4224         }
4225         //}
4226     }
4227 }
4228 func fgetcTimeout(fp *os.File,usec int)(int){
4229     ch := fgetcTimeout1(fp,usec)
4230     if ch != EV_TIMEOUT {
4231         now := Time.Now()
4232         if 0 < len(Events) {
4233             last := Events[len(Events)-1]
4234             dura := int64(now.Sub(last.when))
4235             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4236         }
4237         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4238     }
4239     return ch
4240 }
4241 }
4242 var TtyMaxCol = 72 // to be obtained by ioctl?
4243 var EscTimeout = (100*1000)
4244 var (
4245     MODE_VicMode bool // vi compatible command mode
4246     MODE_ShowMode bool
4247     romkanmode bool // shown translation mode, the mode to be retained
4248     MODE_Recursive bool // recursive translation
4249     MODE_CapsLock bool // software CapsLock

```

```

4250 MODE_LowerLock bool // force lower-case character lock
4251 MODE_ViInsert int // visible insert mode, should be like "I" icon in X Window
4252 MODE_ViTrace bool // output newline before translation
4253 }
4254 type IInput struct {
4255     lno int
4256     lastlno int
4257     pch []int // input queue
4258     prompt string
4259     line string
4260     right string
4261     inMode bool
4262     pinMode bool
4263     waitingMeta string // waiting meta character
4264     lastCmd string
4265 }
4266 func (iin*IInput)Getc(timeoutUs int)(int){
4267     ch1 := EOF
4268     ch2 := EOF
4269     ch3 := EOF
4270     if( 0 < len(iin.pch) ){ // deQ
4271         ch1 = iin.pch[0]
4272         iin.pch = iin.pch[1:]
4273     }else{
4274         ch1 = fgetcTimeout(stdin,timeoutUs);
4275     }
4276     if( ch1 == 033 ){ // escape sequence
4277         ch2 = fgetcTimeout(stdin,EscTimeout);
4278         if( ch2 == EV_TIMEOUT ){
4279             }else{
4280                 ch3 = fgetcTimeout(stdin,EscTimeout);
4281                 if( ch3 == EV_TIMEOUT ){
4282                     iin.pch = append(iin.pch,ch2) // enQ
4283                 }else{
4284                     switch( ch2 ){
4285                         default:
4286                             iin.pch = append(iin.pch,ch2) // enQ
4287                             iin.pch = append(iin.pch,ch3) // enQ
4288                         case '!':
4289                             switch( ch3 ){
4290                                 case 'A': ch1 = GO_UP; // ^
4291                                 case 'B': ch1 = GO_DOWN; // v
4292                                 case 'C': ch1 = GO_RIGHT; // >
4293                                 case 'D': ch1 = GO_LEFT; // <
4294                                 case '3':
4295                                     ch4 := fgetcTimeout(stdin,EscTimeout);
4296                                     if( ch4 == '-' ){
4297                                         //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4298                                         ch1 = DEL_RIGHT
4299                                     }
4300                                 }
4301                         case '\\':
4302                             //ch4 := fgetcTimeout(stdin,EscTimeout);
4303                             //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4304                             switch( ch3 ){
4305                                 case '-': ch1 = DEL_RIGHT
4306                             }
4307                     }
4308                 }
4309             }
4310         }
4311     }
4312     return ch1
4313 }
4314 func (inn*IInput)clearline(){
4315     var i int
4316     fprintf(stderr,"\r");
4317     // should be ANSI ESC sequence
4318     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4319         fputc(' ',os.Stderr);
4320     }
4321     fprintf(stderr,"\r");
4322 }
4323 func (iin*IInput)Redraw(){
4324     redraw(iin,iin.lno,iin.line,iin.right)
4325 }
4326 func redraw(iin *IInput,lno int,line string,right string){
4327     inMeta := false
4328     showMode := ""
4329     showMeta := "" // visible Meta mode on the cursor position
4330     showLno := fmt.Sprintf("!&d! ",lno)
4331     InsertMark := "" // in visible insert mode
4332     if MODE_VicMode {
4333     }else{
4334     if 0 < len(iin.right) {
4335         InsertMark = " "
4336     }
4337     }
4338     if( 0 < len(iin.waitingMeta) ){
4339         inMeta = true
4340         if iin.waitingMeta[0] != 033 {
4341             showMeta = iin.waitingMeta
4342         }
4343     }
4344     if( romkanmode ){
4345         //romkanmark = " *";
4346     }else{
4347         //romkanmark = "";
4348     }
4349     if MODE_ShowMode {
4350         romkan := "___"
4351         inmeta := "-"
4352         inveri := ""
4353         if MODE_CapsLock {
4354             inmeta = "A"
4355         }
4356         if MODE_LowerLock {
4357             inmeta = "a"
4358         }
4359         if MODE_ViTrace {
4360             inveri = "v"
4361         }
4362         if MODE_VicMode {
4363             inveri = ":"
4364         }
4365     }
4366     if romkanmode {
4367         romkan = "\343\201\202"
4368         if MODE_CapsLock {
4369             inmeta = "R"
4370         }else{
4371             inmeta = "r"
4372         }
4373     }
4374     if inMeta {
4375         inmeta = "\\ "

```

```

4375     }
4376     showMode = "["+romkan+inmeta+inveri+"]";
4377 }
4378 Pre := "\r" + showMode + showLino
4379 Output := ""
4380 Left := ""
4381 Right := ""
4382 if romkanmode {
4383     Left = convs(line)
4384     Right = InsertMark+convs(right)
4385 }else{
4386     Left = line
4387     Right = InsertMark+right
4388 }
4389 Output = Pre+Left
4390 if MODE_ViTrace {
4391     Output += iin.LastCmd
4392 }
4393 Output += showMeta+Right
4394 for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4395     Output += " "
4396     // should be ANSI ESC sequence
4397     // not necessary just after newline
4398 }
4399 Output += Pre+Left+showMeta // to set the cursor to the current input position
4400 fprintf(stderr,"%s",Output)
4401
4402 if MODE_ViTrace {
4403     if 0 < len(iin.LastCmd) {
4404         iin.LastCmd = ""
4405         fprintf(stderr,"\r\n")
4406     }
4407 }
4408 }
4409 // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
4410 func delHeadChar(str string)(rline string,head string){
4411     clen := utf8.DecodeRune([]byte(str))
4412     head = string(str[0:clen])
4413     return str[clen:],head
4414 }
4415 func delTailChar(str string)(rline string, last string){
4416     var i = 0
4417     var clen = 0
4418     for {
4419         _,siz := utf8.DecodeRune([]byte(str)[i:])
4420         if siz <= 0 { break }
4421         clen = siz
4422         i += siz
4423     }
4424     last = str[len(str)-clen:]
4425     return str[0:len(str)-clen],last
4426 }
4427
4428 // 3> for output and history
4429 // 4> for keylog?
4430 // <a name="getline">Command Line Editor</a>
4431 func xgetline(lno int, prevline string, gsh*GshContext)(string){
4432     var iin IInput
4433     iin.lastlno = lno
4434     iin.lno = lno
4435
4436     CmdIndex = len(gsh.CommandHistory)
4437     if( !satty(0) == 0 ){
4438         if( sfgets(&iin.line,LINESIZE,stdin) == NULL ){
4439             iin.line = "exit\n";
4440         }else{
4441             return iin.line
4442         }
4443     }
4444     if( true ){
4445         //var pts string;
4446         //pts = ptsname(0);
4447         //pts = ttyname(0);
4448         //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
4449     }
4450     if( false ){
4451         fprintf(stderr,"! ");
4452         fflush(stderr);
4453         sfgets(&iin.line,LINESIZE,stdin);
4454         return iin.line
4455     }
4456     system("/bin/stty -echo -icanon");
4457     xline := iin.xgetline1(prevline,gsh)
4458     system("/bin/stty echo sane");
4459     return xline
4460 }
4461 func (iin*IInput)Translate(cmdch int){
4462     romkanmode = !romkanmode;
4463     if MODE_ViTrace {
4464         fprintf(stderr,"%v\r\n",string(cmdch));
4465     }else
4466     if( cmdch == 'J' ){
4467         fprintf(stderr,"J\r\n");
4468         iin.inJmode = true
4469     }
4470     iin.Redraw();
4471     loadDefaultDic(cmdch);
4472     iin.Redraw();
4473 }
4474 func (iin*IInput)Replace(cmdch int){
4475     iin.LastCmd = fmt.Sprintf("\%v",string(cmdch))
4476     iin.Redraw();
4477     loadDefaultDic(cmdch);
4478     dst := convs(iin.line+iin.right);
4479     iin.line = dst
4480     iin.right = ""
4481     if( cmdch == 'I' ){
4482         fprintf(stderr,"I\r\n");
4483         iin.inJmode = true
4484     }
4485     iin.Redraw();
4486 }
4487 // aa 12 alal
4488 func isAlpha(ch rune)(bool){
4489     if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z' {
4490         return true
4491     }
4492     return false
4493 }
4494 func isAlnum(ch rune)(bool){
4495     if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z' {
4496         return true
4497     }
4498     if '0' <= ch && ch <= '9' {
4499         return true

```

```

4500     }
4501     return false
4502 }
4503
4504 // 0.2.8 2020-0901 created
4505 // <a href="https://golang.org/pkg/unicode/utf8/#DecodeRuneInString">DecodeRuneInString</a>
4506 func (iin*Input)GotoTOPW(){
4507     str := iin.line
4508     i := len(str)
4509     if i <= 0 {
4510         return
4511     }
4512     //i0 := i
4513     i -= 1
4514     lastSize := 0
4515     var lastRune rune
4516     var found = -1
4517     for 0 < i { // skip preamble spaces
4518         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4519         if !isAlnum(lastRune) { // character, type, or string to be searched
4520             i -= lastSize
4521             continue
4522         }
4523         break
4524     }
4525     for 0 < i {
4526         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4527         if lastSize <= 0 { continue } // not the character top
4528         if !isAlnum(lastRune) { // character, type, or string to be searched
4529             found = i
4530             break
4531         }
4532         i -= lastSize
4533     }
4534     if found < 0 && i == 0 {
4535         found = 0
4536     }
4537     if 0 <= found {
4538         if isAlnum(lastRune) { // or non-kana character
4539             }else{ // when positioning to the top o the word
4540                 i += lastSize
4541             }
4542             iin.right = str[i:] + iin.right
4543             if 0 < i {
4544                 iin.line = str[0:i]
4545             }else{
4546                 iin.line = ""
4547             }
4548         }
4549         //fmt.Printf("\n(%d,%d,%d)[%s][%s]\n", i0, i, found, iin.line, iin.right)
4550         //fmt.Printf("") // set debug messae at the end of line
4551     }
4552 // 0.2.8 2020-0901 created
4553 func (iin*Input)GotoENDW(){
4554     str := iin.right
4555     if len(str) <= 0 {
4556         return
4557     }
4558     lastSize := 0
4559     var lastRune rune
4560     var lastW = 0
4561     i := 0
4562     inWord := false
4563
4564     lastRune, lastSize = utf8.DecodeRuneInString(str[0:])
4565     if isAlnum(lastRune) {
4566         r, z := utf8.DecodeRuneInString(str[lastSize:])
4567         if 0 < z && isAlnum(r) {
4568             inWord = true
4569         }
4570     }
4571     for i < len(str) {
4572         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4573         if lastSize <= 0 { break } // broken data?
4574         if !isAlnum(lastRune) { // character, type, or string to be searched
4575             break
4576         }
4577         lastW = i // the last alnum if in alnum word
4578         i += lastSize
4579     }
4580     if inWord {
4581         goto DISP
4582     }
4583     for i < len(str) {
4584         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4585         if lastSize <= 0 { break } // broken data?
4586         if isAlnum(lastRune) { // character, type, or string to be searched
4587             break
4588         }
4589         i += lastSize
4590     }
4591     for i < len(str) {
4592         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4593         if lastSize <= 0 { break } // broken data?
4594         if !isAlnum(lastRune) { // character, type, or string to be searched
4595             break
4596         }
4597         lastW = i
4598         i += lastSize
4599     }
4600 DISP:
4601     if 0 < lastW {
4602         iin.line = iin.line + str[0:lastW]
4603         iin.right = str[lastW:]
4604     }
4605     //fmt.Printf("\n(%d)[%s][%s]\n", i, iin.line, iin.right)
4606     //fmt.Printf("") // set debug messae at the end of line
4607 }
4608 // 0.2.8 2020-0901 created
4609 func (iin*Input)GotoNEXTW(){
4610     str := iin.right
4611     if len(str) <= 0 {
4612         return
4613     }
4614     lastSize := 0
4615     var lastRune rune
4616     var found = -1
4617     i := 1
4618     for i < len(str) {
4619         lastRune, lastSize = utf8.DecodeRuneInString(str[i:])
4620         if lastSize <= 0 { break } // broken data?
4621         if !isAlnum(lastRune) { // character, type, or string to be searched
4622             found = i
4623             break
4624         }

```

```

4625     i += lastSize
4626 }
4627 if 0 < found {
4628     if isAlnum(lastRune) { // or non-kana character
4629     }else{ // when positioning to the top o the word
4630         found += lastSize
4631     }
4632     iin.line = iin.line + str[0:found]
4633     if 0 < found {
4634         iin.right = str[found:]
4635     }else{
4636         iin.right = ""
4637     }
4638 }
4639 //fmt.Printf("\n(%d)[%s][%s]\n",i,iin.line,iin.right)
4640 //fmt.Printf("") // set debug messae at the end of line
4641 }
4642 // 0.2.8 2020-0902 created
4643 func (iin*Input)GotoPAIRCH(){
4644     str := iin.right
4645     if len(str) <= 0 {
4646         return
4647     }
4648     lastRune,lastSize := utf8.DecodeRuneInString(str[0:])
4649     if lastSize <= 0 {
4650         return
4651     }
4652     forw := false
4653     back := false
4654     pair := ""
4655     switch string(lastRune){
4656     case "{": pair = "}"; forw = true
4657     case "}": pair = "{"; back = true
4658     case "(": pair = ")"; forw = true
4659     case ")": pair = "("; back = true
4660     case "[": pair = "]"; forw = true
4661     case "]": pair = "["; back = true
4662     case "<": pair = ">"; forw = true
4663     case ">": pair = "<"; back = true
4664     case "\'": pair = "\'"; // context depednet, can be f" or back-double quote
4665     case "'": pair = "'"; // context depednet, can be f' or back-quote
4666     // case Japanese Kakkos
4667     }
4668     if forw {
4669         iin.SearchForward(pair)
4670     }
4671     if back {
4672         iin.SearchBackward(pair)
4673     }
4674 }
4675 // 0.2.8 2020-0902 created
4676 func (iin*Input)SearchForward(pat string)(bool){
4677     right := iin.right
4678     found := -1
4679     i := 0
4680     if strBegins(right,pat) {
4681         _z := utf8.DecodeRuneInString(right[i:])
4682         if 0 < z {
4683             i += z
4684         }
4685     }
4686     for i < len(right) {
4687         if strBegins(right[i:],pat) {
4688             found = i
4689             break
4690         }
4691         _z := utf8.DecodeRuneInString(right[i:])
4692         if z <= 0 { break }
4693         i += z
4694     }
4695     if 0 <= found {
4696         iin.line = iin.line + right[0:found]
4697         iin.right = iin.right[found:]
4698         return true
4699     }else{
4700         return false
4701     }
4702 }
4703 // 0.2.8 2020-0902 created
4704 func (iin*Input)SearchBackward(pat string)(bool){
4705     line := iin.line
4706     found := -1
4707     i := len(line)-1
4708     for i = i; 0 <= i; i-- {
4709         _z := utf8.DecodeRuneInString(line[i:])
4710         if z <= 0 {
4711             continue
4712         }
4713         //fprintf(stderr,"-- %v %v\n",pat,line[i:])
4714         if strBegins(line[i:],pat) {
4715             found = i
4716             break
4717         }
4718     }
4719     //fprintf(stderr,"--%d\n",found)
4720     if 0 <= found {
4721         iin.right = line[found:] + iin.right
4722         iin.line = line[0:found]
4723         return true
4724     }else{
4725         return false
4726     }
4727 }
4728 // 0.2.8 2020-0902 created
4729 // search from top, end, or current position
4730 func (gsh*GshContext)SearchHistory(pat string, forw bool)(bool,string){
4731     if forw {
4732         for _v := range gsh.CommandHistory {
4733             if 0 <= strings.Index(v.CmdLine,pat) {
4734                 //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4735                 return true,v.CmdLine
4736             }
4737         }
4738     }else{
4739         hlen := len(gsh.CommandHistory)
4740         for i := hlen-1; 0 < i; i-- {
4741             v := gsh.CommandHistory[i]
4742             if 0 <= strings.Index(v.CmdLine,pat) {
4743                 //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4744                 return true,v.CmdLine
4745             }
4746         }
4747     }
4748     //fprintf(stderr,"\n--De-- not-found(%v)\n",pat)
4749     return false,"(Not Found in History)"

```

```

4750 }
4751 // 0.2.8 2020-0902 created
4752 func (iin*IInput)GotoFORWSTR(pat string, gsh*GshContext){
4753     found := false
4754     if 0 < len(iin.right) {
4755         found = iin.SearchForward(pat)
4756     }
4757     if !found {
4758         found, line := gsh.SearchHistory(pat, true)
4759         if found {
4760             iin.line = line
4761             iin.right = ""
4762         }
4763     }
4764 }
4765 func (iin*IInput)GotoBACKSTR(pat string, gsh*GshContext){
4766     found := false
4767     if 0 < len(iin.line) {
4768         found = iin.SearchBackward(pat)
4769     }
4770     if !found {
4771         found, line := gsh.SearchHistory(pat, false)
4772         if found {
4773             iin.line = line
4774             iin.right = ""
4775         }
4776     }
4777 }
4778 func (iin*IInput)getString1(prompt string)(string){ // should be editable
4779     iin.clearline();
4780     fprintf(stderr, "\r\v", prompt)
4781     str := ""
4782     for {
4783         ch := iin.Getc(10*1000*1000)
4784         if ch == '\n' || ch == '\r' {
4785             break
4786         }
4787         sch := string(ch)
4788         str += sch
4789         fprintf(stderr, "%s", sch)
4790     }
4791     return str
4792 }
4793
4794 // search pattern must be an array and selectable with ^N/^P
4795 var SearchPat = ""
4796 var SearchForw = true
4797
4798 func (iin*IInput)xgetline1(prevline string, gsh*GshContext)(string){
4799     var ch int;
4800
4801     MODE_ShowMode = false
4802     MODE_VicMode = false
4803     iin.Redraw();
4804     first := true
4805
4806     for cix := 0; ; cix++ {
4807         iin.pinJmode = iin.inJmode
4808         iin.inJmode = false
4809
4810         ch = iin.Getc(1000*1000)
4811
4812         if ch != EV_TIMEOUT && first {
4813             first = false
4814             mode := 0
4815             if romkanmode {
4816                 mode = 1
4817             }
4818             now := time.Now()
4819             Events = append(Events, Event{now, EV_MODE, int64(mode), CmdIndex})
4820         }
4821         if ch == 033 {
4822             MODE_ShowMode = true
4823             MODE_VicMode = !MODE_VicMode
4824             iin.Redraw();
4825             continue
4826         }
4827         if MODE_VicMode {
4828             switch ch {
4829                 case '0': ch = GO_TOPL
4830                 case '$': ch = GO_ENDL
4831                 case 'b': ch = GO_TOPW
4832                 case 'e': ch = GO_ENDW
4833                 case 'w': ch = GO_NEXTW
4834                 case '$': ch = GO_PAIRCH
4835
4836                 case 'j': ch = GO_DOWN
4837                 case 'k': ch = GO_UP
4838                 case 'h': ch = GO_LEFT
4839                 case 'l': ch = GO_RIGHT
4840                 case 'x': ch = DEL_RIGHT
4841                 case 'a': MODE_VicMode = !MODE_VicMode
4842                     ch = GO_RIGHT
4843                 case 'i': MODE_VicMode = !MODE_VicMode
4844                     iin.Redraw();
4845                     continue
4846                 case '-':
4847                     right, head := delHeadChar(iin.right)
4848                     if len([]byte(head)) == 1 {
4849                         ch = int(head[0])
4850                         if( 'a' <= ch && ch <= 'z' ){
4851                             ch = ch + 'A'-'a'
4852                         }else
4853                         if( 'A' <= ch && ch <= 'Z' ){
4854                             ch = ch + 'a'-'A'
4855                         }
4856                         iin.right = string(ch) + right
4857                     }
4858                     iin.Redraw();
4859                     continue
4860                 case 'f': // GO_FORWCH
4861                     iin.Redraw();
4862                     ch = iin.Getc(3*1000*1000)
4863                     if ch == EV_TIMEOUT {
4864                         iin.Redraw();
4865                         continue
4866                     }
4867                     SearchPat = string(ch)
4868                     SearchForw = true
4869                     iin.GotoFORWSTR(SearchPat, gsh)
4870                     iin.Redraw();
4871                     continue
4872                 case '/':
4873                     SearchPat = iin.getString1("/") // should be editable
4874                     SearchForw = true

```

```

4875         iin.GotoFORWSTR(SearchPat,gsh)
4876         iin.Redraw();
4877         continue
4878     case '?':
4879         SearchPat = iin.getstringl("?") // should be editable
4880         SearchForw = false
4881         iin.GotoBACKSTR(SearchPat,gsh)
4882         iin.Redraw();
4883         continue
4884     case 'n':
4885         if SearchForw {
4886             iin.GotoFORWSTR(SearchPat,gsh)
4887         }else{
4888             iin.GotoBACKSTR(SearchPat,gsh)
4889         }
4890         iin.Redraw();
4891         continue
4892     case 'N':
4893         if !SearchForw {
4894             iin.GotoFORWSTR(SearchPat,gsh)
4895         }else{
4896             iin.GotoBACKSTR(SearchPat,gsh)
4897         }
4898         iin.Redraw();
4899         continue
4900     }
4901 }
4902 switch ch {
4903     case GO_TOPW:
4904         iin.GotoTOPW()
4905         iin.Redraw();
4906         continue
4907     case GO_ENDW:
4908         iin.GotoENDW()
4909         iin.Redraw();
4910         continue
4911     case GO_NEXTW:
4912         // To next space then
4913         iin.GotoNEXTW()
4914         iin.Redraw();
4915         continue
4916     case GO_PAIRCH:
4917         iin.GotoPAIRCH()
4918         iin.Redraw();
4919         continue
4920 }
4921
4922 //fprintf(stderr,"A{%02X}\n",ch);
4923 if( ch == '\\' || ch == 033 ){
4924     MODE_ShowMode = true
4925     metach := ch
4926     iin.waitingMeta = string(ch)
4927     iin.Redraw();
4928     // set cursor //fprintf(stderr,"???\b\b\b")
4929     ch = fgetcTimeout(stdin,2000*1000)
4930     // reset cursor
4931     iin.waitingMeta = ""
4932
4933     cmdch := ch
4934     if( ch == EV_TIMEOUT ){
4935         if metach == 033 {
4936             continue
4937         }
4938         ch = metach
4939     }else
4940     /*
4941     if( ch == 'm' || ch == 'M' ){
4942         mch := fgetcTimeout(stdin,1000*1000)
4943         if mch == 'r' {
4944             romkanmode = true
4945         }else{
4946             romkanmode = false
4947         }
4948         continue
4949     }else
4950     */
4951     if( ch == 'k' || ch == 'K' ){
4952         MODE_Recursive = !MODE_Recursive
4953         iin.Translate(cmdch);
4954         continue
4955     }else
4956     if( ch == 'j' || ch == 'J' ){
4957         iin.Translate(cmdch);
4958         continue
4959     }else
4960     if( ch == 'i' || ch == 'I' ){
4961         iin.Replace(cmdch);
4962         continue
4963     }else
4964     if( ch == 'l' || ch == 'L' ){
4965         MODE_LowerLock = !MODE_LowerLock
4966         MODE_CapsLock = false
4967         if MODE_ViTrace {
4968             fprintf(stderr,"%v\r\n",string(cmdch));
4969         }
4970         iin.Redraw();
4971         continue
4972     }else
4973     if( ch == 'u' || ch == 'U' ){
4974         MODE_CapsLock = !MODE_CapsLock
4975         MODE_LowerLock = false
4976         if MODE_ViTrace {
4977             fprintf(stderr,"%v\r\n",string(cmdch));
4978         }
4979         iin.Redraw();
4980         continue
4981     }else
4982     if( ch == 'v' || ch == 'V' ){
4983         MODE_ViTrace = !MODE_ViTrace
4984         if MODE_ViTrace {
4985             fprintf(stderr,"%v\r\n",string(cmdch));
4986         }
4987         iin.Redraw();
4988         continue
4989     }else
4990     if( ch == 'c' || ch == 'C' ){
4991         if 0 < len(iin.line) {
4992             xline,tail := delTailChar(iin.line)
4993             if len([]byte(tail)) == 1 {
4994                 ch = int(tail[0])
4995                 if( 'a' <= ch && ch <= 'z' ){
4996                     ch = ch + 'A'-'a'
4997                 }else
4998                 if( 'A' <= ch && ch <= 'Z' ){
4999                     ch = ch + 'a'-'A'

```



```

5000         }
5001         iin.line = xline + string(ch)
5002     }
5003     }
5004     if MODE_ViTrace {
5005         fprintf(stderr, "%v\r\n", string(cmdch));
5006     }
5007     iin.Redraw();
5008     continue
5009 }else{
5010     iin.pch = append(iin.pch,ch) // push
5011     ch = '\\'
5012 }
5013 }
5014 switch( ch ){
5015 case 'P'-0x40: ch = GO_UP
5016 case 'N'-0x40: ch = GO_DOWN
5017 case 'B'-0x40: ch = GO_LEFT
5018 case 'F'-0x40: ch = GO_RIGHT
5019 }
5020 //fprintf(stderr, "B[%02X]\n", ch);
5021 switch( ch ){
5022 case 0:
5023     continue;
5024
5025 case '\t':
5026     iin.Replace('j');
5027     continue
5028 case 'X'-0x40:
5029     iin.Replace('j');
5030     continue
5031
5032 case EV_TIMEOUT:
5033     iin.Redraw();
5034     if iin.pinJmode {
5035         fprintf(stderr, "\\J\r\n")
5036         iin.inJmode = true
5037     }
5038     continue
5039 case GO_UP:
5040     if iin.lno == 1 {
5041         continue
5042     }
5043     cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
5044     if ok {
5045         iin.line = cmd
5046         iin.right = ""
5047         iin.lno = iin.lno - 1
5048     }
5049     iin.Redraw();
5050     continue
5051 case GO_DOWN:
5052     cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
5053     if ok {
5054         iin.line = cmd
5055         iin.right = ""
5056         iin.lno = iin.lno + 1
5057     }else{
5058         iin.line = ""
5059         iin.right = ""
5060         if iin.lno == iin.lastlno-1 {
5061             iin.lno = iin.lno + 1
5062         }
5063     }
5064     iin.Redraw();
5065     continue
5066 case GO_LEFT:
5067     if 0 < len(iin.line) {
5068         xline,tail := delTailChar(iin.line)
5069         iin.line = xline
5070         iin.right = tail + iin.right
5071     }
5072     iin.Redraw();
5073     continue;
5074 case GO_RIGHT:
5075     if( 0 < len(iin.right) && iin.right[0] != 0 ){
5076         xright,head := delHeadChar(iin.right)
5077         iin.right = xright
5078         iin.line += head
5079     }
5080     iin.Redraw();
5081     continue;
5082 case EOF:
5083     goto EXIT;
5084 case 'R'-0x40: // replace
5085     dst := convs(iin.line+iin.right);
5086     iin.line = dst
5087     iin.right = ""
5088     iin.Redraw();
5089     continue;
5090 case 'T'-0x40: // just show the result
5091     readDic();
5092     romkanmode = !romkanmode;
5093     iin.Redraw();
5094     continue;
5095 case 'L'-0x40:
5096     iin.Redraw();
5097     continue
5098 case 'K'-0x40:
5099     iin.right = ""
5100     iin.Redraw();
5101     continue
5102 case 'E'-0x40:
5103     iin.line += iin.right
5104     iin.right = ""
5105     iin.Redraw();
5106     continue
5107 case 'A'-0x40:
5108     iin.right = iin.line + iin.right
5109     iin.line = ""
5110     iin.Redraw();
5111     continue
5112 case 'U'-0x40:
5113     iin.line = ""
5114     iin.right = ""
5115     iin.clearline();
5116     iin.Redraw();
5117     continue;
5118 case DEL_RIGHT:
5119     if( 0 < len(iin.right) ){
5120         iin.right,_ = delHeadChar(iin.right)
5121         iin.Redraw();
5122     }
5123     continue;
5124 case 0x7F: // BS? not DEL

```

```

5125         if( 0 < len(iin.line) ){
5126             iin.line,_ = delTailChar(iin.line)
5127             iin.Redraw();
5128         }
5129         /*
5130         else
5131         if( 0 < len(iin.right) ){
5132             iin.right,_ = delHeadChar(iin.right)
5133             iin.Redraw();
5134         }
5135         */
5136         continue;
5137     case 'H'-0x40:
5138         if( 0 < len(iin.line) ){
5139             iin.line,_ = delTailChar(iin.line)
5140             iin.Redraw();
5141         }
5142         continue;
5143     }
5144     if( ch == '\n' || ch == '\r' ){
5145         iin.line += iin.right;
5146         iin.right = ""
5147         iin.Redraw();
5148         fputc(ch,stderr);
5149         break;
5150     }
5151     if MODE_CapsLock {
5152         if "a" <= ch && ch <= 'z' {
5153             ch = ch+'A'-'a'
5154         }
5155     }
5156     if MODE_LowerLock {
5157         if "A" <= ch && ch <= 'Z' {
5158             ch = ch+'a'-'A'
5159         }
5160     }
5161     iin.line += string(ch);
5162     iin.Redraw();
5163 }
5164 EXIT:
5165     return iin.line + iin.right;
5166 }
5167
5168 func getline_main(){
5169     line := xgetline(0,"",nil)
5170     fprintf(stderr,"%s\n",line);
5171     /*
5172     dp = strpbrk(line,"\r\n");
5173     if( dp != NULL ){
5174         *dp = 0;
5175     }
5176
5177     if( 0 ){
5178         fprintf(stderr,"\n(%d)\n",int(strlen(line)));
5179     }
5180     if( lseek(3,0,0) == 0 ){
5181         if( romkanmode ){
5182             var buf [8*1024]byte;
5183             convs(line,buf);
5184             strcpy(line,buf);
5185         }
5186         write(3,line,strlen(line));
5187         ftruncate(3,lseek(3,0,SEEK_CUR));
5188         //fprintf(stderr,"outsize=%d\n",int)lseek(3,0,SEEK_END));
5189         lseek(3,0,SEEK_SET);
5190         close(3);
5191     }else{
5192         fprintf(stderr,"\r\n gotline: ");
5193         trans(line);
5194         //printf("%s\n",line);
5195         printf("\n");
5196     }
5197     */
5198 }
5199 //== end ===== getline
5200
5201 //
5202 // $USERHOME/.gsh/
5203 // gsh-rc.txt, or gsh-configure.txt
5204 // gsh-history.txt
5205 // gsh-aliases.txt // should be conditional?
5206 //
5207 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
5208     homedir,found := userHomeDir()
5209     if !found {
5210         fmt.Printf("--E-- You have no UserHomeDir\n")
5211         return true
5212     }
5213     gshhome := homedir + "/" + GSH_HOME
5214     _, err2 := os.Stat(gshhome)
5215     if err2 != nil {
5216         err3 := os.Mkdir(gshhome,0700)
5217         if err3 != nil {
5218             fmt.Printf("--E-- Could not Create %s (%s)\n",
5219                 gshhome,err3)
5220             return true
5221         }
5222         fmt.Printf("--I-- Created %s\n",gshhome)
5223     }
5224     gshCtx.GshHomeDir = gshhome
5225     return false
5226 }
5227 func setupGshContext()(GshContext,bool){
5228     gshPA := syscall.ProcAttr {
5229         "", // the staring directory
5230         os.Environ(), // environ[]
5231         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
5232         nil, // OS specific
5233     }
5234     cwd,_ := os.Getwd()
5235     gshCtx := GshContext {
5236         cwd, // StartDir
5237         "", // GetLine
5238         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
5239         gshPA,
5240         []GCommandHistory{}, //something for invokation?
5241         GCommandHistory{}, // CmdCurrent
5242         false,
5243         []int{},
5244         syscall.Rusage{},
5245         "", // GshHomeDir
5246         Ttyid(),
5247         false,
5248         false,
5249         []PluginInfo{},

```

```

5250     []string{},
5251     " ",
5252     "v",
5253     ValueStack{},
5254     GServer{"", ""}, // LastServer
5255     " ", // RSERVER
5256     cwd, // RWD
5257     CheckSum{},
5258 }
5259 err := gshCtx.gshSetupHomedir()
5260 return gshCtx, err
5261 }
5262 func (gsh*GshContext)gshelllh(gline string)(bool){
5263     ghist := gsh.CmdCurrent
5264     ghist.WorkDir_ = os.Getwd()
5265     ghist.WorkDir_ = len(gsh.ChdirHistory)-1
5266     //fmt.Printf("--D--ChdirHistory(%#d)\n", len(gsh.ChdirHistory))
5267     ghist.StartAt = time.Now()
5268     rusagev1 := Getrusagev()
5269     gsh.CmdCurrent.FoundFile = []string{}
5270     fin := gsh.tgshelllh(gline)
5271     rusagev2 := Getrusagev()
5272     ghist.Rusagev = RusageSubv(rusagev2, rusagev1)
5273     ghist.EndAt = time.Now()
5274     ghist.CmdLine = gline
5275     ghist.FoundFile = gsh.CmdCurrent.FoundFile
5276
5277     /* record it but not show in list by default
5278     if len(gline) == 0 {
5279         continue
5280     }
5281     if gline == "hi" || gline == "history" { // don't record it
5282         continue
5283     }
5284     */
5285     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
5286     return fin
5287 }
5288 // <a name="main">Main loop</a>
5289 func script(gshCtxGiven *GshContext) (_ GshContext) {
5290     gshCtxBuf,err0 := setupGshContext()
5291     if err0 {
5292         return gshCtxBuf;
5293     }
5294     gshCtx := &gshCtxBuf
5295
5296     //fmt.Printf("--I-- GSH_HOME=%s\n", gshCtx.GshHomeDir)
5297     //resmap()
5298
5299     /*
5300     if false {
5301         gsh_getlinev, with_exgetline :=
5302             which("PATH", []string{"which", "gsh-getline", "-s"})
5303         if with_exgetline {
5304             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
5305             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
5306         }else{
5307             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
5308         }
5309     }
5310     */
5311
5312     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
5313     gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist0)
5314
5315     prevline := ""
5316     skipping := false
5317     for hix := len(gshCtx.CommandHistory); ; {
5318         gline := gshCtx.getline(hix, skipping, prevline)
5319         if skipping {
5320             if strings.Index(gline, "fi") == 0 {
5321                 fmt.Printf("fi\n");
5322                 skipping = false;
5323             }else{
5324                 //fmt.Printf("%s\n", gline);
5325             }
5326             continue
5327         }
5328         if strings.Index(gline, "if") == 0 {
5329             //fmt.Printf("--D-- if start: %s\n", gline);
5330             skipping = true;
5331             continue
5332         }
5333         if false {
5334             os.Stdout.Write([]byte("gotline:"))
5335             os.Stdout.Write([]byte(gline))
5336             os.Stdout.Write([]byte("\n"))
5337         }
5338         gline = strsubst(gshCtx, gline, true)
5339         if false {
5340             fmt.Printf("fmt.Printf %%v - %v\n", gline)
5341             fmt.Printf("fmt.Printf %%s - %s\n", gline)
5342             fmt.Printf("fmt.Printf %%x - %x\n", gline)
5343             fmt.Printf("fmt.Printf %%U - %s\n", gline)
5344             fmt.Printf("Stoutt.Write -")
5345             os.Stdout.Write([]byte(gline))
5346             fmt.Printf("\n")
5347         }
5348         /*
5349         // should be cared in substitution ?
5350         if 0 < len(gline) && gline[0] == '!' {
5351             xgline, set, err := searchHistory(gshCtx, gline)
5352             if err {
5353                 continue
5354             }
5355             if set {
5356                 // set the line in command line editor
5357             }
5358             gline = xgline
5359         }
5360         */
5361         fin := gshCtx.gshelllh(gline)
5362         if fin {
5363             break;
5364         }
5365         prevline = gline;
5366         hix++;
5367     }
5368     return *gshCtx
5369 }
5370 func main() {
5371     gshCtxBuf := GshContext{}
5372     gsh := &gshCtxBuf
5373     argv := os.Args
5374     if 1 < len(argv) {

```

```

5375     if isin("version",argv){
5376         gsh.showVersion(argv)
5377         return
5378     }
5379     comx := isinX("-c",argv)
5380     if 0 < comx {
5381         gshCtxBuf,err := setupGshContext()
5382         gsh := gshCtxBuf
5383         if !err {
5384             gsh.gshellv(argv[comx+1:])
5385         }
5386         return
5387     }
5388 }
5389 if 1 < len(argv) && isin("-s",argv) {
5390 }else{
5391     gsh.showVersion(append(argv,{jstring{"-l","-a"...}))
5392 }
5393 script(nil)
5394 //gshCtx := script(nil)
5395 //gshell(gshCtx,"time")
5396 }
5397 </div></details>
5398 <details id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
5399 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
5400 // - merged histories of multiple parallel gsh sessions
5401 // - alias as a function or macro
5402 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
5403 // - retrieval PATH of files by its type
5404 // - gsh as an IME with completion using history and file names as dictionaies
5405 // - gsh a scheduler in precise time of within a millisecond
5406 // - all commands have its subucmand after "---" symbol
5407 // - filename expansion by "-find" command
5408 // - history of ext code and output of each commoand
5409 // - "script" output for each command by pty-tee or telnet-tee
5410 // - $BULLTIN command in PATH to show the priority
5411 // - "?" symbol in the command (not as in arguments) shows help request
5412 // - searching command with wild card like: which ssh-
5413 // - longformat prompt after long idle time (should dismiss by BS)
5414 // - customizing by building plugin and dynamically linking it
5415 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
5416 // - "!" symbol should be used for negation, don't wast it just for job control
5417 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
5418 // - making canonical form of command at the start adding quotation or white spaces
5419 // - name(a,b,c)... use "(" and ")" to show both delimiter and realm
5420 // - name? or name! might be useful
5421 // - htar format - packing directory contents into a single html file using data scheme
5422 // - filepath substitution shold be done by each command, especially in case of builtins
5423 // - @N substitution for the history of working directory, and @spec for more generic ones
5424 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
5425 // - GSH_PATH for plugins
5426 // - standard command output: list of data with name, size, resouce usage, modified time
5427 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
5428 // -wc word-count, grep match line count, ...
5429 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
5430 // - -tailf-filename like tail -f filename, repeat close and open before read
5431 // - max. size and max. duration and timeout of (generated) data transfer
5432 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
5433 // - IME "?" at the top of the command line means searching history
5434 // - IME %d/0x10000/ %x/ffff/
5435 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
5436 // - gsh in WebAssembly
5437 // - gsh as a HTTP server of online-manual
5438 //---END--- (-~/ITS more</div></details>
5439
5440 <<span class="gsh-golang-data">
5441
5442 var WorldDic = <<span id="gsh-world-dic">
5443 "data:text/dic;base64, "+
5444 "Ly8gTXJlTUUVwC4wJjEg6L6e5pu4ICgyMDiWLT4MThlKQpzZWthaSDkuJbnlYwKa28g44GT"+
5445 "Cm5lI0OCkwpaSDjgasKY2hpI00BoQp0aSDjgaEkaEGe44GvCnNlI00BmwprYSdjYkSaSDj"+
5446 "gYOK";
5447 </span>
5448
5449 var WnnDic = <<span id="gsh-wnn-dic">
5450 "data:text/dic;base64, "+
5451 "Pg1ldEGy2hhcnl1dD0iVVrGLTgiPgo8dGV4dGFyZWEgY29scz04MCBYb3dzPTQwPgovL2Rr"+
5452 "Y3ZlcglHU2hlbGxc01NRVxzZG1jdGlVbmFyeVxzZm9yXHNtbn5ccy8vXHMxMDiWLT4MzAK"+
5453 "RlNoZnVhZGZlY2h1bGx0Cm5lI00BmwprYSdjYkSaSDjgaEkaEGe44GvCnNlI00BmwprYSdjYkSaSDj"+
5454 "44G44GICewQjEwNjQpuY1hZQn1K13liY0K44GQ44GL44GuCes4remHjpuY1hbh8J5LiI"+
5455 "6YeOCndhCeOCjvp0YQnjgZ8Kc2kJ44GXcnNoaQnjgZcK8m8J44GuCm5hCeOBqgptYQnjg4K"+
5456 "ZQnjgYgKaGEJ44GvCm5hCeOBqgptYQnjgYk8m8J44GuCmRlCeOBpwpzQnjgZkZVxzCWVj"+
5457 "aG8ZKZgljCWmpRwplY2hvcWVjaG8KcmVwGF5X3c1JGxheQpyXZBlYXQjcmVwZWF0cmR0WRh"+
5458 "dGvcyysnJvKlSVkL5VlOIvNO1VTJvp0aW9uCXRPb24KJXQJlY8dG8gYmUgYW4gYWN0"+
5459 "aW9uYjVwdGV4dGFyZWE+Cg=="
5460 </span>
5461
5462 var SumomoDic = <<span id="gsh-sumomo-dic">
5463 "data:text/dic;base64, "+
5464 "Pg1ldEGy2hhcnl1dD0iVVrGLTgiPgo8dGV4dGFyZWEgY29scz04MCBYb3dzPTQwPgovL3Z1"+
5465 "cg1HU2hlbGxc01NRVxzZG1jdGlVbmFyeVxzZm9yXHNtdWlvbW9ccy8vXHMxMDiWLT4MzAK"+
5466 "c3UJ44GZCm1vCeOCggpubwnjga4KdQnJgYYkY2hpcEOBQp0aQnjgaEKdWnoaQn1hUkdXRP"+
5467 "CeWghOpzd1vW8J44GZ44KC44KCCnNlBw9t821vCeOBmeOCguOCguOCggptb21vCeahgwpt"+
5468 "b21vbW8J5gQd44KCCiwsCeOAgqouLgnjgiIKPC90ZXh0YXJlYjY4K"
5469 </span>
5470
5471 var SijimiDic = <<span id="gsh-sijimi-dic">
5472 "data:text/dic;base64, "+
5473 "Pg1ldEGy2hhcnl1dD0iVVrGLTgiPgo8dGV4dGFyZWEgY29scz04MCBYb3dzPTQwPgovL3Z1"+
5474 "cg1HU2hlbGxc01NRVxzZG1jdGlVbmFyeVxzZm9yXHNtdWlvbW9ccy8vXHMxMDiWLT4MzAK"+
5475 "CnNpCeOB1wPzAGJ44GXCMppCeOBmAptAQnjg8KbmEJ44GqCmpIeOBmOOCqp4eXUJ44KF"+
5476 "CNuJ44GCGM5pCeOBqwrbnwjZMKamTq2psCeOB1Opqamtqa2WJ44GXCMpamtqbAnjgkKa2pqamts"+
5477 "gaEka2EJ44GLCNJhCeOCiQosLAnjgIEKl4J44CCnHUY5hCeS4gwp4anVlCeWNgOp4bmkJ"+
5478 "5LpCmtevAnlIaK29xvCeWaiwprB3gY5YCLCm5hbmFgdXVuaXgJNzIKbmFuYwpldW5peHJg"+
5479 "77yx77yScM5hbmFgdXVuaVg77yx77yScU4gWnges6JhJgJNzIKa29dW5UeWai+Wihgp0"+
5480 "aWtcmEXCeOBoeOBi+OCiQp0aWtcmEJ5YqbcCmNoaWtcmEJ5YqbcjVwdGV4dGFyZWE+Cg=="
5481 </span>
5482
5483 var JA_JKLDic = <<span id="gsh-ja-jkl-dic">
5484 "data:text/dic;base64, "+
5485 "Ly92Z2JscU15SU1FamRyY2ptb3JzZWpKQWpKS0w0mJyAYMGowODE5KSsheLV4pL1NhdG94SVRT"+
5486 "CmtqamprBGTqa2tsa2psIOS41ueVjApqamtqamwJ44GCMtqbAnjgYKqa2tqbAnjgYKamtq"+
5487 "amwJ44GICmtq2trBAnjgYKa2pra2wJ44GLCmpramtrBAnjgY0Ka2tramwJ44GCMpramps"+
5488 "CeOBKOpqampqbAnjgZMKamtqa2psCeOB1Opqamtqa2WJ44GXCMpamtqbAnjgkKa2pqamts"+
5489 "CeOBmwPqamprBAnjgZ0KamtscCeOBnwpra2prBAnjgaEka2pqa2wJ44GCMtqa2pqBAnjgaYK"+
5490 "a2tqa2tsCeOBqApramtscCeOBqppq2prBAnjgKa2tra2wJ44GCMpqa2psCeOBpOpra2p"+
5491 "BAnjga4Kamttra2wJ44GvCmpqa2tqbAnjgIKampra2WJ44GICmtscCeOBAppqa2tsCeOBuwpq"+
5492 "a2tqbAnjgkKa2tqa2psCeOBvwpqbAnjgOKamtra2psCeOCqppqa2tqa2WJ44KCCmjqamwJ"+
5493 "44KECmpra2pqbAnjgYKampsCeOCiApra2tsCeOCiQpamtscCeOCiQpqa2pqa2WJ44KLCmpq"+
5494 "amwJ44KMCmtqa2psCeOCjOqpa2psCeOCjwpramramwJ44KQCmtqamtrBAnjgpeka2pqamwJ"+
5495 "44KSCmtqa2prBAnjgpmKa2pqa2psCeODvApra2wJ44KCMramprBAnjgppKa2pramtqbAnj"+
5496 "gIEK";
5497 </span>
5498
5499 </span>

```

```

5500 /*
5501 <details id="references"><summary>References</summary><div class="gsh-src">
5502 <p>
5503 <a href="https://golang.org">The Go Programming Language</a>
5504 <iframe src="https://golang.org" width="100%" height="300"></iframe>
5505
5506 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
5507 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
5508 CSS:
5509 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors">Selectors</a>
5510 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
5511 HTTP
5512 JavaScript:
5513 ..
5514 </p>
5515 </div></details>
5516 */
5517 /*
5518 <details id="html-src" onclick="frame_open();"><summary>Raw Source</summary><div>
5519
5520 <!-- h2>The full of this HTML including the Go code is here.</h2 -->
5521 <details id="gsh-whole-view"><summary>Whole file</summary>
5522 <a name="whole-src-view"></a>
5523 <span id="src-frame"></span><!-- a window to show source code -->
5524 </details>
5525
5526 <details id="gsh-style-frame" onclick="fill_CSSView()"><summary>CSS part</summary>
5527 <a name="style-src-view"></a>
5528 <span id="gsh-style-view"></span>
5529 </details>
5530
5531 <details id="gsh-script-frame" onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
5532 <a name="script-src-view"></a>
5533 <span id="gsh-script-view"></span>
5534 </details>
5535
5536 <details id="gsh-data-frame" onclick="fill_DataView()"><summary>Builtin data part</summary>
5537 <a name="gsh-data-frame"></a>
5538 <span id="gsh-data-view"></span>
5539 </details>
5540
5541 </div></details>
5542 */
5543 /*
5544 <div id="gsh-footer" style=""></div><!-- ----- END-OF-VISIBLE-PART ----- -->
5545
5546
5547 <style id="gsh-style-def">
5548 //body {display:none;}
5549 .gsh-link{color:green;}
5550 #gsh {border-width:1px;margin:0;padding:0;}
5551 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
5552 #gsh header{height:100px;}
5553 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
5554 #gsh-menu{font-size:14pt;color:#f88;}
5555 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
5556 #gsh note{color:#000;font-size:10pt;}
5557 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
5558 #gsh h3{color:#24a;font-family:Georgia;font-size:16pt;}
5559 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
5560 #gsh summary{font-size:16pt;color:#fff;background-color:#8af,height:30px;}
5561 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
5562 #gsh a{color:#24a;}
5563 #gsh a[name]{color:#24a;font-size:16pt;}
5564 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5565 #gsh .gsh-src{background-color:#faffff;color:#223;}
5566 #gsh-src-src{spellcheck:false}
5567 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5568 #src-frame-textarea{background-color:#faffff;color:#223;}
5569 .gsh-code {white-space:pre;font-family:monospace !import;}
5570 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
5571 .gsh-golang-data {display:none;}
5572 #gsh-winId {color:#000;font-size:14pt;}
5573
5574 .gsh-document {font-size:11pt;background-color:#fff;font-family:Georgia;}
5575 .gsh-document {color:#000;background-color:#fff !import;}
5576 .gsh-document > h2{color:#000;background-color:#fff !import;}
5577 .gsh-document details{color:#000;background-color:#fff;font-family:Georgia;}
5578 .gsh-document p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
5579 .gsh-document address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
5580
5581 @media print {
5582 #gsh pre{font-size:11pt !import;}
5583 }
5584 </style>
5585
5586 <!--
5587 // Logo image should be drawn by JavaScript from a meta-font.
5588 // CSS seems not follow line-splitted URL
5589 -->
5590 <script id="gsh-data">
5591 //GshLogo="QR-ITS-more.jp.png"
5592 GshLogo="data:image/png;base64,\
5593 iVBORw0KGooAAANSUHEuGAAQAEEAAB/CAYAADvs3f4AAAAAXNSR0IARs4c6QAAAHlWE1m\
5594 TU0AKgAAAAGABAEaAAUAAAABAAAPGEBAAUAAAABAAARgEoAAMAAAABAAIAAIpAAQAAAAAB\
5595 AAAATgAAAAAABAAIAAAQAABAAEgAAABAAQAAQAAQAAQAAQAAQAAQAAQAAQAAQAAQAAQAAQAA\
5596 AAAAQAABH8AAAAAYx1BhgAAAA1WSF1zAAALEWAAACMBAJqcGAAAF3RJRFEUeAhtnQUFNWZ\
5597 x+xt7ukZ3iCggO/jY6Gosb8WgmZAvn7uG4+biSTR7YnXQDQPCKgJ2aNwLd2MSLRkeUAPnoCdu\
5598 4iuJx7jriYz50D0GMf2VqIBEiSggCoIMMA+mu+vu//ZMD9U1da6a2aUbv91CKRq3vvd6/q\
5599 fnXvd8tBA88IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5600 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5601 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5602 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5603 2eXs9H+ftksdHxsic2qqdE7YusS+1qaalKfnY5YsokMhWEPTdk4MQFz5UEExLbLYsAYU15\
5604 npdILKXEZClPirM53JSUaq9SocqU6i+2kK3StuONy5reEKGJ7Qw7mOvKec2Toq0iZw0ljfS\
5605 jboVHCstMRb3USXEJ8hFu7dsdFm2+u4vWVWFvXbBMeZULAE/hcKogab6eKGoL1Nkh56PC\
5606 Hx2VVBKORqKq3qUeKi1YdaOFONJ56OkdI6w5BwommoQlyPziON9DLMXpFK/60p2P/PiyovF\
5607 N8mfM+nlWNGnJ9kQToLVGSF+2p2Ri1lgn3iJ0vK7YsowWmzEuVPFIRkydfOak2LRS80q\
5608 zrWocCOG6EhvgRacj/dktj3g7dXXH4gKN6ARS0zPzYerqS6RAoZDQqfk79SKTRXHu/e+9FN\
5609 L66ae88pU/PN1PNITLJKS373dPXSR20ur7iIwPcC8QhbnNcyHUIrlyyOTQVYF5fvqBL7jx\
5610 +cNhjBj5gRyDlJHy39o84D40H2QtX8THaPeFUIOU+wLc+KnyhK5FEVOWGAEhB8xeMOLX\
5611 rikb9gHEP52VgQ14h89FUA6kYJfbbQbnzLJg4zFiesndHCwvUoeiVQob/5C9FY9DlUeOH\
5612 +zGthU9NsGQgrm0WgurkI9RpjBD4Y6uQcQd5TU0W63zD3MHesy14V49isbKyxh8iCpFR\
5613 UJ6toACF7F9Vf58NBFDHTOMBae74Ent+eWrRw+lZ/QTw60AdB7QUJps/OA7COoNBNCMEUz\
5614 ttCu/coG28f1pVKElTPFV8juRasEahbHvxaRlguoeBPyfudo4+0feBdybL8Lz9XeSXFAMoc\
5615 bgGovog1zgG6w4JF392xnHhdc+Mwf3JTjntz2yCIYJBJXNUT5KIKyck1sxRdl6BmcevN\
5616 aJovy/VBacMevgEP46/ZlnJt9jx17VL53Z15Mtvap1QGLNhw5pQDqYNTQ1Z2b8mcG2ZV\
5617 qOoFjSdyvV0AZ2fayidvFJ35CS4jXzk9hir7e27zm6p3T8hLJpYciJpV1Htk/DJFU4Jw\
5618 1ImhX5IR9fzzgRkx4w/C+HQSPe+krbIyrN3qEPTNaHSaLDS2xh5Q5NCoPPVdEpgcqm/8\
5619 7/2doAptag/mlKJ77UOVG0xybTdx/Ex/Ptfa/i7r7Ku+cSoiCuxUwroUXF16wEV9H+ccVg\
5620 pd/CUFU42AK21UP1TK1L/sJjyE5PvHQr728NzvftuzvDODGy9GoopuuhNLNfctX48YHL2GH\
5621 f/8hpXVv/43rQ9xtq6YtcvLXDC3fMNDQn9bF21e7wKE1bOK65ieBu0Eqhd3IAW82dKPUw\
5622 hrauc6ZcWdkcjUZK8EUXMae71zUgwC2nbi6eVn1J19/P7eW+ioMaogF+NI3JLJf8dn9ipA\
5623 WNN4+Py9jJxuPeDL/HXzNzgtSves1D2vsWHWt9mu5rvVvZX9fo4v/LfmqdeIpHDGfM2UCW\
5624 gJTy2wOENPa23fEciVd+ZYNCNJYtrNyhyGAo8jRoJTAUmriqOCJnRW5FPTN++frrTwdh4SiUv\

```



```

5750 V9eGEnbQ/DSrW0YsRkRiQ3B4EOX/ssYpD73Wk9owbTpEezP+++jftSogyoAzc6xR/ofj5QrDY\
5751 Bnasw4zhsV+2rDYr5qZQAvdp5WetL/GQsumZyW6HmgfPJRo/y4aaU+7GfYl+LS0KkWC+3\
5752 AjxxwWCLD+BIJ07RA6LHTuc8jclEpNZZ5qZ4F58xK09H9p55d2S3TmJgu8zUPfTN+OL/PC\
5753 dc2P4UFahmsfnd47H6RP12Vnwjzrz5LufLwLBS0tF72KosqJJyIzn2FL0oagK4U6b8+BXyQ\
5754 TkKvWenYqcupTgZ2ftH6ghg26/jB8aFkNoBo59jzZLh9L+0B4E59USUQhki65Vwg6P3njyDW\
5755 85ziR500l+qabrR6TS0+rzbMqXwv2xf+csSSmQl/fCFY7LPdz2LJZrSKK+C5dELh6ixITfW\
5756 V1/nm4/cmbCW+nXmNWee48EznelWaoF+EKyUroIldOGpL2Pawp2RGfPlnIhtCOXYQ5ClgPQW\
5757 RGj4fbl+LEuV3VncC8Bf4KfZnelWaoF+EKyUroIldOGpL2Pawp2RGfPlnIhtCOXYQ5ClgPQW\
5758 KEPL9fdsBxp/2Xs6sgXK3dfclatfd8adBNlU0UNh1AEwg6Bw93seveqj1HMULPw/b14a6npg\
5759 pksWlw06fYm+v3MLU6mwfbd3KvyWstXwj2u0u04j5J+6WUyJpTL+qj1HMLPw/b14a6npg\
5760 Mg+21W6vtW1T147Y/bUNdXms71u5Baga20YX5DbUx1z9BRpGEvdrDHJ51k3m3z394VgdSYp\
5761 qZBk1kQbbVhbtEHI61/0vu/7gszaeFLR+tNOBCXy90Xa7q7BbtQ6tbu/oyIpuh8xh2A4R\
5762 o1HaDu3Q4pY2WHwWc1a17Ndi3bXo2P7v2p70cmmEfycew8L4Q6770Evh+zPnED+mNFy/W2\
5763 9LRAHh37/q05f1Lw7StTREM44Au5+Q3T6aRsqdmx7z+/G847ui290Uwv9JX4AnJ711IS\
5764 16Y+xi8smf6YcRdul4KlySH6Be9110YHs791/4cxbyvgnH2jWblj1XxXecYQZU005WDluq\
5765 Y6mFg2XReb6wTrjP5NO7ZuP3v9lrmdNn4F5eSKoH0tab6aStQot7/beUgSubPmhrC27B\
5766 0YqH510JvclY04FkKoz+kgw+oaJdVEsgVER9PehP+SxYwnkMNLm6VpOnUkXlzm+PveYqf\
5767 h4F8J1j9WOrt+64Stf500WEz2G5Cd/FZS/VXH3naqrQUL+4B2j8m8SsS/Fmi9D3McBjwo\
5768 kRn3KXuzgZpsZkZU1cbOCRjMPhQlabXm1gsdui315d3Jlr9yw0Vm9Np1kTie/CQYIdtWZV2\
5769 8/KpXqKkvclbdveIDD0Usc+RmxmD1pnmXlW78tYm6H3GdCt2fGZn+8xzSRBvQ4zrhEw9\
5770 H926s9WJSOHRzRdII7AAPQwzKI7LepLurBj0Qpxybyb/8dmn2//ll/qgnago2awgF/38L+W\
5771 I4af5Q5EXMARKAoI2CCP2xvJNV+LMZ78LkH3V27LWv+2n9w4/+63qkXJPLq7b1Tadkw1p\
5772 nIHS+QSI+HiEW5RrPuvengV20d6Nf7K0t0lF1dj/kUsatCEBEiABEiABEiABEiABEiAB\
5773 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5774 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5775 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5776 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5777 ITSmoreQR="data:image/png;base64,\
5778 iVBORw0KGgoAAANSUgEUAAGSAAABvAQMAAADYCVwJAAAAB1BMVEX///9BaePhqDaJAAAB\
5779 Hk1EQ0V4jdxTsa2EMaGYCMX7sICKVqjXVaCbe7CarASXda1LAWgS4HwM5zEVS+mv8SgS+ZBQ\
5780 8gcb4BdHyZw8szMSaUBHNM+KAd4QCLDPdn8og74UpPGci2jI8IGF3eLWpWahWYveceV\
5781 UEBDXaB0X2AnjueYD0zNklQassPckjc4nW3E1Sfwqy6jv/uAKPhg0ALSFhve8Jt0dkvDMwr\
5782 YMSGSuPyWHAr19k0t2v3sb3sdm2RUCqW88g4RplA9s1JPv9cTPlNRD4Xfkn8XaQCIV76Lzq\
5783 Z08dhw/4+U2Gzq1s8gbqVmkfr1N6YK8OqLD00mLGTWzPERA8AL9vbo0iFpS0L33f5vYtrL\
5784 S9wiqDznhUI38v5n783/gBuUs2ELgic8GAAAABJRU5ErkJggg=";
5785
5786
5787 </script>
5788
5789 <script id="gsh-script">
5790 //document.getElementById('gsh-iconurl').href = GshIcon
5791 //document.getElementById('gsh-iconurl').href = GshLogo
5792 document.getElementById('gsh-iconurl').href = ITSmoreQR
5793
5794 // id of GShell HTML elemets
5795 var E_BANNER = "gsh-banner" // banner element in HTML
5796 var E_FOOTER = "gsh-footer" // footer element in HTML
5797 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
5798 var E_GOCODE = "gsh-gocode" // Golang code of GShell
5799 var E_TODO = "gsh-todo" // TODO of GShell
5800 var E_DICT = "gsh-dict" // Dictionary of GShell
5801
5802 function bannerElem(){ return document.getElementById(E_BANNER); }
5803 function bannerStyleFunc(){ return bannerElem().style; }
5804 var bannerStyle = bannerStyleFunc()
5805 bannerStyle.backgroundImage = "url("+GshLogo+")";
5806
5807 function footerElem(){ return document.getElementById(E_FOOTER); }
5808 function footerStyle(){ return footerElem().style; }
5809 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
5810 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
5811
5812 function html_fold(e){
5813   if( e.innerHTML == "Fold" ){
5814     e.innerHTML = "Unfold"
5815     document.getElementById('gsh-menu-exit').innerHTML=""
5816     document.getElementById('gsh-statement').open=false
5817     document.getElementById('html-src').open=false
5818     document.getElementById(E_GINDEX).open=false
5819     document.getElementById(E_GOCODE).open=false
5820     document.getElementById(E_TODO).open=false
5821     document.getElementById('references').open=false
5822   }else{
5823     e.innerHTML = "Fold"
5824     document.getElementById('gsh-statement').open=true
5825     document.getElementById(E_GINDEX).open=true
5826     document.getElementById(E_GOCODE).open=true
5827     document.getElementById(E_TODO).open=true
5828     document.getElementById('references').open=true
5829   }
5830 }
5831
5832 function html_pure(e){
5833   if( e.innerHTML == "Pure" ){
5834     document.getElementById('gsh').style.display=true
5835     //document.style.display = false
5836     e.innerHTML = "Unpure"
5837   }else{
5838     document.getElementById('gsh').style.display=false
5839     //document.style.display = true
5840     e.innerHTML = "Pure"
5841   }
5842 }
5843
5844 var bannerIsStopping = false
5845 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5846 function shiftBG(){
5847   bannerIsStopping = !bannerIsStopping
5848   bannerStyle.backgroundPosition = "0 0";
5849 }
5850 // status should be inherited on Window Fork(), so use the status in DOM
5851 function html_stop(e,toggle){
5852   if( toggle ){
5853     if( e.innerHTML == "Stop" ){
5854       bannerIsStopping = true
5855       e.innerHTML = "Start"
5856     }else{
5857       bannerIsStopping = false
5858       e.innerHTML = "Stop"
5859     }
5860   }else{
5861     // update JavaScript variable from DOM status
5862     if( e.innerHTML == "Stop" ){ // shown if it's running
5863       bannerIsStopping = false
5864     }else{
5865       bannerIsStopping = true
5866     }
5867   }
5868 }
5869 //html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
5870 //html_stop(bannerElem(),false) // onInit.
5871
5872 //https://www.w3schools.com/jsref/met_win_setinterval.asp
5873 function shiftBanner(){
5874   var now = new Date().getTime();
5875   //console.log("now="+now%10)

```



```

5875     if( !bannerIsStopping ){
5876         bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
5877     }
5878 }
5879 setInterval(shiftBanner,10); // onInit.
5880
5881 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open(</a>
5882 // from embedded html to standalone page
5883 var MyChildren = 0
5884 function html_fork(){
5885     MyChildren += 1
5886     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5887     newwin = window.open("",WinId,"");
5888     src = document.getElementById("gsh");
5889     newwin.document.write("<"+<"+html>\n");
5890     newwin.document.write("<"+span id=\"gsh\">");
5891     newwin.document.write(src.innerHTML);
5892     newwin.document.write("<"+/span><"+/html>\n"); // gsh span
5893     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5894     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5895     newwin.document.close();
5896     newwin.focus();
5897 }
5898 function html_close(){
5899     window.close()
5900 }
5901 function win_jump(win){
5902     //win = window.top;
5903     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
5904     if( win == null ){
5905         console.log("jump to window.opener("+win+") (Error)\n")
5906     }else{
5907         console.log("jump to window.opener("+win+")\n")
5908         win.focus();
5909     }
5910 }
5911
5912 // source code viewr
5913 function frame_close(){
5914     srcframe = document.getElementById("src-frame");
5915     srcframe.innterHTML = "";
5916     //srcframe.style.cols = 1;
5917     srcframe.style.rows = 1;
5918     srcframe.style.height = 0;
5919     srcframe.style.display = false;
5920     src = document.getElementById("src-frame-textarea");
5921     src.innerHTML = ""
5922     //src.cols = 0
5923     src.rows = 0
5924     src.display = false
5925     //alert("--closed--")
5926 }
5927 //<!-- | <span onclick="html_view();">Source</span> -->
5928 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
5929 //<!-- | <span>Download</span> -->
5930 function frame_open(){
5931     oldsrc = document.getElementById("GENSRC");
5932     if( oldsrc != null ){
5933         //alert("--I--(erasing old text)")
5934         oldsrc.innterHTML = "";
5935         return
5936     }else{
5937         //alert("--I--(no old text)")
5938     }
5939     banner = document.getElementById('gsh-banner').style.backgroundImage;
5940     footer = document.getElementById('gsh-footer').style.backgroundImage;
5941     document.getElementById('gsh-banner').style.backgroundImage = "";
5942     document.getElementById('gsh-banner').style.backgroundPosition = "";
5943     document.getElementById('gsh-footer').style.backgroundImage = "";
5944
5945     src = document.getElementById("gsh");
5946     srcframe = document.getElementById("src-frame");
5947     srcframe.innerHTML = ""
5948     + "<"+cite id=\"GENSRC\">\n"
5949     + "<"+style>\n"
5950     + "#GENSRC textarea{tab-size:4;}\n"
5951     + "#GENSRC textarea{-o-tab-size:4;}\n"
5952     + "#GENSRC textarea{-moz-tab-size:4;}\n"
5953     + "#GENSRC textarea{spellcheck:false;}\n"
5954     + "<"+style>\n"
5955     + "<"+textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">'
5956     + /*<"+html>\n" // lost preamble text
5957     + "<"+span id=\"gsh\">" // lost preamble text
5958     + src.innerHTML
5959     + "<"+/span><"+/html>\n" // lost trail text
5960     + "<"+textarea>\n"
5961     + "<"+cite>!-- GENSRC -->\n";
5962
5963     //srcframe.style.cols = 80;
5964     //srcframe.style.rows = 80;
5965
5966     document.getElementById('gsh-banner').style.backgroundImage = banner;
5967     document.getElementById('gsh-footer').style.backgroundImage = footer;
5968 }
5969 function fill_CSSView(){
5970     part = document.getElementById('gsh-style-def')
5971     view = document.getElementById('gsh-style-view')
5972     view.innerHTML = ""
5973     + "<"+textarea cols=100 rows=20 class="gsh-code">'
5974     + part.innerHTML
5975     + "<"+/textarea>"
5976 }
5977 function fill_JavaScriptView(){
5978     jspart = document.getElementById('gsh-script')
5979     view = document.getElementById('gsh-script-view')
5980     view.innerHTML = ""
5981     + "<"+textarea cols=100 rows=20 class="gsh-code">'
5982     + jspart.innerHTML
5983     + "<"+/textarea>"
5984 }
5985 function fill_DataView(){
5986     part = document.getElementById('gsh-data')
5987     view = document.getElementById('gsh-data-view')
5988     view.innerHTML = ""
5989     + "<"+textarea cols=100 rows=20 class="gsh-code">'
5990     + part.innerHTML
5991     + "<"+/textarea>"
5992 }
5993 function jumpto_StyleView(){
5994     jsview = document.getElementById('html-src')
5995     jsview.open = true
5996     jsview = document.getElementById('gsh-style-frame')
5997     jsview.open = true
5998     fill_CSSView()
5999 }

```



```
6000 function jumpto_JavaScriptView(){
6001     jsview = document.getElementById('html-src')
6002     jsview.open = true
6003     jsview = document.getElementById('gsh-script-frame')
6004     jsview.open = true
6005     fill_JavaScriptView()
6006 }
6007 function jumpto_DataView(){
6008     jsview = document.getElementById('html-src')
6009     jsview.open = true
6010     jsview = document.getElementById('gsh-data-frame')
6011     jsview.open = true
6012     fill_DataView()
6013 }
6014 function jumpto_WholeView(){
6015     jsview = document.getElementById('html-src')
6016     jsview.open = true
6017     jsview = document.getElementById('gsh-whole-view')
6018     jsview.open = true
6019     frame_open()
6020 }
6021 function html_view(){
6022     html_stop();
6023 }
6024 banner = document.getElementById('gsh-banner').style.backgroundImage;
6025 footer = document.getElementById('gsh-footer').style.backgroundImage;
6026 document.getElementById('gsh-banner').style.backgroundImage = "";
6027 document.getElementById('gsh-banner').style.backgroundPosition = "";
6028 document.getElementById('gsh-footer').style.backgroundImage = "";
6029
6030 //srcwin = window.open("", "CodeView2", "");
6031 srcwin = window.open("", "", "");
6032 srcwin.document.write("<span id='gsh'\>\n");
6033
6034 src = document.getElementById("gsh");
6035 srcwin.document.write("<"+style>\n");
6036 srcwin.document.write("textareat{tab-size:4;}\n");
6037 srcwin.document.write("textareat{-o-tab-size:4;}\n");
6038 srcwin.document.write("textareat{-moz-tab-size:4;}\n");
6039 srcwin.document.write("</style>\n");
6040 srcwin.document.write("<h2>\n");
6041 srcwin.document.write("<"+span onclick='window.close();">Close</span> | \n");
6042 //srcwin.document.write("<"+span onclick='html_stop();">Run</span>\n");
6043 srcwin.document.write("</h2>\n");
6044 srcwin.document.write("<textarea id='gsh-src-src' cols=100 rows=60>");
6045 srcwin.document.write("<"+html>\n");
6046 srcwin.document.write("<"+span id='gsh'\>");
6047 srcwin.document.write(src.innerHTML);
6048 srcwin.document.write("<"+/span>"+/html>\n");
6049 srcwin.document.write("<"+textarea>\n");
6050
6051 document.getElementById('gsh-banner').style.backgroundImage = banner;
6052 document.getElementById('gsh-footer').style.backgroundImage = footer
6053
6054 sty = document.getElementById("gsh-style-def");
6055 srcwin.document.write("<"+style>\n");
6056 srcwin.document.write(sty.innerHTML);
6057 srcwin.document.write("<"+/style>\n");
6058
6059 run = document.getElementById("gsh-script");
6060 srcwin.document.write("<"+script>\n");
6061 srcwin.document.write(run.innerHTML);
6062 srcwin.document.write("<"+/script>\n");
6063
6064 srcwin.document.write("<"+/span>"+/html>\n"); // gsh span
6065 srcwin.document.close();
6066 srcwin.focus();
6067 }
6068 </script>
6069 -->
6070 *//<br></span></details></html>
6071
```