

```

1 //<html><details open><summary>GShell-0.2.5-HtmlArchive</summary>
2 /*<span id="gsh">
3 <link rel="icon" id="gsh-iconurl" href=""/><!-- place holder -->
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>GShell-0.2.4 by SatoxITS</title>
7 <header id="gsh-banner" height="100px" onclick="shiftBG();" style=""/>
8 <div align="right"><note>GShell version 0.2.5 // 2020-08-29 // SatoxITS</note></div>
9 </header>
10 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
11 <p>
12 <note>
13 It is a shell for myself, by myself, of myself. --SatoxITS(^-^)</note>
14 </note>
15 </p>
16 <span id="gsh-WinId" onclick="win_jump('0.1');">0</span>
17 <span id="gsh-menu">
18 | <span id="gsh-menu-exit" onclick="html_close();"></span>
19 | <span id="gsh-menu-fork" onclick="html_fork();">Fork</span>
20 | <span id="gsh-menu-stop" onclick="html_stop(this,true);">Stop</span>
21 | <span id="gsh-menu-fold" onclick="html_fold(this);">Unfold</span>
22 |<!-- |<span id="gsh-menu-pure" onclick="html_pure(this);">Pure</span> -->
23 |</span>
24 */
25 /*
26 <details id="gsh-statement" open><summary>Statement</summary><p id="gsh-statement">
27 <h2>Fun to create a shell</h2>
28 <p>For a programmer, it must be far easy and fun to create his own simple shell
29 rightly fitting to his favor and necessities, than learning existing shells with
30 complex full features that he never use.
31 I, as one of programmers, am writing this tiny shell for my own real needs,
32 totally from scratch, with fun.
33 </p><p>
34 For a programmer, it is fun to learn new computer languages. For long years before
35 writing this software, I had been specialized to C and early HTML2 :-).
36 Now writing this software, I'm learning Go language, HTML5, JavaScript and CSS
37 on demand as a novice of these, with fun.
38 </p><p>
39 This single file "gsh.go", that is executable by Go, contains all of the code written
40 in Go. Also it can be displayed as "gsh.go.html" by browsers. It is a standalone
41 HTML file that works as the viewer of the code of itself, and as the "home page" of
42 this software.
43 </p><p>
44 Because this HTML file is a Go program, you may run it as a real shell program
45 on your computer.
46 But you must be aware that this program is written under situation like above.
47 Needless to say, there is no warranty for this program in any means.
48 </p>
49 <address>Aug 2020, SatoxITS (sato@its-more.jp)</address>
50 </details>
51 */
52 /*
53 <details id="gsh-gindex" open>
54 <summary>Index</summary><div class="gsh-src">
55 Documents
56 <span class="gsh-link" onclick="jumpto_JavaScriptView();">Command summary</span>
57 Go lang part<span class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
58 Package structures
59 <a href="#import">import</a>
60 <a href="#struct">struct</a>
61 Main functions
62 <a href="#comexpansion">str-expansion</a> // macro processor
63 <a href="#finder">finder</a> // builtin find + du
64 <a href="#grep">grep</a> // builtin grep + wc + cksum + ...
65 <a href="#plugin">plugin</a> // plugin commands
66 <a href="#ex-commands">system</a> // external commands
67 <a href="#builtin">builtin</a> // builtin commands
68 <a href="#network">network</a> // socket handler
69 <a href="#remote-sh">remote-sh</a> // remote shell
70 <a href="#redirect">redirect</a> // StdIn/Out redirection
71 <a href="#history">history</a> // command history
72 <a href="#rusage">rusage</a> // resource usage
73 <a href="#encode">encode</a> // encode / decode
74 <a href="#IME">IME</a> // command line IME
75 <a href="#getline">getline</a> // line editor
76 <a href="#scanf">scanf</a> // string decomposer
77 <a href="#interpreter">interpreter</a> // command interpreter
78 <a href="#main">main</a>
79 </span>
80 JavaScript part
81 <a href="#script-src-view" class="gsh-link" onclick="jumpto_JavaScriptView();">Source</a>
82 <a href="#gsh-data-frame" class="gsh-link" onclick="jumpto_DataView();">Builtin data</a>
83 CSS part
84 <a href="#style-src-view" class="gsh-link" onclick="jumpto_StyleView();">Source</a>
85 References
86 <a href="#" class="gsh-link" onclick="jumpto_WholeView();">Internal</a>
87 <a href="#gsh-reference" class="gsh-link" onclick="jumpto_ReferenceView();">External</a>
88 Whole parts
89 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Source</a>
90 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Download</a>
91 <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Dump</a>
92 </div>
93 </details>
94 */
95 /*
96 <details id="gsh-gocode">
97 <summary>Go Source</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
98 // gsh - Go lang based Shell
99 // (c) 2020 ITS more Co., Ltd.
100 // 2020-0807 created by SatoxITS (sato@its-more.jp)
101
102 package main // gsh main
103 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
104 import (
105 "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
106 "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
107 "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
108 "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
109 "time" // <a href="https://golang.org/pkg/time/">time</a>
110 "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
111 "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
112 "os" // <a href="https://golang.org/pkg/os/">os</a>
113 "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
114 "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
115 "net" // <a href="https://golang.org/pkg/net/">net</a>
116 "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
117 "html" // <a href="https://golang.org/pkg/html/">html</a>
118 "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
119 "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
120 "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
121 "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
122 "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
123 // "gshdata" // gshell's logo and source code
124 "hash/crc32" // <a href="https://golang.org/pkg/hash/crc32/">crc32</a>

```

```

125 )
126 const (
127     NAME = "gsh"
128     VERSION = "0.2.5"
129     DATE = "2020-08-29"
130     AUTHOR = "SatoxITS(^-^)/"
131 )
132 var (
133     GSH_HOME = ".gsh" // under home directory
134     GSH_PORT = 9999
135     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
136     PROMPT = ">"
137     LINESIZE = (8*1024)
138     PATHSEP = ";" // should be ";" in Windows
139     DIRSEP = "/" // canbe \ in Windows
140 )
141
142 // -xX logging control
143 // --A-- all
144 // --I-- info.
145 // --D-- debug
146 // --T-- time and resource usage
147 // --W-- warning
148 // --E-- error
149 // --F-- fatal error
150 // --Xn- network
151
152 // <a name="struct">Structures</a>
153 type GCommandHistory struct {
154     StartAt time.Time // command line execution started at
155     EndAt time.Time // command line execution ended at
156     ResCode int // exit code of (external command)
157     CmdError error // error string
158     OutData *os.File // output of the command
159     FoundFile []string // output - result of unfid
160     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
161     CmdId int // maybe with identified with arguments or impact
162     // redirection commands should not be the CmdId
163     WorkDir string // working directory at start
164     WorkDirX int // index in ChdirHistory
165     CmdLine string // command line
166 }
167 type GChdirHistory struct {
168     Dir string
169     MovedAt time.Time
170     CmdIndex int
171 }
172 type CmdMode struct {
173     Background bool
174 }
175 type Event struct {
176     when time.Time
177     event int
178     evarg int64
179     CmdIndex int
180 }
181 var CmdIndex int
182 var Events []Event
183 type PluginInfo struct {
184     Spec *plugin.Plugin
185     Addr plugin.Symbol
186     Name string // maybe relative
187     Path string // this is in Plugin but hidden
188 }
189 type GServer struct {
190     host string
191     port string
192 }
193
194 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
195 const ( // SumType
196     SUM_ITEMS = 0x000001 // items count
197     SUM_SIZE = 0x000002 // data length (simply added)
198     SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
199     SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
200     // also envelope attributes like time stamp can be a part of digest
201     // hashed value of sizes or mod-date of files will be useful to detect changes
202
203     SUM_WORDS = 0x000010 // word count is a kind of digest
204     SUM_LINES = 0x000020 // line count is a kind of digest
205     SUM_SUM64 = 0x000040 // simple add of bytes, useful for human too
206
207     SUM_SUM32_BITS = 0x000100 // the number of true bits
208     SUM_SUM32_2BYTE = 0x000200 // 16bits words
209     SUM_SUM32_4BYTE = 0x000400 // 32bits words
210     SUM_SUM32_8BYTE = 0x000800 // 64bits words
211
212     SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
213     SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
214     SUM_UNIXFILE = 0x004000
215     SUM_CRCIEEE = 0x008000
216 )
217 type CheckSum struct {
218     Files int64 // the number of files (or data)
219     Size int64 // content size
220     Words int64 // word count
221     Lines int64 // line count
222     SumType int
223     Sum64 uint64
224     Crc32Table crc32.Table
225     Crc32Val uint32
226     Sum16 int
227     Ctime time.Time
228     Atime time.Time
229     Mtime time.Time
230     Start time.Time
231     Done time.Time
232     RusgAtStart [2]syscall.Rusage
233     RusgAtEnd [2]syscall.Rusage
234 }
235 type ValueStack [][]string
236 type GshContext struct {
237     StartDir string // the current directory at the start
238     GetLine string // gsh-getline command as a input line editor
239     ChdirHistory []GChdirHistory // the 1st entry is wd at the start
240     gshPA syscall.ProcAttr
241     CommandHistory []GCommandHistory
242     CmdCurrent GCommandHistory
243     Background bool
244     BackgroundJobs []int
245     LastRusage syscall.Rusage
246     GshHomeDir string
247     TerminalId int
248     CmdTrace bool // should be [map]
249     CmdTime bool // should be [map]

```

```

250 PluginFuncs []PluginInfo
251 iValues      []string
252 iDelimiter   string // field separator of print out
253 iFormat      string // default print format (of integer)
254 iValStack    ValueStack
255 LastServer   GServer
256 RSRV         string // [gsh://]host[:port]
257 RWD          string // remote (target, there) working directory
258 lastChecksum CheckSum
259 }
260
261 func nsleep(ns time.Duration){
262     time.Sleep(ns)
263 }
264 func usleep(ns time.Duration){
265     nsleep(ns*1000)
266 }
267 func msleep(ns time.Duration){
268     nsleep(ns*1000000)
269 }
270 func sleep(ns time.Duration){
271     nsleep(ns*1000000000)
272 }
273
274 func strBegins(str, pat string)(bool){
275     if len(pat) <= len(str){
276         yes := str[0:len(pat)] == pat
277         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat, yes)
278         return yes
279     }
280     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
281     return false
282 }
283 func isin(what string, list []string) bool {
284     for _, v := range list {
285         if v == what {
286             return true
287         }
288     }
289     return false
290 }
291 func isinX(what string,list[]string)(int){
292     for i,v := range list {
293         if v == what {
294             return i
295         }
296     }
297     return -1
298 }
299
300 func env(opts []string) {
301     env := os.Environ()
302     if isin("-s", opts){
303         sort.Slice(env, func(i,j int) bool {
304             return env[i] < env[j]
305         })
306     }
307     for _, v := range env {
308         fmt.Printf("%v\n",v)
309     }
310 }
311
312 // - rewriting should be context dependent
313 // - should postpone until the real point of evaluation
314 // - should rewrite only known notation of symbol
315 func scanInt(str string)(val int,leng int){
316     leng = -1
317     for i,ch := range str {
318         if '0' <= ch && ch <= '9' {
319             leng = i+1
320         }else{
321             break
322         }
323     }
324     if 0 < leng {
325         ival, _ := strconv.Atoi(str[0:leng])
326         return ival,leng
327     }else{
328         return 0,0
329     }
330 }
331
332 func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
333     if len(str[i+1:]) == 0 {
334         return 0,rstr
335     }
336     hi := 0
337     histlen := len(gshCtx.CommandHistory)
338     if str[i+1] == '!' {
339         hi = histlen - 1
340         leng = 1
341     }else{
342         hi,leng = scanInt(str[i+1:])
343         if leng == 0 {
344             return 0,rstr
345         }
346         if hi < 0 {
347             hi = histlen + hi
348         }
349     }
350     if 0 <= hi && hi < histlen {
351         var ext byte
352         if 1 < len(str[i+leng:]) {
353             ext = str[i+leng:][1]
354         }
355         //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
356         if ext == 'f' {
357             leng += 1
358             xlist := []string{}
359             list := gshCtx.CommandHistory[hi].FoundFile
360             for _,v := range list {
361                 //list[i] = escapeWhiteSP(v)
362                 xlist = append(xlist,escapeWhiteSP(v))
363             }
364             //rstr += strings.Join(list," ")
365             rstr += strings.Join(xlist," ")
366         }else{
367             if ext == 'e' || ext == 'd' {
368                 // !N0 .. workdir at the start of the command
369                 leng += 1
370                 rstr += gshCtx.CommandHistory[hi].WorkDir
371             }else{
372                 rstr += gshCtx.CommandHistory[hi].CmdLine
373             }
374         }
375     }else{
376         leng = 0
377     }

```

```

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

```

```

500     }else
501     if ! isin("-n",opts) {
502         fmt.Printf("\n")
503     }
504 }
505 func userHomeDir()(string,bool){
506     /*
507     homedir,_ = os.UserHomeDir() // not implemented in older Golang
508     */
509     homedir,found := os.LookupEnv("HOME")
510     //fmt.Printf("--I-- HOME=%v\n",homedir,found)
511     if !found {
512         return "/tmp",found
513     }
514     return homedir,found
515 }
516
517 func toFullpath(path string) (fullpath string) {
518     if path[0] == '/' {
519         return path
520     }
521     pathv := strings.Split(path,DIRSEP)
522     switch {
523     case pathv[0] == ".":
524         pathv[0],_ = os.Getwd()
525     case pathv[0] == "..": // all ones should be interpreted
526         cwd,_ = os.Getwd()
527         ppathv := strings.Split(cwd,DIRSEP)
528         pathv[0] = strings.Join(ppathv,DIRSEP)
529     case pathv[0] == "-":
530         pathv[0],_ = userHomeDir()
531     default:
532         cwd,_ = os.Getwd()
533         pathv[0] = cwd + DIRSEP + pathv[0]
534     }
535     return strings.Join(pathv,DIRSEP)
536 }
537
538 func IsRegFile(path string)(bool){
539     fi, err := os.Stat(path)
540     if err == nil {
541         fm := fi.Mode()
542         return fm.IsRegular();
543     }
544     return false
545 }
546
547 // <a name="encode">Encode / Decode</a>
548 // <a href="https://golang.org/pkg/encoding/base64/#example_NewEncoder">Encoder</a>
549 func (gshCtx *GshContext)Enc(argv[]string){
550     file := os.Stdin
551     buff := make([]byte,LINESIZE)
552     li := 0
553     encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
554     for li = 0; ; li++ {
555         count, err := file.Read(buff)
556         if count <= 0 {
557             break
558         }
559         if err != nil {
560             break
561         }
562         encoder.Write(buff[0:count])
563     }
564     encoder.Close()
565 }
566 func (gshCtx *GshContext)Dec(argv[]string){
567     decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
568     li := 0
569     buff := make([]byte,LINESIZE)
570     for li = 0; ; li++ {
571         count, err := decoder.Read(buff)
572         if count <= 0 {
573             break
574         }
575         if err != nil {
576             break
577         }
578         os.Stdout.Write(buff[0:count])
579     }
580 }
581 // lnspp [N] [-crlf][-C \\\]
582 func (gshCtx *GshContext)SplitLine(argv[]string){
583     reader := bufio.NewReaderSize(os.Stdin,64*1024)
584     ni := 0
585     toi := 0
586     for ni = 0; ; ni++ {
587         line, err := reader.ReadString('\n')
588         if len(line) <= 0 {
589             if err != nil {
590                 fmt.Fprintf(os.Stderr,"--I-- lnspp %d to %d (%v)\n",ni,toi,err)
591                 break
592             }
593         }
594         off := 0
595         ilen := len(line)
596         remlen := len(line)
597         for oi := 0; 0 < remlen; oi++ {
598             olen := remlen
599             addnl := false
600             if 72 < olen {
601                 olen = 72
602                 addnl = true
603             }
604             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
605                 toi,ni,oi,off,olen,remlen,ilen)
606             toi += 1
607             os.Stdout.Write([]byte(line[0:olen]))
608             if addnl {
609                 //os.Stdout.Write([]byte("\r\n"))
610                 os.Stdout.Write([]byte("\n"))
611                 os.Stdout.Write([]byte("\n"))
612             }
613             line = line[olen:]
614             off += olen
615             remlen -= olen
616         }
617     }
618     fmt.Fprintf(os.Stderr,"--I-- lnspp %d to %d\n",ni,toi)
619 }
620
621 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
622 // 1 0000 0100 1100 0001 0001 1101 1011 0111
623 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
624 var CRC32IEEE uint32 = uint32(0xEDB88320)

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

1500     if len(argv) < 2 {
1501         fmt.Printf("Sleep 100ms, 100us, 100ns, ... \n")
1502         return
1503     }
1504     duration := argv[1];
1505     d, err := time.ParseDuration(duration)
1506     if err != nil {
1507         d, err = time.ParseDuration(duration+"s")
1508         if err != nil {
1509             fmt.Printf("duration ? %s (%s)\n",duration,err)
1510             return
1511         }
1512     }
1513     //fmt.Printf("Sleep %v\n",duration)
1514     time.Sleep(d)
1515     if 0 < len(argv[2:]) {
1516         gshCtx.gshellv(argv[2:])
1517     }
1518 }
1519 func (gshCtx *GshContext)repeat(argv []string) {
1520     if len(argv) < 2 {
1521         return
1522     }
1523     start0 := time.Now()
1524     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1525         if 0 < len(argv[2:]) {
1526             //start := time.Now()
1527             gshCtx.gshellv(argv[2:])
1528             end := time.Now()
1529             elps := end.Sub(start0);
1530             if( 1000000000 < elps ){
1531                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1532             }
1533         }
1534     }
1535 }
1536
1537 func (gshCtx *GshContext)gen(argv []string) {
1538     gshPA := gshCtx.gshPA
1539     if len(argv) < 2 {
1540         fmt.Printf("Usage: %s N\n",argv[0])
1541         return
1542     }
1543     // should br repeated by "repeat" command
1544     count, _ := strconv.Atoi(argv[1])
1545     fd := gshPA.Files[1] // Stdout
1546     file := os.NewFile(fd,"internalStdOut")
1547     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1548     //buf := []byte{}
1549     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1550     for gi := 0; gi < count; gi++ {
1551         file.WriteString(outdata)
1552     }
1553     //file.WriteString("\n")
1554     fmt.Printf("\n(%d B)\n",count*len(outdata));
1555     //file.Close()
1556 }
1557
1558 // <a name="rexec">Remote Execution</a> // 2020-0820
1559 func Elapsed(from time.Time)(string){
1560     elps := time.Now().Sub(from)
1561     if 1000000000 < elps {
1562         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/10000000)
1563     }else
1564     if 1000000 < elps {
1565         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1566     }else{
1567         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1568     }
1569 }
1570 func abftime(nanos int64)(string){
1571     if 1000000000 < nanos {
1572         return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/10000000)
1573     }else
1574     if 1000000 < nanos {
1575         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1576     }else{
1577         return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1578     }
1579 }
1580 func absbsize(size int64)(string){
1581     fsize := float64(size)
1582     if 1024*1024*1024 < size {
1583         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1584     }else
1585     if 1024*1024 < size {
1586         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1587     }else{
1588         return fmt.Sprintf("%.3fKiB",fsize/1024)
1589     }
1590 }
1591 func absize(size int64)(string){
1592     fsize := float64(size)
1593     if 1024*1024*1024 < size {
1594         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1595     }else
1596     if 1024*1024 < size {
1597         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1598     }else{
1599         return fmt.Sprintf("%.3fKiB",fsize/1024)
1600     }
1601 }
1602 func abbspd(totalB int64,ns int64)(string){
1603     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1604     if 1000 <= MBs {
1605         return fmt.Sprintf("%.3fGB/s",MBs/1000)
1606     }
1607     if 1 <= MBs {
1608         return fmt.Sprintf("%.3fMB/s",MBs)
1609     }else{
1610         return fmt.Sprintf("%.3fKB/s",MBs*1000)
1611     }
1612 }
1613 func abspsd(totalB int64,ns time.Duration)(string){
1614     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1615     if 1000 <= MBs {
1616         return fmt.Sprintf("%.3fGBps",MBs/1000)
1617     }
1618     if 1 <= MBs {
1619         return fmt.Sprintf("%.3fMBps",MBs)
1620     }else{
1621         return fmt.Sprintf("%.3fKBps",MBs*1000)
1622     }
1623 }
1624 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

3125 case cmd == "close":
3126     gshCtx.xClose(argv)
3127 case cmd == "gcp":
3128     gshCtx.FileCopy(argv)
3129 case cmd == "dec" || cmd == "decode":
3130     gshCtx.Dec(argv)
3131 case cmd == "#define":
3132     case cmd == "dic":
3133         xDic(argv)
3134 case cmd == "dump":
3135     gshCtx.Dump(argv)
3136 case cmd == "echo":
3137     echo(argv,true)
3138 case cmd == "enc" || cmd == "encode":
3139     gshCtx.Enc(argv)
3140 case cmd == "env":
3141     env(argv)
3142 case cmd == "eval":
3143     xEval(argv[1:],true)
3144 case cmd == "ev" || cmd == "events":
3145     dumpEvents(0)
3146 case cmd == "exec":
3147     // = gshCtx.excommand(true,argv[1:])
3148     // should not return here
3149 case cmd == "exit" || cmd == "quit":
3150     // write Result code EXIT to 3>
3151     return true
3152 case cmd == "fds":
3153     // dump the attributes of fds (of other process)
3154 case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3155     gshCtx.xFind(argv[1:])
3156 case cmd == "fu":
3157     gshCtx.xFind(argv[1:])
3158 case cmd == "fork":
3159     // mainly for a server
3160 case cmd == "-gen":
3161     gshCtx.gen(argv)
3162 case cmd == "-go":
3163     gshCtx.xGo(argv)
3164 case cmd == "-grep":
3165     gshCtx.xFind(argv)
3166 case cmd == "gdeg":
3167     gshCtx.Deq(argv)
3168 case cmd == "geng":
3169     gshCtx.Eng(argv)
3170 case cmd == "gpop":
3171     gshCtx.Pop(argv)
3172 case cmd == "gpush":
3173     gshCtx.Push(argv)
3174 case cmd == "history" || cmd == "hi": // hi should be alias
3175     gshCtx.xHistory(argv)
3176 case cmd == "jobs":
3177     gshCtx.xJobs(argv)
3178 case cmd == "lnsp":
3179     gshCtx.SplitLine(argv)
3180 case cmd == "-ls":
3181     gshCtx.xFind(argv)
3182 case cmd == "nop":
3183     // do nothing
3184 case cmd == "pipe":
3185     gshCtx.xOpen(argv)
3186 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3187     gshCtx.xPlugin(argv[1:])
3188 case cmd == "print" || cmd == "-pr":
3189     // output internal slice // also sprintf should be
3190     gshCtx.Printv(argv)
3191 case cmd == "ps":
3192     gshCtx.xPs(argv)
3193 case cmd == "pstitle":
3194     // to be gsh.title
3195 case cmd == "rexc" || cmd == "rexc":
3196     gshCtx.RxecServer(argv)
3197 case cmd == "rexc" || cmd == "rex":
3198     gshCtx.RxecClient(argv)
3199 case cmd == "repeat" || cmd == "rep": // repeat cond command
3200     gshCtx.repeat(argv)
3201 case cmd == "replay":
3202     gshCtx.xReplay(argv)
3203 case cmd == "scan":
3204     // scan input (or so in fscanf) to internal slice (like Files or map)
3205     gshCtx.Scanv(argv)
3206 case cmd == "set":
3207     // set name ...
3208 case cmd == "serv":
3209     gshCtx.httpServer(argv)
3210 case cmd == "shift":
3211     gshCtx.Shiftv(argv)
3212 case cmd == "sleep":
3213     gshCtx.sleep(argv)
3214 case cmd == "-sort":
3215     gshCtx.Sortv(argv)
3216 case cmd == "j" || cmd == "join":
3217     gshCtx.RJoin(argv)
3218 case cmd == "a" || cmd == "alpa":
3219     gshCtx.Rxec(argv)
3220 case cmd == "jcd" || cmd == "jchdir":
3221     gshCtx.Rchdir(argv)
3222 case cmd == "jget":
3223     gshCtx.Rget(argv)
3224 case cmd == "jls":
3225     gshCtx.Rls(argv)
3226 case cmd == "jput":
3227     gshCtx.Rput(argv)
3228 case cmd == "jpwd":
3229     gshCtx.Rpwd(argv)
3230 case cmd == "time":
3231     fin = gshCtx.xTime(argv)
3232 case cmd == "ungets":
3233     if 1 < len(argv) {
3234         ungets(argv[1]+\n")
3235     }else{
3236     }
3237 case cmd == "pwd":
3238     gshCtx.xPwd(argv);
3239 case cmd == "ver" || cmd == "-ver" || cmd == "version":
3240     gshCtx.showVersion(argv)
3241 case cmd == "where":
3242     // data file or so?
3243 case cmd == "which":
3244     which("PATH",argv);
3245 default:
3246     if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3247         gshCtx.xPlugin(argv)

```

```

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

```

```

3375 * getline.c
3376 * 2020-0819 extracted from dog.c
3377 * getline.go
3378 * 2020-0822 ported to Go
3379 */
3380 /*
3381 package main // getline main
3382 import (
3383     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3384     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3385     "os" // <a href="https://golang.org/pkg/os/">os</a>
3386     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3387     //"bytes" // <a href="https://golang.org/pkg/bytes/">bytes</a>
3388     //"os/exec" // <a href="https://golang.org/pkg/os/">os</a>
3389 )
3390 */
3391
3392 // C language compatibility functions
3393 var errno = 0
3394 var stdin *os.File = os.Stdin
3395 var stdout *os.File = os.Stdout
3396 var stderr *os.File = os.Stderr
3397 var EOF = -1
3398 var NULL = 0
3399 type FILE os.File
3400 type StrBuff []byte
3401 var NULL_FP *os.File = nil
3402 var NULLSP = 0
3403 //var LINESIZE = 1024
3404
3405 func system(cmdstr string)(int){
3406     PA := syscall.ProcAttr {
3407         "", // the starting directory
3408         os.Environ(),
3409         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3410         nil,
3411     }
3412     argv := strings.Split(cmdstr, " ")
3413     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3414     if( err != nil ){
3415         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3416     }
3417     syscall.Wait4(pid,nil,0,nil)
3418
3419     /*
3420     argv := strings.Split(cmdstr, " ")
3421     fmt.Fprintf(os.Stderr, "--I-- system(%v)\n", argv)
3422     //cmd := exec.Command(argv[0],...)
3423     cmd := exec.Command(argv[0],argv[1],argv[2])
3424     cmd.Stdin = strings.NewReader("output of system")
3425     var out bytes.Buffer
3426     cmd.Stdout = &out
3427     var serr bytes.Buffer
3428     cmd.Stderr = &serr
3429     err := cmd.Run()
3430     if err != nil {
3431         fmt.Fprintf(os.Stderr, "--E-- system(%v)err(%v)\n", argv, err)
3432         fmt.Printf("ERR:%s\n",serr.String())
3433     }else{
3434         fmt.Printf("%s",out.String())
3435     }
3436     */
3437     return 0
3438 }
3439 func atoi(str string)(ret int){
3440     ret,err := fmt.Sscanf(str,"%d",ret)
3441     if err == nil {
3442         return ret
3443     }else{
3444         // should set errno
3445         return 0
3446     }
3447 }
3448 func getenv(name string)(string){
3449     val,got := os.LookupEnv(name)
3450     if got {
3451         return val
3452     }else{
3453         return "?"
3454     }
3455 }
3456 func strcpy(dst StrBuff, src string){
3457     var i int
3458     srcb := []byte(src)
3459     for i = 0; i < len(src) && srcb[i] != 0; i++ {
3460         dst[i] = srcb[i]
3461     }
3462     dst[i] = 0
3463 }
3464 func xstrcpy(dst StrBuff, src StrBuff){
3465     dst = src
3466 }
3467 func strcat(dst StrBuff, src StrBuff){
3468     dst = append(dst,src...)
3469 }
3470 func strdup(str StrBuff)(string){
3471     return string(str[0:strlen(str)])
3472 }
3473 func sstrlen(str string)(int){
3474     return len(str)
3475 }
3476 func strlen(str StrBuff)(int){
3477     var i int
3478     for i = 0; i < len(str) && str[i] != 0; i++ {
3479     }
3480     return i
3481 }
3482 func sizeof(data StrBuff)(int){
3483     return len(data)
3484 }
3485 func isatty(fd int)(ret int){
3486     return 1
3487 }
3488
3489 func fopen(file string,mode string)(fp*os.File){
3490     if mode == "r" {
3491         fp,err := os.Open(file)
3492         if( err != nil ){
3493             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
3494             return NULL_FP;
3495         }
3496         return fp;
3497     }else{
3498         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3499         if( err != nil ){

```

```

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

```

```

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

```

```

3750 //fprintf(stderr, "\r\n")
3751 xDic({}string{"dic", DIC_COM_LOAD});
3752
3753 var info = false
3754 if info {
3755     fprintf(stderr, "--Id-- Conguraturations!! WorldDic is now activated.\r\n")
3756     fprintf(stderr, "--Id-- enter \"dic\" command for help.\r\n")
3757 }
3758 }
3759 func readDic()(int){
3760 /*
3761     var rk *os.File;
3762     var dic = "MyIME-dic.txt";
3763     //rk = fopen("romkana.txt", "r");
3764     //rk = fopen("JK-JA-morse-dic.txt", "r");
3765     rk = fopen(dic, "r");
3766     if( rk == NULL_FP ){
3767         if( true ){
3768             fprintf(stderr, "--s-- Could not load %s\n", MyIMEVER, dic);
3769         }
3770         return -1;
3771     }
3772     if( true ){
3773         var di int;
3774         var line = make(StrBuff, 1024);
3775         var pat string
3776         var out string
3777         for di = 0; di < 1024; di++ {
3778             if( fgets(line, sizeof(line), rk) == NULLSP ){
3779                 break;
3780             }
3781             fmt.Sscanf(string(line[0:strlen(line)]), "%s %s", &pat, &out);
3782             //sscanf(line, "%s %[\r\n]", &pat, &out);
3783             romkana[di].pat = pat;
3784             romkana[di].out = out;
3785             //fprintf(stderr, "--Dd- %s\n", pat, out)
3786         }
3787         dicents += di
3788         if( false ){
3789             fprintf(stderr, "--s-- loaded romkana.txt [%d]\n", MyIMEVER, di);
3790             for di = 0; di < dicents; di++ {
3791                 fprintf(stderr,
3792                     "%s %s\n", romkana[di].pat, romkana[di].out);
3793             }
3794         }
3795     }
3796     fclose(rk);
3797
3798     //romkana[dicents].pat = "//ddump"
3799     //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3800     */
3801     return 0;
3802 }
3803 func matchlen(stri string, pati string)(int){
3804     if strBegins(stri, pati) {
3805         return len(pati)
3806     }else{
3807         return 0
3808     }
3809 }
3810 func convs(src string)(string){
3811     var si int;
3812     var sx = len(src);
3813     var di int;
3814     var mi int;
3815     var dstb []byte
3816
3817     for si = 0; si < sx; { // search max. match from the position
3818         if strBegins(src[si:], "%x/") {
3819             // %x/integer/ // s/a/b/
3820             ix := strings.Index(src[si+3:], "/")
3821             if 0 < ix {
3822                 var iv int = 0
3823                 //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3824                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3825                 sval := fmt.Sprintf("%x", iv)
3826                 bval := []byte(sval)
3827                 dstb = append(dstb, bval...)
3828                 si = si+3+ix+1
3829                 continue
3830             }
3831         }
3832         if strBegins(src[si:], "%d/") {
3833             // %d/integer/ // s/a/b/
3834             ix := strings.Index(src[si+3:], "/")
3835             if 0 < ix {
3836                 var iv int = 0
3837                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3838                 sval := fmt.Sprintf("%d", iv)
3839                 bval := []byte(sval)
3840                 dstb = append(dstb, bval...)
3841                 si = si+3+ix+1
3842                 continue
3843             }
3844         }
3845         if strBegins(src[si:], "%t") {
3846             now := time.Now()
3847             if true {
3848                 date := now.Format(time.Stamp)
3849                 dstb = append(dstb, []byte(date)...)
3850                 si = si+3
3851             }
3852             continue
3853         }
3854         var maxlen int = 0;
3855         var len int;
3856         mi = -1;
3857         for di = 0; di < dicents; di++ {
3858             len = matchlen(src[si:], romkana[di].pat);
3859             if( maxlen < len ){
3860                 maxlen = len;
3861                 mi = di;
3862             }
3863         }
3864         if( 0 < maxlen ){
3865             out := romkana[mi].out;
3866             dstb = append(dstb, []byte(out)...);
3867             si += maxlen;
3868         }else{
3869             dstb = append(dstb, src[si])
3870             si += 1;
3871         }
3872     }
3873     return string(dstb)
3874 }

```

```

3875 func trans(src string)(int){
3876     dst := convs(src);
3877     xfprintf(dst,stderr);
3878     return 0;
3879 }
3880
3881 //----- LINEEDIT
3882 // "?" at the top of the line means searching history
3883
3884 const (
3885     GO_UP = 201
3886     GO_DOWN = 202
3887     GO_RIGHT = 203
3888     GO_LEFT = 204
3889     DEL_RIGHT= 205
3890     EV_TIMEOUT = 206
3891     EV_IDLE = 207
3892 )
3893
3894 // should return number of octets ready to be read immediately
3895 //fprintf(stderr,"\n--Select(%v %v)\n",err,r.Bits[0])
3896
3897
3898 var EventRecvFd = -1 // file descriptor
3899 var EventSendFd = -1
3900 const EventFdOffset = 1000000
3901 const NormalFdOffset = 100
3902
3903 func putEvent(event int, evarg int){
3904     if true {
3905         if EventRecvFd < 0 {
3906             var pv = [int{-1,-1}]
3907             syscall.Pipe(pv)
3908             EventRecvFd = pv[0]
3909             EventSendFd = pv[1]
3910             //fmt.Printf("--De-- EventPipe created[%v,%v]\n",EventRecvFd,EventSendFd)
3911         }
3912     }else{
3913         if EventRecvFd < 0 {
3914             // the document differs from this spec
3915             // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L1340
3916             sv,err := syscall.Socketpair(syscall.AF_UNIX,syscall.SOCK_STREAM,0)
3917             EventRecvFd = sv[0]
3918             EventSendFd = sv[1]
3919             if err != nil {
3920                 fmt.Printf("--De-- EventSock created[%v,%v]({%v})\n",
3921                     EventRecvFd,EventSendFd,err)
3922             }
3923         }
3924     }
3925     var buf = [byte{ byte(event)}]
3926     n,err := syscall.Write(EventSendFd,buf)
3927     if err != nil {
3928         fmt.Printf("--De-- putEvent[%v]({%3v})({%v %v})\n",EventSendFd,event,n,err)
3929     }
3930 }
3931 func ungets(str string){
3932     for _,ch := range str {
3933         putEvent(int(ch),0)
3934     }
3935 }
3936 func (gsh*GshContext)xReplay(argv[]string){
3937     hix := 0
3938     if 1 < len(argv) {
3939         fmt.Sscanf(argv[1],"%d",&hix)
3940     }
3941     if hix == 0 || len(argv) <= 1 {
3942         hix = len(gsh.CommandHistory)-1
3943     }
3944     fmt.Printf("--Ir-- Replay(!%v)\n",hix)
3945     //dumpEvents(hix)
3946     //gsh.xScanReplay(hix,false)
3947     go gsh.xScanReplay(hix,true)
3948 }
3949
3950 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
3951 // 2020-0827 GShell-0.2.3
3952 func FpollIn1(fp *os.File,usec int)(uintptr){
3953     nfd := 1
3954
3955     rdv := syscall.FdSet {}
3956     fd1 := fp.Fd()
3957     bank1 := fd1/32
3958     mask1 := int32(1 << fd1)
3959     rdv.Bits[bank1] = mask1
3960
3961     fd2 := -1
3962     bank2 := -1
3963     var mask2 int32 = 0
3964
3965     if 0 <= EventRecvFd {
3966         fd2 = EventRecvFd
3967         nfd = fd2 + 1
3968         bank2 = fd2/32
3969         mask2 = int32(1 << fd2)
3970         rdv.Bits[bank2] |= mask2
3971         //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n",fd2,bank2,mask2)
3972     }
3973
3974     tout := syscall.NsecToTimeval(int64(usec*1000))
3975     //n,err := syscall.Select(nfd,&rdv,nil,nil,&tout) // spec. mismatch
3976     err := syscall.Select(nfd,&rdv,nil,nil,&tout)
3977     if err != nil {
3978         //fmt.Printf("--De-- select() err(%v)\n",err)
3979     }
3980     if err == nil {
3981         if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
3982             if false {
3983                 fmt.Printf("--De-- got Event\n")
3984             }
3985             return uintptr(EventFdOffset + fd2)
3986         }else{
3987             if (rdv.Bits[bank1] & mask1) != 0 {
3988                 return uintptr(NormalFdOffset + fd1)
3989             }else{
3990                 return 1
3991             }
3992         }else{
3993             return 0
3994         }
3995     }
3996 func fgetcTimeout1(fp *os.File,usec int)(int){
3997     readyFd := FpollIn1(fp,usec)
3998     if readyFd < 100 {
3999         return EV_TIMEOUT

```



```

4000 }
4001
4002 var buf [1]byte
4003
4004 if EventFdOffset <= readyFd {
4005     fd := int(readyFd-EventFdOffset)
4006     _,err := syscall.Read(fd,buf[0:1])
4007     if( err != nil ){
4008         return EOF;
4009     }else{
4010         return int(buf[0])
4011     }
4012 }
4013
4014 _,err := fp.Read(buf[0:1])
4015 if( err != nil ){
4016     return EOF;
4017 }else{
4018     return int(buf[0])
4019 }
4020 }
4021
4022 func visibleChar(ch int)(string){
4023     switch {
4024     case '!' <= ch && ch <= '-':
4025         return string(ch)
4026     }
4027     switch ch {
4028     case ' ': return "\\s"
4029     case '\n': return "\\n"
4030     case '\r': return "\\r"
4031     case '\t': return "\\t"
4032     }
4033     switch ch {
4034     case 0x00: return "NUL"
4035     case 0x07: return "BEL"
4036     case 0x08: return "BS"
4037     case 0x0E: return "SO"
4038     case 0x0F: return "SI"
4039     case 0x1B: return "ESC"
4040     case 0x7F: return "DEL"
4041     }
4042     switch ch {
4043     case EV_IDLE: return fmt.Sprintf("IDLE")
4044     }
4045     return fmt.Sprintf("%X",ch)
4046 }
4047 func (gsh*GshContext)xScanReplay(hix int,replay bool){
4048     var Start time.Time
4049     var events = []Event{}
4050     for _,e := range Events {
4051         if hix == 0 || e.CmdIndex == hix {
4052             events = append(events,e)
4053         }
4054     }
4055     elen := len(events)
4056     if 0 < elen {
4057         if events[elen-1].event == EV_IDLE {
4058             events = events[0:elen-1]
4059         }
4060     }
4061     for i,e := range events {
4062         nano := e.when.Nanosecond()
4063         micro := nano / 1000
4064         if Start.Second() == 0 {
4065             Start = time.Now()
4066         }
4067         diff := time.Now().Sub(Start)
4068         if replay {
4069             if e.event != EV_IDLE {
4070                 putEvent(e.event,0)
4071             }
4072         }else{
4073             fmt.Printf("#7.3fms #%-3v !%-3v [%v.%06d] %3v %02X %%-4v %10.3fms\n",
4074                 float64(diff)/1000000.0,
4075                 i,
4076                 e.CmdIndex,
4077                 e.when.Format(time.Stamp),micro,
4078                 e.event,e.event,visibleChar(e.event),
4079                 float64(e.evarg)/1000000.0)
4080         }
4081         if e.event == EV_IDLE {
4082             nsleep(time.Duration(e.evarg))
4083         }
4084     }
4085 }
4086 func dumpEvents(hix int){
4087     for i,e := range Events {
4088         nano := e.when.Nanosecond()
4089         micro := nano / 1000
4090         //if e.event != EV_TIMEOUT {
4091         if hix == 0 || e.CmdIndex == hix {
4092             fmt.Printf("#%-3v !%-3v [%v.%06d] %3v %02X %%-4v %10.3fms\n",i,
4093                 e.CmdIndex,
4094                 e.when.Format(time.Stamp),micro,
4095                 e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4096         }
4097         //}
4098     }
4099 }
4100 func fgetcTimeout(fp *os.File,usec int)(int){
4101     ch := fgetcTimeout1(fp,usec)
4102     if ch != EV_TIMEOUT {
4103         now := time.Now()
4104         if 0 < len(Events) {
4105             last := Events[len(Events)-1]
4106             dura := int64(now.Sub(last.when))
4107             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4108         }
4109         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4110     }
4111     return ch
4112 }
4113
4114 var TtyMaxCol = 72
4115 var EscTimeout = (100*1000)
4116 var (
4117     MODE_ShowMode bool
4118     romkanmode bool
4119     MODE_CapsLock bool // software CapsLock
4120     MODE_LowerLock bool // force lower-case character lock
4121     MODE_ViInsert int // visible insert mode, should be like "I" icon in X Window
4122     MODE_ViTrace bool // output newline before translation
4123 )
4124 type IInput struct {

```

```

4125 lno      int
4126 lastlno  int
4127 pch      []int // input queue
4128 prompt   string
4129 line     string
4130 right    string
4131 inJmode  bool
4132 pinJmode bool
4133 waitingMeta string // waiting meta character
4134 lastCmd  string
4135 }
4136 func (iin*IInput)Getc(timeoutUs int)(int){
4137     ch1 := EOF
4138     ch2 := EOF
4139     ch3 := EOF
4140     if( 0 < len(iin.pch) ){ // deQ
4141         ch1 = iin.pch[0]
4142         iin.pch = iin.pch[1:]
4143     }else{
4144         ch1 = fgetcTimeout(stdin,timeoutUs);
4145     }
4146     if( ch1 == 033 ){ // escape sequence
4147         ch2 = fgetcTimeout(stdin,EscTimeout);
4148         if( ch2 == EV_TIMEOUT ){
4149             }else{
4150                 ch3 = fgetcTimeout(stdin,EscTimeout);
4151                 if( ch3 == EV_TIMEOUT ){
4152                     iin.pch = append(iin.pch,ch2) // enQ
4153                 }else{
4154                     switch( ch2 ){
4155                         default:
4156                             iin.pch = append(iin.pch,ch2) // enQ
4157                             iin.pch = append(iin.pch,ch3) // enQ
4158                         case '|':
4159                             switch( ch3 ){
4160                                 case 'A': ch1 = GO_UP; // ^
4161                                 case 'B': ch1 = GO_DOWN; // v
4162                                 case 'C': ch1 = GO_RIGHT; // >
4163                                 case 'D': ch1 = GO_LEFT; // <
4164                                 case '3':
4165                                     ch4 := fgetcTimeout(stdin,EscTimeout);
4166                                     if( ch4 == '-' ){
4167                                         //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4168                                         ch1 = DEL_RIGHT
4169                                     }
4170                                 }
4171                             case '\\':
4172                                 //ch4 := fgetcTimeout(stdin,EscTimeout);
4173                                 //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4174                                 switch( ch3 ){
4175                                     case '-': ch1 = DEL_RIGHT
4176                                 }
4177                             }
4178                     }
4179                 }
4180             }
4181         return ch1
4182     }
4183 func (inn*IInput)clearline(){
4184     var i int
4185     fprintf(stderr,"r");
4186     // should be ANSI ESC sequence
4187     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4188         fputc(' ',os.Stderr);
4189     }
4190     fprintf(stderr,"r");
4191 }
4192 func (iin*IInput)Redraw(){
4193     redraw(iin,iin.lno,iin.line,iin.right)
4194 }
4195 func redraw(iin *IInput,lno int,line string,right string){
4196     inMeta := false
4197     showMode := ""
4198     showMeta := "" // visible Meta mode on the cursor position
4199     showLino := fmt.Sprintf("%d!", lno)
4200     InsertMark := "" // in visible insert mode
4201
4202     if 0 < len(iin.right) {
4203         InsertMark = " "
4204     }
4205
4206     if( 0 < len(iin.waitingMeta) ){
4207         inMeta = true
4208         if iin.waitingMeta[0] != 033 {
4209             showMeta = iin.waitingMeta
4210         }
4211     }
4212     if( romkanmode ){
4213         //romkanmark = " *";
4214     }else{
4215         //romkanmark = "";
4216     }
4217     if MODE_ShowMode {
4218         romkan := "--"
4219         inmeta := "-"
4220         inveri := ""
4221         if MODE_CapsLock {
4222             inmeta = "A"
4223         }
4224         if MODE_LowerLock {
4225             inmeta = "a"
4226         }
4227         if MODE_ViTrace {
4228             inveri = "v"
4229         }
4230         if romkanmode {
4231             romkan = "\343\201\202"
4232             if MODE_CapsLock {
4233                 inmeta = "R"
4234             }else{
4235                 inmeta = "r"
4236             }
4237         }
4238         if inMeta {
4239             inmeta = "\\ "
4240         }
4241         showMode = "["+romkan+inmeta+inveri+"]";
4242     }
4243     Pre := "\r" + showMode + showLino
4244     Output := ""
4245     Left := ""
4246     Right := ""
4247     if romkanmode {
4248         Left = convs(line)
4249         Right = InsertMark+convs(right)

```

```

4250     }else{
4251         Left = line
4252         Right = InsertMark+right
4253     }
4254     Output = Pre+Left
4255     if MODE_ViTrace {
4256         Output += iin.LastCmd
4257     }
4258     Output += showMeta+Right
4259     for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4260         Output += " "
4261         // should be ANSI ESC sequence
4262         // not necessary just after newline
4263     }
4264     Output += Pre+Left+showMeta // to set the cursor to the current input position
4265     fprintf(stderr,"%s",Output)
4266
4267     if MODE_ViTrace {
4268         if 0 < len(iin.LastCmd) {
4269             iin.LastCmd = ""
4270             fprintf(stderr,"\r\n")
4271         }
4272     }
4273 }
4274 func delHeadChar(str string)(rline string,head string){
4275     _,clen := utf8.DecodeRune([]byte(str))
4276     head = string(str[0:clen])
4277     return str[clen:],head
4278 }
4279 func delTailChar(str string)(rline string, last string){
4280     var i = 0
4281     var clen = 0
4282     for {
4283         _,siz := utf8.DecodeRune([]byte(str)[i:])
4284         if siz <= 0 { break }
4285         clen = siz
4286         i += siz
4287     }
4288     last = str[len(str)-clen:]
4289     return str[0:len(str)-clen],last
4290 }
4291
4292 // 3> for output and history
4293 // 4> for keylog?
4294 // <a name="getline">Command Line Editor</a>
4295 func xgetline(lno int, prevline string, gsh*GshContext)(string){
4296     var iin Iinput
4297     iin.lastlno = lno
4298     iin.lno = lno
4299
4300     CmdIndex = len(gsh.CommandHistory)
4301     if( isatty(0) == 0 ){
4302         if( sfgets(&iin.line,LINESIZE,stdin) == NULL ){
4303             iin.line = "exit\n";
4304         }else{
4305             return iin.line
4306         }
4307     }
4308     if( true ){
4309         //var pts string;
4310         //pts = ptsname(0);
4311         //pts = ttyname(0);
4312         //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
4313     }
4314     if( false ){
4315         fprintf(stderr,"! ");
4316         fflush(stderr);
4317         sfgets(&iin.line,LINESIZE,stdin);
4318         return iin.line
4319     }
4320     system("/bin/stty -echo -icanon");
4321     xline := iin.xgetline(prevline,gsh)
4322     system("/bin/stty echo sane");
4323     return xline
4324 }
4325 func (iin*Iinput)Translate(cmdch int){
4326     romkanmode = !romkanmode;
4327     if MODE_ViTrace {
4328         fprintf(stderr,"%v\r\n",string(cmdch));
4329     }else
4330     if( cmdch == 'J' ){
4331         fprintf(stderr,"J\r\n");
4332         iin.inJmode = true
4333     }
4334     iin.Redraw();
4335     loadDefaultDic(cmdch);
4336     iin.Redraw();
4337 }
4338 func (iin*Iinput)Replace(cmdch int){
4339     iin.LastCmd = fmt.Sprintf("\\%v",string(cmdch))
4340     iin.Redraw();
4341     loadDefaultDic(cmdch);
4342     dst := convs(iin.line+iin.right);
4343     iin.line = dst
4344     iin.right = ""
4345     if( cmdch == 'I' ){
4346         fprintf(stderr,"I\r\n");
4347         iin.inJmode = true
4348     }
4349     iin.Redraw();
4350 }
4351 func (iin*Iinput)xgetline(prevline string, gsh*GshContext)(string){
4352     var ch int;
4353     iin.Redraw();
4354     for {
4355         iin.pinJmode = iin.inJmode
4356         iin.inJmode = false
4357
4358         ch = iin.Getc(1000*1000)
4359         //fprintf(stderr,"A[%02X]\n",ch);
4360         if( ch == '\\ ' || ch == 033 ){
4361             MODE_ShowMode = true
4362             metach := ch
4363             iin.waitingMeta = string(ch)
4364             iin.Redraw();
4365             // set cursor //fprintf(stderr,"???\b\b\b")
4366             ch = fgetcTimeout(stdin,2000*1000)
4367             // reset cursor
4368             iin.waitingMeta = ""
4369
4370             cmdch := ch
4371             if( ch == EV_TIMEOUT ){
4372                 if metach == 033 {
4373                     continue
4374                 }

```

```

4375     ch = metach
4376 }else
4377 if( ch == 'j' || ch == 'J' ){
4378     iin.Translate(cmdch);
4379     continue
4380 }else
4381 if( ch == 'i' || ch == 'I' ){
4382     iin.Replace(cmdch);
4383     continue
4384 }else
4385 if( ch == 'l' || ch == 'L' ){
4386     MODE_LowerLock = IMODE_LowerLock
4387     MODE_CapsLock = false
4388     if MODE_ViTrace {
4389         fprintf(stderr, "%v\r\n", string(cmdch));
4390     }
4391     iin.Redraw();
4392     continue
4393 }else
4394 if( ch == 'u' || ch == 'U' ){
4395     MODE_CapsLock = IMODE_CapsLock
4396     MODE_LowerLock = false
4397     if MODE_ViTrace {
4398         fprintf(stderr, "%v\r\n", string(cmdch));
4399     }
4400     iin.Redraw();
4401     continue
4402 }else
4403 if( ch == 'v' || ch == 'V' ){
4404     MODE_ViTrace = IMODE_ViTrace
4405     if MODE_ViTrace {
4406         fprintf(stderr, "%v\r\n", string(cmdch));
4407     }
4408     iin.Redraw();
4409     continue
4410 }else
4411 if( ch == 'c' || ch == 'C' ){
4412     if 0 < len(iin.line) {
4413         xline,tail := delTailChar(iin.line)
4414         if len([]byte(tail)) == 1 {
4415             ch = int(tail[0])
4416             if( 'a' <= ch && ch <= 'z' ){
4417                 ch = ch + 'A'-'a'
4418             }else
4419             if( 'A' <= ch && ch <= 'z' ){
4420                 ch = ch + 'a'-'A'
4421             }
4422             iin.line = xline + string(ch)
4423         }
4424     }
4425     if MODE_ViTrace {
4426         fprintf(stderr, "%v\r\n", string(cmdch));
4427     }
4428     iin.Redraw();
4429     continue
4430 }else{
4431     iin.pch = append(iin.pch,ch) // push
4432     ch = '\\'
4433 }
4434 }
4435 switch( ch ){
4436 case 'P'-0x40: ch = GO_UP
4437 case 'N'-0x40: ch = GO_DOWN
4438 case 'B'-0x40: ch = GO_LEFT
4439 case 'F'-0x40: ch = GO_RIGHT
4440 }
4441 //fprintf(stderr, "B[802X]\n",ch);
4442 switch( ch ){
4443 case 0:
4444     continue;
4445
4446 case '\t':
4447     iin.Replace('j');
4448     continue
4449 case 'X'-0x40:
4450     iin.Replace('j');
4451     continue
4452
4453 case EV_TIMEOUT:
4454     iin.Redraw();
4455     if iin.pinJmode {
4456         fprintf(stderr, "\\J\r\n")
4457         iin.inJmode = true
4458     }
4459     continue
4460 case GO_UP:
4461     if iin.lno == 1 {
4462         continue
4463     }
4464     cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
4465     if ok {
4466         iin.line = cmd
4467         iin.right = ""
4468         iin.lno = iin.lno - 1
4469     }
4470     iin.Redraw();
4471     continue
4472 case GO_DOWN:
4473     cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
4474     if ok {
4475         iin.line = cmd
4476         iin.right = ""
4477         iin.lno = iin.lno + 1
4478     }else{
4479         iin.line = ""
4480         iin.right = ""
4481         if iin.lno == iin.lastlno-1 {
4482             iin.lno = iin.lno + 1
4483         }
4484     }
4485     iin.Redraw();
4486     continue
4487 case GO_LEFT:
4488     if 0 < len(iin.line) {
4489         xline,tail := delTailChar(iin.line)
4490         iin.line = xline
4491         iin.right = tail + iin.right
4492     }
4493     iin.Redraw();
4494     continue;
4495 case GO_RIGHT:
4496     if( 0 < len(iin.right) && iin.right[0] != 0 ){
4497         xright,head := delHeadChar(iin.right)
4498         iin.right = xright
4499         iin.line += head

```

```

4500     }
4501     iin.Redraw();
4502     continue;
4503 case EOF:
4504     goto EXIT;
4505 case 'R'-0x40: // replace
4506     dst := convs(iin.line+iin.right);
4507     iin.line = dst
4508     iin.right = ""
4509     iin.Redraw();
4510     continue;
4511 case 'T'-0x40: // just show the result
4512     readDic();
4513     romkanmode = !romkanmode;
4514     iin.Redraw();
4515     continue;
4516 case 'I'-0x40:
4517     iin.Redraw();
4518     continue
4519 case 'K'-0x40:
4520     iin.right = ""
4521     iin.Redraw();
4522     continue
4523 case 'E'-0x40:
4524     iin.line += iin.right
4525     iin.right = ""
4526     iin.Redraw();
4527     continue
4528 case 'A'-0x40:
4529     iin.right = iin.line + iin.right
4530     iin.line = ""
4531     iin.Redraw();
4532     continue
4533 case 'U'-0x40:
4534     iin.line = ""
4535     iin.right = ""
4536     iin.clearline();
4537     iin.Redraw();
4538     continue;
4539 case DEL RIGHT:
4540     if( 0 < len(iin.right) ){
4541         iin.right,_ = delHeadChar(iin.right)
4542         iin.Redraw();
4543     }
4544     continue;
4545 case 0x7F: // BS? not DEL
4546     if( 0 < len(iin.line) ){
4547         iin.line,_ = delTailChar(iin.line)
4548         iin.Redraw();
4549     }
4550     /*
4551     else
4552     if( 0 < len(iin.right) ){
4553         iin.right,_ = delHeadChar(iin.right)
4554         iin.Redraw();
4555     }
4556     */
4557     continue;
4558 case 'H'-0x40:
4559     if( 0 < len(iin.line) ){
4560         iin.line,_ = delTailChar(iin.line)
4561         iin.Redraw();
4562     }
4563     continue;
4564 }
4565 if( ch == '\n' || ch == '\r' ){
4566     iin.line += iin.right;
4567     iin.right = ""
4568     iin.Redraw();
4569     fputc(ch,stderr);
4570     break;
4571 }
4572 if MODE_CapsLock {
4573     if 'a' <= ch && ch <= 'z' {
4574         ch = ch+'A'-'a'
4575     }
4576 }
4577 if MODE_LowerLock {
4578     if 'A' <= ch && ch <= 'Z' {
4579         ch = ch+'a'-'A'
4580     }
4581 }
4582 iin.line += string(ch);
4583 iin.Redraw();
4584 }
4585 EXIT:
4586     return iin.line + iin.right;
4587 }
4588
4589 func getline_main(){
4590     line := Xgetline(0,"",nil)
4591     fprintf(stderr,"%s\n",line);
4592 /*
4593     dp = strpbrk(line,"\r\n");
4594     if( dp != NULL ){
4595         *dp = 0;
4596     }
4597
4598     if( 0 ){
4599         fprintf(stderr,"\n(%d)\n",int(strlen(line)));
4600     }
4601     if( lseek(3,0,0) == 0 ){
4602         if( romkanmode ){
4603             var buf [8*1024]byte;
4604             convs(line,buf);
4605             strcpy(line,buf);
4606         }
4607         write(3,line,strlen(line));
4608         ftruncate(3,lseek(3,0,SEEK_CUR));
4609         //fprintf(stderr,"outsize=%d\n",(int)lseek(3,0,SEEK_END));
4610         lseek(3,0,SEEK_SET);
4611         close(3);
4612     }else{
4613         fprintf(stderr,"\r\ngotline: ");
4614         trans(line);
4615         //printf("%s\n",line);
4616         printf("\n");
4617     }
4618 */
4619 }
4620 //== end ===== getline
4621
4622 //
4623 // $USERHOME/.gsh/
4624 // gsh-rc.txt, or gsh-configure.txt

```

```

4625 //          gsh-history.txt
4626 //          gsh-aliases.txt // should be conditional?
4627 //
4628 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
4629     homedir_found := userHomeDir()
4630     if !found {
4631         fmt.Printf("--E-- You have no UserHomeDir\n")
4632         return true
4633     }
4634     gshhome := homedir + "/" + GSH_HOME
4635     _, err2 := os.Stat(gshhome)
4636     if err2 != nil {
4637         err3 := os.Mkdir(gshhome,0700)
4638         if err3 != nil {
4639             fmt.Printf("--E-- Could not Create %s (%s)\n",
4640                 gshhome,err3)
4641             return true
4642         }
4643         fmt.Printf("--I-- Created %s\n",gshhome)
4644     }
4645     gshCtx.GshHomeDir = gshhome
4646     return false
4647 }
4648 func setupGshContext()(GshContext,bool){
4649     gshPA := syscall.ProcAttr {
4650         "", // the starting directory
4651         os.Environ(), // environ[]
4652         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
4653         nil, // OS specific
4654     }
4655     cwd, _ := os.Getwd()
4656     gshCtx := GshContext {
4657         cwd, // StartDir
4658         "", // GetLine
4659         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
4660         gshPA,
4661         []GCommandHistory{}, //something for invokation?
4662         GCommandHistory{}, // CmdCurrent
4663         false,
4664         []jint{},
4665         syscall.Rusage{},
4666         "", // GshHomeDir
4667         Ttyid(),
4668         false,
4669         false,
4670         []PluginInfo{},
4671         []string{},
4672         "",
4673         "v",
4674         ValueStack{},
4675         GServer{"", ""}, // LastServer
4676         "", // RSERV
4677         cwd, // RND
4678         CheckSum{},
4679     }
4680     err := gshCtx.gshSetupHomedir()
4681     return gshCtx, err
4682 }
4683 func (gsh*GshContext)gshellh(gline string)(bool){
4684     ghist := gsh.CmdCurrent
4685     ghist.WorkDir,_ = os.Getwd()
4686     ghist.WorkDir = len(gsh.ChdirHistory)-1
4687     //fmt.Printf("--D--ChdirHistory(%d)\n",len(gsh.ChdirHistory))
4688     ghist.StartAt = time.Now()
4689     rusagev1 := Getrusagev()
4690     gsh.CmdCurrent.FoundFile = []string{}
4691     fin := gsh.tgshellh(gline)
4692     rusagev2 := Getrusagev()
4693     ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
4694     ghist.EndAt = time.Now()
4695     ghist.CmdLine = gline
4696     ghist.FoundFile = gsh.CmdCurrent.FoundFile
4697
4698     /* record it but not show in list by default
4699     if len(gline) == 0 {
4700         continue
4701     }
4702     if gline == "hi" || gline == "history" { // don't record it
4703         continue
4704     }
4705     */
4706     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
4707     return fin
4708 }
4709 // <a name="main">Main loop</a>
4710 func script(gshCtxGiven *GshContext) (_ GshContext) {
4711     gshCtxBuf,err0 := setupGshContext()
4712     if err0 {
4713         return gshCtxBuf;
4714     }
4715     gshCtx := &gshCtxBuf
4716
4717     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
4718     //resmap()
4719
4720     /*
4721     if false {
4722         gsh_getlinev, with_exgetline :=
4723             which("PATH",[]string{"which","gsh-getline","-s"})
4724         if with_exgetline {
4725             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
4726             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
4727         }else{
4728             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
4729         }
4730     }
4731     */
4732
4733     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
4734     gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
4735
4736     prevline := ""
4737     skipping := false
4738     for hix := len(gshCtx.CommandHistory); ; {
4739         gline := gshCtx.getline(hix,skipping,prevline)
4740         if skipping {
4741             if strings.Index(gline,"fi") == 0 {
4742                 fmt.Printf("fi\n");
4743                 skipping = false;
4744             }else{
4745                 //fmt.Printf("%s\n",gline);
4746             }
4747             continue
4748         }
4749         if strings.Index(gline,"if") == 0 {

```

```

4750 //fmt.Printf("--D-- if start: %s\n",gline);
4751 skipping = true;
4752 continue
4753 }
4754 if false {
4755 os.Stdout.Write([]byte("gotline:"))
4756 os.Stdout.Write([]byte(gline))
4757 os.Stdout.Write([]byte("\n"))
4758 }
4759 gline = strsubst(gshCtx,gline,true)
4760 if false {
4761 fmt.Printf("fmt.Printf %%v - %v\n",gline)
4762 fmt.Printf("fmt.Printf %%s - %s\n",gline)
4763 fmt.Printf("fmt.Printf %%x - %s\n",gline)
4764 fmt.Printf("fmt.Printf %%U - %s\n",gline)
4765 fmt.Printf("Stout.Write -")
4766 os.Stdout.Write([]byte(gline))
4767 fmt.Printf("\n")
4768 }
4769 /*
4770 // should be cared in substitution ?
4771 if 0 < len(gline) && gline[0] == '!' {
4772 xgline, set, err := searchHistory(gshCtx,gline)
4773 if err {
4774 continue
4775 }
4776 if set {
4777 // set the line in command line editor
4778 }
4779 gline = xgline
4780 }
4781 */
4782 fin := gshCtx.gshelllh(gline)
4783 if fin {
4784 break;
4785 }
4786 prevline = gline;
4787 hix++;
4788 }
4789 return *gshCtx
4790 }
4791 func main() {
4792 gshCtxBuf := GshContext{}
4793 gsh := &gshCtxBuf
4794 argv := os.Args
4795 if 1 < len(argv) {
4796 if isin("version",argv){
4797 gsh.showVersion(argv)
4798 return
4799 }
4800 comx := isinX("-c",argv)
4801 if 0 < comx {
4802 gshCtxBuf,err := setupGshContext()
4803 gsh := &gshCtxBuf
4804 if !err {
4805 gsh.gshellv(argv[comx+1:])
4806 }
4807 return
4808 }
4809 }
4810 if 1 < len(argv) && isin("-s",argv) {
4811 }else{
4812 gsh.showVersion(append(argv,[jstring{"-l","-a"}...])
4813 }
4814 script(nil)
4815 //gshCtx := script(nil)
4816 //gshell(gshCtx,"time")
4817 }
4818 </div></details>
4819 <div id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
4820 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
4821 // - merged histories of multiple parallel gsh sessions
4822 // - alias as a function or macro
4823 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
4824 // - retrieval PATH of files by its type
4825 // - gsh as an IME with completion using history and file names as dictioaies
4826 // - gsh a scheduler in precise time of within a millisecond
4827 // - all commands have its subucomand after "---" symbol
4828 // - filename expansion by "-find" command
4829 // - history of ext code and output of each commoand
4830 // - "script" output for each command by pty-tee or telnet-tee
4831 // - $BULLETIN command in PATH to show the priority
4832 // - "?" symbol in the command (not as in arguments) shows help request
4833 // - searching command with wild card like: which ssh-*
4834 // - longformat prompt after long idle time (should dismiss by BS)
4835 // - customizing by building plugin and dynamically linking it
4836 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
4837 // - "!" symbol should be used for negation, don't wast it just for job control
4838 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
4839 // - making canonical form of command at the start adding quation or white spaces
4840 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
4841 // - name? or name! might be useful
4842 // - htar format - packing directory contents into a single html file using data scheme
4843 // - filepath substitution should be done by each command, expecially in case of builtins
4844 // - @N substitution for the history of working directory, and @spec for more generic ones
4845 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
4846 // - GSH_PATH for plugins
4847 // - standard command output: list of data with name, size, resouce usage, modified time
4848 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
4849 // -wc word-count, grep match line count, ...
4850 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
4851 // - -tailf-filename like tail -f filename, repeat close and open before read
4852 // - max. size and max. duration and timeout of (generated) data transfer
4853 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
4854 // - IME "?" at the top of the command line means searching history
4855 // - IME %d/0x10000/ %x/ffff/
4856 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
4857 // - gsh in WebAssembly
4858 // - gsh as a HTTP server of online-manual
4859 //---END--- (^~)/ITS more</div></details>
4860
4861 <span class="gsh-golang-data">
4862 var WorldDic = <span id="gsh-world-dic">
4863 "data:text/dic;base64, "+
4864 "Ly8tX1JlTUUVNCAwLjEg6L6e5pu4ICgyMDIwLTA4MTlhKQpzZWthaSDkUJbnlYwKa28g44GT"+
4865 "Cm5uIOOCkwpuaSDjgasKY2hpIOOBoQp0aSDjgaEKaEGg44GvCnNlIOO8mwpYrYSDjgYsKaSDj"+
4866 "gYQK";
4867 </span>
4868 var JA_JKLDic = <span id="gsh-ja-jkl-dic">
4869 "data:text/dic;base64, "+
4870 "Ly92ZXJscCU15SU1FamRyY2ptb3JzZWpKQWpKS0wMjAyYmGowODE5KSHeLV4pL1NhdcG94SVRT"+
4871 "CmtqamprbGta2tsa2psIOS4lueVjApqamtqamwJ44GCMtqbAnjgYQKa2tqbAnjgYVYKamtq"+
4872 "amwJ44GCMtqa2trbAnjgYoKa2pra2wJ44GLCmpramtrbAnjgY0Ka2tramwJ44GCPmcramps"+
4873 "CeOBkOqamprbAnjgZMKamtqa2psCeOB1Opqamtqa2wJ44GXCMpqamtqbAnjgZkKa2pqamts"+
4874 "CeOBmwpqamprbAnjgZOKamtsCeOBnwpva2prbAnjgaEKa2pqa2wJ44GkCmtqa2pqbAnjgaYK"+

```

```

4875 "a2tqa2tsCe0BgApramtsCe0Bggpqa2prbAnjgasKa2tra2wJ44GsCmpqa2psCe0BrQpra2pg"+
4876 "banjga4Kamtra2wJ44GvCmpqa2tqbAnjgbiKamptra2wJ44G1CmtsCe0BuApqa2tsCe0Buwpg"+
4877 "a2tqbAnjgb4Ka2tga2psCe0BvwpqbAnjgoAKamtra2psCe0CgQpqa2tga2wJ44KCmtgamwJ"+
4878 "44KCmptra2pqbAnjgoKampsCe0CiApra2tsCe0C1QpgamtsCe0Cigpqa2pqa2wJ44KLCmpg"+
4879 "amwJ44KCmtqa2psCe0CjQpqa2psCe0CjwpramtramwJ44KQCmtgamrbanjgpbKa2pqa2wJ"+
4880 "44KCmtqa2prbAnjgpbKa2pqa2psCe0DvApra2wJ44KbCmtramprbanjgpbKa2pramtqbAnj"+
4881 "gIER";
4882 //</span>
4883 //</span>
4884 /*
4885 <details id="references"><summary>References</summary><div class="gsh-src">
4886 <p>
4887 <a href="https://golang.org">The Go Programming Language</a>
4888 <iframe src="https://golang.org" width="100%" height="300"></iframe>
4889
4890 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
4891 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
4892 CSS:
4893 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors">Selectors</a>
4894 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
4895 HTTP
4896 JavaScript:
4897 .
4898 </p>
4899 </div></details>
4900 */
4901 /*
4902 <details id="html-src" onclick="frame_open();"><summary>Raw Source</summary><div>
4903
4904 <!-- h2>The full of this HTML including the Go code is here.</h2 -->
4905 <details id="gsh-whole-view"><summary>Whole file</summary>
4906 <a name="whole-src-view"></a>
4907 <span id="src-frame"></span><!-- a window to show source code -->
4908 </details>
4909
4910 <details id="gsh-style-frame" onclick="fill_CSSView()"><summary>CSS part</summary>
4911 <a name="style-src-view"></a>
4912 <span id="gsh-style-view"></span>
4913 </details>
4914
4915 <details id="gsh-script-frame" onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
4916 <a name="script-src-view"></a>
4917 <span id="gsh-script-view"></span>
4918 </details>
4919
4920 <details id="gsh-data-frame" onclick="fill_DataView()"><summary>Builtin data part</summary>
4921 <a name="gsh-data-frame"></a>
4922 <span id="gsh-data-view"></span>
4923 </details>
4924
4925 </div></details>
4926 */
4927 /*
4928 <div id="gsh-footer" style=""></div><!-- END-OF-VISIBLE-PART ----- -->
4929
4930
4931 <style id="gsh-style-def">
4932 //body {display:none;}
4933 .gsh-link{color:green;}
4934 #gsh {border-width:1px;margin:0;padding:0;}
4935 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
4936 #gsh header{height:100px;}
4937 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
4938 #gsh-menu{font-size:14pt;color:#f88;}
4939 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
4940 #gsh note{color:#000;font-size:10pt;}
4941 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
4942 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
4943 #gsh summary{font-size:16pt;color:#fff;background-color:#8af,height:30px;}
4944 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
4945 #gsh a{color:#24a;}
4946 #gsh a[name]{color:#24a;font-size:16pt;}
4947 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
4948 #gsh .gsh-src{background-color:#faffff;color:#223;}
4949 #gsh-src-src{spellcheck:false}
4950 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
4951 #src-frame-textarea{background-color:#faffff;color:#223;}
4952 .gsh-code {white-space:pre;font-family:monospace !import;}
4953 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
4954 .gsh-golang-data {display:none;}
4955 #gsh-winId {color:#000;font-size:14pt;}
4956
4957 #gsh-statement {font-size:11pt;background-color:#fff;font-family:Georgia;}
4958 #gsh-statement {color:#000;background-color:#fff !import;}
4959 #gsh-statement h2{color:#000;background-color:#fff !import;}
4960 #gsh-statement details{color:#000;background-color:#fff;font-family:Georgia;}
4961 #gsh-statement p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
4962 #gsh-statement address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
4963
4964 @media print {
4965 #gsh pre{font-size:11pt !import;}
4966 }
4967 </style>
4968
4969 <!--
4970 // Logo image should be drawn by JavaScript from a meta-font.
4971 // CSS seems not follow line-splitted URL
4972 -->
4973 <script id="gsh-data">
4974 //GshLogo="QR-ITS-more.jp.png"
4975 GshLogo="data:image/png;base64,\
4976 iVBORw0KGgoAAAANSUHUeUgAAAEAAAB/CAYAADvs3f4AAAAAXNSR0Iars4c6QAAAHl1WE1m\
4977 TU0AKgAAAAGBAEaAAUAAAABAAAAPgEbaAAUAAAABAAAARgEoAAAAAABAAAIAidpAAQAAAAAB\
4978 AAAATgAAAAAABAAIAAAAAQAAAEgAAAAAABAAQQAADAAAAAQAABAACgAgEAAAAAQAAGGAWAE\
4979 AAAAAQAAAH8AAAAAYx1BhgAAAA1wSF1zAAALEwAACMBAJgcGAAAF3RJREFUeAhtnQUfUNw\
4980 x++t7ukZ3iCggO/0Y60sb8WgmZavn7uG4+biSTR7YnXQDQPCkCj2aNU1D2MS1rKeUAPnoCdu\
4981 4iUJx7jr1YZ50D0GmF2VqIBEiSggCoIMMA+mu+vu//ZMD9U1da6a2au9bV1GK9r3vvd6/q\
4982 fnXdx8tBA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4983 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4984 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4985 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4986 2eXs9H9+ftKsDhXs1c2qgdE7YusS+1qaalKfnY5YsokMhWEptd4MQFz5UEExLbLYsAU15\
4987 npdiLKXEZC1FiRm53JSUaq9ScqcU6i+2kk3StuOnY5reEGKJ7Qw7m0vKec2Toq0iZw0jhFS\
4988 jboVHCstMRb3USXEJ8hFu7DsdmF2+u4vWVWFxbBMeZULAE/hcKoGAb6eKGo1Nyhk56PC\
4989 HxI2VVBKORkqh3qUeKi1YdaOfONJ56OkdI6v5BwomnOQlyPzi0N9DLMxPFK/60p2P/Piyov\
4990 N8mfM+1NjWNg1j9KqOToLVGSF2+2pRi1lgn3i3j0V7YsowVMzEuVPf1RkydfOaK2LRSB0q\
4991 zrWocCOG6gEhvgRaCj/dktj3g7dXXH4gKN6ARS0zPzYzYzYzYzYzYzYzYzYzYzYzYzYz\
4992 L66ae88pU/PN1PN1TLQJKS73dPXSR20ur7i1wPcC8QhbnNcyHu11ryyOTQVY5fvgBL7jx\
4993 +cNhj5jgRyD1JHy39o84D04H2QtX8ThaPeFIOU+wlc+KnyhK5FEVgOWGAEbX8eXMoLJ\
4994 rikbD9gHEP52VgQ14h89FUA6kYjYFbbQbnzLJg4zFiesnDHCwvUoeiVQob/5C9F9YD1UeOH\
4995 +zGhUhnSgOqrm0WgurkI9RpjBD4Y6uQcQd5TUOW63zD3MHesy14V49isbdKyxhG1CpFR\
4996 UJ6toACF7F9V58NBFDHTOMBae74Ent+eWrr+Lz/L/Q7w60AdB7QUjps/OA7COoNBNCeMUZ\
4997 ttCo/coG28f1pVKE1TPFV8juRasEahbHvxaR1guoeBpYfudo4+ofeBdybL8L2z9XeSXFAMCo\
4998 bgGgov0g1zG6Gw4fF392xnHhdc+Mwf3JTjntZ2yC1YJBJXNUT5KIKyck1sxXRdl6BmcevN\
4999 aJovy/VBacMevgEP46/21nJjt9jx17VL53Z15Mtvap1QGINHw5pDQqYwTq1Z2b8n6cMG2ZV\

```



```

5000 qOoFjSdYvV0AZz2Fayidv6FJ35CS4jXzk9hir7e27zm6p3T8hLJpkyicJpVlHtk/DJFU4Jw1\
5001 1ImhM5IR9fz2zGRKx4w/C+HQSPe+krbYrN3gEPTNahsHaLDSzXh5Q5MCoPPVdEppccqbm/8e\
5002 7\zdo0aPtag/mlk377UUVG0xybTdx/Ex/PTfa/17r7Ku+cSoiCxwroohXf16wEV9H+ccVg1\
5003 pd/CU42AK2IUP1VTKL1/sjYy5P5PvHq728NzvFuZvVODGy9GoopuuhMLNfCtX48VH4KFPuW\
5004 f/8hpXVv/43rQ9gxtg6YtcvLXDC3EmBDQn9nbf21Ji9/P7eW+ioMAoGF+N13iJ5f8dn9\
5005 hrAuc6EwDkEjUZK8EUXMAe7LzUgWCU2nb6e9N1Ji9/P7eW+ioMAoGF+N13iJ5f8dn9\
5006 WNW4Pfy9JjXuPwDl/EXNZgTaveaLDzvsHWI9mu5rvvZX9f0S4v/LfmgDEIpHdG1fM2uCW\
5007 gJiyZw0FNfz3EeaiVdZYNcNjYrNyhyGHW9mu5rvvZX9f0S4v/LfmgDEIpHdG1fM2uCW\
5008 bVlIwbfELCERF04qazRD7176/fBJkyLD59BlZ5W46wQp/7LkPBeCOpuWfTj3Osgp8GHNOZuWu\
5009 10AuywDh9zR2zDRQVQF15GvYhOvJRKAWA46PvT+RxAJVLJvYv9+CeUBmk168/rFQm\
5010 mCuifwaidFN/y18ga5wC3dkKhayvZuCYSGV/KhewFWRDKAMKcHd8EKk+HF12A9bt2d17Z\
5011 2gnOvOCpDYMfEhN70oqXDXWIKAIQ7COGchyADNnerqN5VXktc3Jdgp2OtWgmJU7+eH7\
5012 yHbUgmlX17f7k1DwaRYUEN42FluxNDVETaml6eYCR26VbbZaw2px8fmeH63EM+mgs0Lk\
5013 d3/ZvGEBLkPQWuzg1Yc5ew5/zBcy54wOgWwKfEnwbqctvEWT4FUBVov32gwe8DLzDTMaJ\
5014 aupp7/bmXx+yy/egJGkTksy2d+gFb9VoDvX5Bl2TOR+Hfjy0pP6UOXG0Nqr/gqta3vB\
5015 F6gaugvz27vnm8dV3r1dBw34GSFg910DC9h5XWkh9KaAmmyJ6dk1PzZmtD3cnU7vtw5C\
5016 h/rxG1P7Wp/VvuDuc+wsq54ymr9zzkOyRSPrAdIKoGz1i8b6ytagcEpmB9V/im09UATz\
5017 Jow6VnPkMkH3z+ennPhsCjYja6cRsrMyrGkiwF415UouliLiLRW7EmLEx3z2+/cWmlLU2\
5018 Y572b6EazkfyOvctJi15QlnyLdrFrUzj/3pmkug/Y9ga0GYMTf7neVix/6CHUghl1uh/\
5019 F9Uvo+g7G03g7zFL8xozw+/8P6W6fVXsXhInlayWdz2X1ULm/4uLmPwA05UgedoL9\
5020 ZFA6cgoxztT66Q4LR5D0j9xuvlcy+rFbcujVsnLkXvOCefphUbICLRW1+9KP4vngHfG62\
5021 NCGMSiCsnKcFexD+mtf1BwuxdmPboZqT/194225Y3TCzPQWhthG2zHraJ0/yb0kdkhpanzq\
5022 KxWF66/8Cb5AhcbzdpnhuJec6YFowlgzeMmtgNCDeKtXiVuc3Lk4YVtJepug5tWGSKkDA\
5023 ufu9MfWiG3sgntk67+3EXQWwzVeegSpvrZcm2afVSVy461+O4KvYVgicCugG2r2poyTveJ\
5024 02Um1zEJZOEO+EK70dFtNXfW2U9x70/bqZct5z0Poio+vdpyDjcdxrd34U9XcEhrloSktt3ug\
5025 AcwtK009F2Fn+gWdS60DcFodrAxnoCfRXWUSO93pBZXm7VAe+qvr506/20ALXgngLbrC\
5026 7HgRdxrEHz2H1MYVvqgm5zTTP5+7vo1RR/zJlOylx+8oH2wEbcV/OTU5ic3NGfjks3OMZ\
5027 tFtUe1lY4yFackjzxpZy6h1gJebwqLXyo09/j8k//WW3S32QPPhV5AMtPl1DFN2Op6\
5028 fz5yW4HfmdX+Buy4Nvu7yEFBOK65icot+zjP+8qf4JkyiTNKktb/gST0MKKAC18jJPL\
5029 A4PCxYmPKMOTkRy84HpyOsws/BsqyT2RGZ6zrl0gA9sBhEp46hsP2ratmOJeGrugBWD2Pw\
5030 NYD1B4OSTMmcmcd5E/GG2zvrF7Ue3sqw/7A7quEH6Ky19q3fp0QvQxtX4dz+Ueg+Lmy5\
5031 bJYtO+h5Lsgp5Nz6nbwFPhudaYemZy4ap1z5dlbYA3NQTc4F3RKYfOTKaU4P9Xy0LwU8\
5032 sdWC/H29vO0TVM1C+izhTu27rgAebk4+8H3P553qOoy/WHj21Zwbd72X2Luv41qmQSV\
5033 2GML+6KmhczwAqWgnellYz/gLLX+IBncN2PQ7F9Y5XQEN/qua+Hr3URAgG1MTRLg3bfYEtP\
5034 m6450KcorJmzX9nQ2JAggBymXSL9vzQsGfXUBjHpbXbz2+VkuEBrRIotE/Bw80gf/LIzX\
5035 /20Tdegnb681t7DtgQRE1EVT29eWTS5JF71FSz0v1YtLVqUT0b62etccBRO1He8685YeS\
5036 292UdegnmTW757ng7dKrV19rLztoMPBR73naAYrdzfm+5DzsymDymaHnClkvPOVHG5FrQ\
5037 wCY6RwU9DkxSMU9vQXMAx+ePguLw8/dvEGU1LpvsPbpXspOniQwagElsm9gNxtc0EO1vj5\
5038 7BBBjAdhkmPdx0/qirWlbf445cNKQWq7DsuJzH16CLz8bk+1u2u78FYWFKlQ4/qY2x\
5039 TyYjX8boyWm6zvc9/Ojwz7pUtVlPpNQ2UXL08PKQDmLuvootDjlyxcRNWHEHj0wYKrKPs\
5040 2JH14LpJiCQXOyp6nMs5fYsKeileOC95+WxPc3m5mcmjMe5b+lyBZYLXgJrMdnY/HMK05\
5041 aP7Md34PueUYz8WDov5zXVf/xsFe+Lpz/wjQO9eih94ZwqV562+CUHv31MtnjshfXorHf\
5042 Wkzq9FwLrTCRwJwh5+/ocSLzLzG52BvtG+wOgpxRYEwcaRfRdbSgC5bd/PySkBhakPWO\
5043 GzX9y4L1U0AB4k5we8gDh06+bn0wzFXyAUviy6Ece00117SAZkxvOqmxzB9ReavYvxx\
5044 zCMBjAdTCrWuYkrIwy4myTH9zt3R93/8X1j0ESWety7y7F1jlodwAmhFA2KD6D1wteH6\
5045 H52HwWwLaLHQH0QZv76y6znTl57rgu40YBjg4JBMJCayRnTYeX4X8/xCv+russ9L5yc50A+W\
5046 8v0w02Zxaw7ADPzCzEdpXpdsLx0dKefwEMy+j47AEaa7YxMjXm+1fZUL46ch7Oo6D/mP\
5047 Wncf9BTVKbs6z3hnxPIvm1kKhJubTFKRBqag1QCNIwubiPytyKlhwZag8YK0eMcj191y9J\
5048 PwK79U/55bk75ESKchwhj79Y35xY7qu8YspvTbgSG+55hdjN6Y586ErfyqV02LxrbmW\
5049 YwkG5S2p1IOK5dJzgs+2LB1B4Z6/g+uoae6yuyOV1jzCuocG41lxV00Yep1wulXUL4pPR\
5050 zD3GL6w1VE4J35xexPK1NLSuBb/34RcwB6JXGz6rflBBjbbJH7t1WbGDRVdb4bieXgpBMP\
5051 NQ73iqMHZ7ETHUvRxn45r8FpQWRNdqFV2zGblxEF16+rgdVL82CnTYvBdbs2Jf6pWmJP\
5052 Aw3rYbgn9qKMLmChjCnvUN5fKMRC2LbzJBk8mU55cm4x/2rLdJQzN1jKkyu01pdeccfMz\
5053 gKp/ahfXooVi+JToimZuYjn8F7QhMAMxdAaUeXK6cF075UUKygq5Oz33vW/20C7b+scH\
5054 Ltnp1tH3Yew841pGt4JWAnu7Pn5xwqjxB4IMabC3Q8rFLzPCJfTe0Sf08BnAdzSfWf1wU\
5055 nm1dJITHGN3sRt+42M5KwTsxPMe35RjTvorP3mm49VMogfP8oiD191X61dvbXmkqjvb\
5056 NfydX9m8Wim1MLKzeSL/VzQSkDPZcdycte7lq/B4XfKQaNeK3mL47+29fQL/gaT+/vrEO\
5057 gDTTY0U9UwKXUMfH9MYLzJvpxxu0fP00/pTehod/1XXGzawFuXp6G1Lz+eme2X9lbo\
5058 OUu119F0bLAKGqHafa5NVPhxjK7X0GuLOmRm+JAFefennaKzLRhZLYB5edUwK1/wD7\
5059 fd+JL72vEtDPEIqgWkzj6zEP/d5duztZHXhXfKlnhs7umT011AjkyVScenpJ1WA1AACzAE\
5060 qdZSx/S+nLNd0PdlXVd/SKUr+JL579vshL75z+bvNS8Q2EuQn/Oa3x1/FJZS/VZ30EGCB\
5061 ePdTCRCROKCR3q6Vl0pof7EXvDAAzVeGjOECZ56CYcmxZ/7CyuWar2IIN2x4NOC075/\
5062 4YTRK3XuwYfGjgmxT/xdbpt8uSR17f11luoFJtQm3U17cKXfygWvFdvppV9R9Aeh07FRV\
5063 hUL4693pwu1Yn+FXOC+Cy0vRWXzyh/w3n7fiiibreUtTsvURMitjpkWRWmPKKzZmDzFCiM\
5064 dm1f6+e+10/651MmCDD2YFE12dFycqj38aAbQSPGX1sCGUCaKRDOUysauzqz6z2vZ6\
5065 LLGqLXPjFyIthCkphr+cN+76L0JLd3d451+sndV9Yr4veCWg9+Srtx6G/areZLB4WX\
5066 tgzv7Wk4n+Z8f/FfzUKIa3ky5ULmo9CE8N3HgLinI5ISERNy32hsXoRnTBMbWmI9zT7o3\
5067 j0g8vnn35zecGfYgCmlw2/fvicJoJxYt:ieoL10xvRGHMynZ1/IJtL6w3j5y8+711dy057\
5068 XLJdJm+X0FGotrucgEUTDVPfEnovWAE2KAeWArG5T3tBJQOT+5rCIU+U1BzXPjPumpRV\
5069 4YUe29wP9Xlfv/oppuyDp9nUPyih91/XXNOvNSd5dGG88wms31CzfrkCQUTCZSHj+vm8Q\
5070 JV7XE3Xm6wqjLSrFLVB668ToExtHjJ4Cdw24+uzFvsJrT11RkFOoALtZPFz2D12QrQ\
5071 8Y588pdsboVhRLQD6exvrE0j9y4gDQPk5Zm3jy021LdV7y3zfl8qmsDmOPARTWFC3I1\
5072 N1NQGWk1jEavqOmz78D2ZVeFmHcdPFCU86nfbFB5:KF1EPMRHE6FOOS0Aotvm/d8vW8km7D\
5073 58YrseFULvslpXb7z64erdZNYuNLKileJdaUak7j0orr315x+YA9CbQBF/cK7QKHdB\
5074 E5eg69OKMH9pdrJd6v3gEvYhdQucslVM9no/QaPP3K21ve8zWcmJjk30kx+30RK08BK1wN\
5075 blXahe29JqBL80f8Gkam6n5P9mdG5PmUlKpme22R7BHSKjPj0kMctCt/KAM180JtXeJK\
5076 v7q/OzmZbn/Z5IOHT3+NpGzn2eyx7uiZ0JDM9xoyTcZBTOya+vndqW3URP1jyxwMoe1\
5077 zq4BzYggsLmpHdLIXkcmLwXskzBGwa940stveb+f7141iK3oi05Mxod+r9/12v80P9E3\
5078 xp7QYUu8JTGmcatO+NeY/99v3bh+21bh03cnot1jdfCZnzkeapsDN/vjg4XP4Cb8+n9p\
5079 9zduKz2Q3fev05lytqto030pzK9E5shOY+FEV150r6x5HkDFMGaAdKQ3yA09dydfdj\
5080 ppf5kjq6grny13DfyKI5h14o0K1zazehBJNtWTFBAGvV1IawS2vTahfES05drepEaP\
5081 mR1fLX0m8Xmnp/fBy6av27y5SkWno2mPpSF3sgCf3o4UGGSj/wI548wVlfbvVab720b\
5082 Xx/MwrgLf9zXPQMmX5CiAfjhIYXjhsR7BKkMFg8mLT+D3cdJF2qod1vnn3v3d60xwY7\
5083 koSvF0EpKzFegJWQ1d70c6dnp1h7i20z933hOLHWYUu1RrhZ7ptxeV69XW+3Jdsam6tO\
5084 IEmsY1G5j8EajZNR0adga7IevOR2LBSccVC8Z0u5Ue1JbspvqHecusjRkYkLW0V501tmrW\
5085 LaycFkHswTKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5086 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5087 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5088 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5089 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5090 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5091 m38w0ncAAAAASUVRK5CYII=";
5092
5093 GshIcon="data:image/png;base64,\
5094 iVBoRw0KGgoAAANSUHEUGAAKWAAB/CAYAAABymy1ZAAAAAXNSR0IARs4c6QAAAH1WE1m\
5095 TUOARgAAAABAAEAAUAAAABAAAAPAgEBAUAAAABAAAARgEoAAMAAAABAAIAAAIpaQAAAAB\
5096 AAAATgAAAAAAAABIAAAAQAAAABAAAABAAQAAQADAAAQAAAQAAAACAgEAAAQAAAQAAKygAwAE\
5097 AAAAQAHAH8AAAACt6zAAAA1wSF1zAAALEWAACMBAJcGAADQRJREFUEAhtn09wFNu0\
5098 x9/b21+zYCKCIk1amW1j/jh6BckstFEFth1IqPrWdstQoqEunttW2FnQ0Y1TIatin20\
5099 amdaqY6jYIOXI7kgglarVv74b3BAQPKbVAJj3e3r94WcJpe93csmcbj784kd/ve7723+3nF\
5100 ffv+nx88SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
5101 SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESMCCGLR84/Tylk/EYasHcAMHXRK\
5102 XiEuf52tAHwEad81mP2wLTxTadyBmzrT+42pzRsrD3peQvpXsMtrhgNYCn8fewHfUap+zyH\
5103 ZUASIAESIAESOKII+LPR9dzsk5ELVxdKBtZ1hpQokIEfE+8Jan8AzkmYA/67BKX1iek+pmQ\
5104 hs7VveE6PyDz+oM6JmAxwzzn6REVcG4SQXwh1L18XtFFcuWqH7AMRgIKQAIKQAIHIAH/\
5105 Nbgem30981H00S8sa50440umz+EzEAE/FWHXfnjsofLD/YP80ZNUKRALVAWLqEGG4C/BGSE\
5106 rgK0Amb/d3ucCJ1WJsIyVnsIyOKAQ8FevYNmsYJYR5F3cqmUmL6v/FwDD1QAV457EBPzYQ\
5107 65pV5cdz246nqHyziLzEdt1K7M51Ayw2yWtMxNda8LCpLj1Ww0i0U1/3d05692zGp\
5108 2D8HBZtDxp+28XKicYgq9Fve6Eh5QVCinOVeQokFL1ka7gpKvWbUnKnFoods814tKyaQB\
5109 e0Lc2e8+3+r1p162LEVYNps6SQfapwSgmw0Kf8upDNKzLsBawPFDureUtyYUePwH9e\
5110 faWf9i90Qt4Cz7gYoroBFV9CrE/2qnEo/ElFa4DQdHalosCUT4rpJzGLGN56Z10QVRtE\
5111 rHxDxjvvnShmYsWPTq6UEZEGJES2Ewmpj4sk8jSVQat/zSeLUz5aemLH2iRVkdhpAVH0\
5112 FR5zaZf851OgmVortLSeXG/otLBR9s+r5h8u0ihusYfZ19fGlP2cNSyt1IA2//w008aEK\
5113 IX87zhymPTktb3oc1bXka/DKX3bYpoeHh686jgQSEXcUgvXALY+CVNO808Mmi9BU0OH+oH\
5114 zFN7phGbb3k0oJ1FO9zr7rGVn3d1QNRtg4570TS1hkYjsUok6478h1PhfRo614d4mU7N\
5115 4t2xw1BPIU1BE60Gw2+T9JpFNkn7Wubrmu5WgppjG7K13801ulcApeIwCLADauggEoc1Ys4\
5116 1B7b03kBUet04panzx0+9yONN9ANsdFQmN4oxSnokzgtN+f0CANUSNm3unJyAGvHzsuc6+\
5117 CGJp2DufdMGRf0n8BezIjXdyEzHa+vntqfDT10QH1kcvCeJ9j5VjebVqEtoSfV1/ERXV9E\
5118 74rav8+EyC/w6f7Xamno5Kqnr60Ylem/+GgTOA6dKXNTz8wCCamh8Mobur8I5bb1bk66C\
5119 RbHvm2brM7h195Jv11hSpPwyEn9XhL6Jne71K1+UwQNK2hcmG5M6VzJwULY1T9tUte+E\
5120 S0ngishHovWt/2fW5q6vHwVvYt4r/QkuK2WP20giXhfsNqYqgYz52bBav0/PdhDX8Pw\
5121 1fhpLH6f7fbbP7VC6fKUXFwHs0ZYSjze1TK3T7jWR1jeer0Ht81rJCXhIU09BNI1fVgkC\
5122 5w2uDRZKXn211qV08pLmxu0BcaZpaVgVJed1ZO+SUopx3S11+3idu2NxnecDUVAJk04gkC\
5123 Gpge4duA7r/74/WPD77y76+A3c574a/FyENPbyb9Yb/cvZPn30yFrtv3PCJpJ0FAKqAet3\
5124 7T4w6c7EqpkHyoeAeRDXFG7FWKkhh72w453a+vBKSbw1La7r0hPRM9S9otYqd1vQtdf2/1\

```

```

5125 Tr/7DA8W5jK8WHVLRkfuMts4CsZRV41LA+Y8t/zDLl0nowlVcTe7wfdHK3+TazxSTjKLi2K6\
5126 C8zVlgGfHksRneUxngQ9UBVok390MFfZTUYZA8prA05RfneJKA0/huOtNw+cc5y9264CLN\
5127 TyPH0lR+3pl9VWMIIdCp6p1LTstasnpJrcQ+BiH0q9kKgnrXmH4dRfneZuJfavkKtINQRzX\
5128 eQd16tYHZX6MCWt2lh9JeF9+0/wj2AjzQ+hFTPFKytZogIpuvsrX70zOxw0bWlyJ0eP8EN3\
5129 cSfWvFsi3jRX+KH17ky7CCT+GEao:st0dzvhgHl/O9rVwtaHpulzsyOkf99UCZDBLz1lH0gTf\
5130 zyDwtdlSVH9FEdt1hlfOgMKMF/29W2qY7k7zDUuzlbutncUDLMkyKR60OL450y2RSIuy8F\
5131 213vcxG8egY2Df3BH6gAT7f3yc0VXRidIomX/886DlMVSBlTZPt55DnaCmXwpHmaF0mmfM\
5132 vH9sgEnbQ/DSrW0YsRwB1Q34B0x/SSYpD73Wk9owbTbPezP++jFTSogyoA6xR/ofj5QrDY\
5133 BnaeW4zhz+2rDY5qZQAVdp5We1t/G0SumZY6HgmGfPRJo/y4aaU+7cfFy+L50KkKwC\
5134 AjzxWwCLD+BYJ07R86THuc8jclEpWZZ5qZ4P89K09H9p55283TmJnu8zUPTN+OL/PC\
5135 dc2P4UFahmsfn47H6RP12VnwjzZLuflwSLBs0YF72KosQJyJzn2FL00acKAG4U6b8+BXyQ\
5136 TkKvWenYqepTgZ2fH6ghg26/jB8aPKnoB59jZL9L+084E59USUQhki65Vw6P3njyDW\
5137 85ziR500lqAbRR6Ts0+rZbMQXwv2xr0csSSmQl/fCFY7LPdz21JzrSKK+C5dELh6ixYITwL\
5138 V1/nm4/cmbCw+XmNWee48EznelWaoF+EKYUroI1D0GpL3PawpZRGFPInIhtCOXYQ5CLQPW\
5139 RCJ4fb1+LEuV3VncC8BzF4KvIsze2fVNVs6qjyQv++Y0t29BUzqWjrjgWZD1oBJWxe18Vix\
5140 KEPL9fdsBxp/2X6sgXKM3dfClatfd8adBN1U0Unh1AEwg6Bw93sevqjIHMULPW/b14a6npq\
5141 pkSWlwr06FYmV3MLU6MwfbD3KvyWtXwJ2zUc04jSj+6WUYjPTLlPSPV1okE327S/NdWx\
5142 NgJ+21W6VtW1T4TY/bUnDxmS71u9baqa20YX5DbUX1z9BRpGEvdrDHJ51k3m3z394VgdSyp\
5143 qZbnk1kQbbVhbtteH61/0vu/ZgszafEIR+tNOBCXy90Xa7g7BBtQ6tbu/vOYiPhu8xhz24R7\
5144 c1MaOu3Q4pYZWHWwCt1a17ndi3bXo2P7v2p70cmmEpycew8L4Q6770Ev+htZPNED+mNFy/W2\
5145 9LRARH5EJ/vQ5f1lW7StTREMd4Au5+Q3T6aRSqdmk7Z+/GB47ui290Uw9JX4AnJ711IS\
5146 16Y+xi8Sfm6YcrXul4K1ysH6Be9110YHs791/4cxvbnH2jWB1j1XXxecYQuZU0g5WDUoq\
5147 Y6MFg2XReb6wTrTjP5NO7ZuP3v9IrmDnn4F5eSKoH0tab6aStQot7/beUGSubPmhrC27B1\
5148 0YqS10JvclK1Yt4XkKoz+kgw+oaJdVesgVer9PehP+SrXwnkMNLm6VpOnUikXlzm+PveGof\
5149 h4F8J1j9WOrt+64Stf500WEZd2G5tCd/FZS/VXH3nagrQUL+4B2j8m8Ss/Fmi9D3M3eJwo\
5150 Krn3KXzuzqzpsZKZU1cbOCRjMPhQ1abXm1gsdui135d3JlR9yw0Vm9Np1kTie/CQYIdWZV2\
5151 8/KpxqnKkvclbdeIDDt0Usc+RmxmDipnxmTw78tYm6HGDc2f2gZnJ+8xzSRBvO4zrhEw9\
5152 H26s89VJSOHZrR117AAPQwzKI7LepLurBj0Qpxybyb/8dmn2//1l/qgnago2Awgf/38+WEL\
5153 I4af+5Q5EXMARKAoI2CCP2xvJNV+LMZ78LkH3V27LWv2n9w4/+63gdKxJPLq7b1TDkdw1p\
5154 nIhS+QSI+HiEWsRPuVengV20d6nf7K0tloP1dJ/iKUsatCEBEIABEiABEiABEiABEiABE\
5155 EIABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5156 EIABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5157 EIABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5158 EIABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5159 EIABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5160
5161 ITSmoreQR="data:image/png;base64,\
5162 iVBORw0KGgoAAANSUgEuHAAAGSAAABvAQMAADYCWwJAAAAB1BMVEX///9BaePhqDaJAAAB\
5163 hK1EQVQ4jdxTsa2EMawGYCMX7sICKVqjXVaCBe7CarASXda1LAWgS4HwM5zEVS+mvSgS+ZBQ\
5164 8gcB4BHyZw8zMSaUBHNM+KAd4QCLDpDn8ogT4UpPGci2jJ8IGF3eLWpWahWnyVeWec\
5165 UEBDXaB0X2aJueYDOzNk1QassPckjcn4W3E1SfwqYk6jv/vAKPhg0AlSfHve8Jt0dkwDMwr\
5166 YMSGSuPWHAr19k0tkV2sb3sdm2rUCw88g4Rp1A9s1JPv9cTPlNRD4Xfkin8XaQCITw6Lzq\
5167 Z08dhw/4+U2GzclS8gbqVmkfr1N6YX80qLD00mlCTWvzPERA8AL9vvoIfoSpL33fsvytrL\
5168 S9wiqDznhUI38v5N73/gBuUs2eLgic8GAAAABJRu5ErkJggg==";
5169
5170 </script>
5171
5172 <script id="gsh-script">
5173 //document.getElementById('gsh-iconurl').href = GshIcon
5174 //document.getElementById('gsh-iconurl').href = GshLogo
5175 document.getElementById('gsh-iconurl').href = ITSmoreQR
5176
5177 // id of GShell HTML elemets
5178 var E_BANNER = "gsh-banner" // banner element in HTML
5179 var E_FOOTER = "gsh-footer" // footer element in HTML
5180 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
5181 var E_GOCODE = "gsh-gocode" // Golang code of GShell
5182 var E_TODO = "gsh-todo" // TODO of GShell
5183 var E_DICT = "gsh-dict" // Dictionary of GShell
5184
5185 function bannerElem(){ return document.getElementById(E_BANNER); }
5186 function bannerStyleFunc(){ return bannerElem().style; }
5187 var bannerStyle = bannerStyleFunc()
5188 bannerStyle.backgroundImage = "url("+GshLogo+")";
5189
5190 function footerElem(){ return document.getElementById(E_FOOTER); }
5191 function footerStyle(){ return footerElem().style; }
5192 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
5193 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
5194
5195 function html_fold(e){
5196 if ( e.innerHTML == "Fold" ){
5197 e.innerHTML = "Unfold"
5198 document.getElementById('gsh-menu-exit').innerHTML=""
5199 document.getElementById('gsh-statement').open=false
5200 document.getElementById('html-src').open=false
5201 document.getElementById(E_GINDEX).open=false
5202 document.getElementById(E_GOCODE).open=false
5203 document.getElementById(E_TODO).open=false
5204 document.getElementById('references').open=false
5205 }else{
5206 e.innerHTML = "Fold"
5207 document.getElementById('gsh-statement').open=true
5208 document.getElementById(E_GINDEX).open=true
5209 document.getElementById(E_GOCODE).open=true
5210 document.getElementById(E_TODO).open=true
5211 document.getElementById('references').open=true
5212 }
5213 }
5214 function html_pure(e){
5215 if ( e.innerHTML == "Pure" ){
5216 document.getElementById('gsh').style.display=true
5217 //document.style.display = false
5218 e.innerHTML = "Unpure"
5219 }else{
5220 document.getElementById('gsh').style.display=false
5221 //document.style.display = true
5222 e.innerHTML = "Pure"
5223 }
5224 }
5225
5226 var bannerIsStopping = false
5227 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5228 function shiftBG(){
5229 bannerIsStopping = !bannerIsStopping
5230 bannerStyle.backgroundPosition = "0 0";
5231 }
5232 // status should be inherited on Window Fork(), so use the status in DOM
5233 function html_stop(e,toggle){
5234 if ( toggle ){
5235 if ( e.innerHTML == "Stop" ){
5236 bannerIsStopping = true
5237 e.innerHTML = "Start"
5238 }else{
5239 bannerIsStopping = false
5240 e.innerHTML = "Stop"
5241 }
5242 }else{
5243 // update JavaScript variable from DOM status
5244 if ( e.innerHTML == "Stop" ){ // shown if it's running
5245 bannerIsStopping = false
5246 }else{
5247 bannerIsStopping = true
5248 }
5249 }

```

```

5250 }
5251 html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
5252 //html_stop(bannerElem(),false) // onInit.
5253
5254 //https://www.w3schools.com/jsref/met_win_setinterval.asp
5255 function shiftBanner(){
5256     var now = new Date().getTime();
5257     //console.log("now="+now%10)
5258     if( !bannerIsStopping ){
5259         bannerStyle.backgroundColor = ((now/10)%100000)+" 0";
5260     }
5261 }
5262 setInterval(shiftBanner,10); // onInit.
5263
5264 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
5265 // from embedded html to standalone page
5266 var MyChildren = 0
5267 function html_fork(){
5268     MyChildren += 1
5269     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5270     newwin = window.open("",WinId,"");
5271     src = document.getElementById("gsh");
5272     newwin.document.write("<"+<"html">\n");
5273     newwin.document.write("<"+<"span id=\"gsh\">");
5274     newwin.document.write(src.innerHTML);
5275     newwin.document.write("<"+<"span"><"html">\n"); // gsh span
5276     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5277     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5278     newwin.document.close();
5279     newwin.focus();
5280 }
5281 function html_close(){
5282     window.close()
5283 }
5284 function win_jump(win){
5285     //win = window.top;
5286     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
5287     if( win == null ){
5288         console.log("jump to window.opener("+win+") (Error)\n")
5289     }else{
5290         console.log("jump to window.opener("+win+")\n")
5291         win.focus();
5292     }
5293 }
5294
5295 // source code view
5296 function frame_close(){
5297     srcframe = document.getElementById("src-frame");
5298     srcframe.innterHTML = "";
5299     //srcframe.style.cols = 1;
5300     srcframe.style.rows = 1;
5301     srcframe.style.height = 0;
5302     srcframe.style.display = false;
5303     src = document.getElementById("src-frame-textarea");
5304     src.innerHTML = ""
5305     //src.cols = 0
5306     src.rows = 0
5307     src.display = false
5308     //alert("--closed--")
5309 }
5310 //<!-- | <span onclick="html_view();">Source</span> -->
5311 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
5312 //<!-- | <span>Download</span> -->
5313 function frame_open(){
5314     oldsrc = document.getElementById("GENSRC");
5315     if( oldsrc != null ){
5316         //alert("--I--(erasing old text)")
5317         oldsrc.innterHTML = "";
5318         return
5319     }else{
5320         //alert("--I--(no old text)")
5321     }
5322     banner = document.getElementById('gsh-banner').style.backgroundImage;
5323     footer = document.getElementById('gsh-footer').style.backgroundImage;
5324     document.getElementById('gsh-banner').style.backgroundImage = "";
5325     document.getElementById('gsh-banner').style.backgroundColor = "";
5326     document.getElementById('gsh-footer').style.backgroundImage = "";
5327
5328     src = document.getElementById("gsh");
5329     srcframe = document.getElementById("src-frame");
5330     srcframe.innerHTML = ""
5331     + "<"+<"cite id=\"GENSRC\">\n"
5332     + "<"+<"style">\n"
5333     + "#GENSRC textarea{tab-size:4;}\n"
5334     + "#GENSRC textarea{-o-tab-size:4;}\n"
5335     + "#GENSRC textarea{-moz-tab-size:4;}\n"
5336     + "#GENSRC textarea{spellcheck:false;}\n"
5337     + "<"+<"style">\n"
5338     + "<"+<"textarea id=\"src-frame-textarea\" cols=100 rows=20 class=\"gsh-code\">"
5339     + "/<"+<"html">\n" // lost preamble text
5340     + "<"+<"span id=\"gsh\">" // lost preamble text
5341     + src.innerHTML
5342     + "<"+<"span"><"html">\n" // lost trail text
5343     + "<"+<"textarea">\n"
5344     + "<"+<"cite><!-- GENSRC -->\n";
5345
5346     //srcframe.style.cols = 80;
5347     //srcframe.style.rows = 80;
5348
5349     document.getElementById('gsh-banner').style.backgroundImage = banner;
5350     document.getElementById('gsh-footer').style.backgroundImage = footer;
5351 }
5352 function fill_CSSView(){
5353     part = document.getElementById('gsh-style-def')
5354     view = document.getElementById('gsh-style-view')
5355     view.innerHTML = ""
5356     + "<"+<"textarea cols=100 rows=20 class=\"gsh-code\">"
5357     + part.innerHTML
5358     + "<"+<"textarea"
5359 }
5360 function fill_JavaScriptView(){
5361     jspart = document.getElementById('gsh-script')
5362     view = document.getElementById('gsh-script-view')
5363     view.innerHTML = ""
5364     + "<"+<"textarea cols=100 rows=20 class=\"gsh-code\">"
5365     + jspart.innerHTML
5366     + "<"+<"textarea"
5367 }
5368 function fill_DataView(){
5369     part = document.getElementById('gsh-data')
5370     view = document.getElementById('gsh-data-view')
5371     view.innerHTML = ""
5372     + "<"+<"textarea cols=100 rows=20 class=\"gsh-code\">"
5373     + part.innerHTML
5374     + "<"+<"textarea"

```

```
5375 }
5376 function jumpto_StyleView(){
5377     jsview = document.getElementById('html-src')
5378     jsview.open = true
5379     jsview = document.getElementById('gsh-style-frame')
5380     jsview.open = true
5381     fill_CSSView()
5382 }
5383 function jumpto_JavaScriptView(){
5384     jsview = document.getElementById('html-src')
5385     jsview.open = true
5386     jsview = document.getElementById('gsh-script-frame')
5387     jsview.open = true
5388     fill_JavaScriptView()
5389 }
5390 function jumpto_DataView(){
5391     jsview = document.getElementById('html-src')
5392     jsview.open = true
5393     jsview = document.getElementById('gsh-data-frame')
5394     jsview.open = true
5395     fill_DataView()
5396 }
5397 function jumpto_WholeView(){
5398     jsview = document.getElementById('html-src')
5399     jsview.open = true
5400     jsview = document.getElementById('gsh-whole-view')
5401     jsview.open = true
5402     frame_open()
5403 }
5404 function html_view(){
5405     html_stop();
5406
5407     banner = document.getElementById('gsh-banner').style.backgroundImage;
5408     footer = document.getElementById('gsh-footer').style.backgroundImage;
5409     document.getElementById('gsh-banner').style.backgroundImage = "";
5410     document.getElementById('gsh-banner').style.backgroundPosition = "";
5411     document.getElementById('gsh-footer').style.backgroundImage = "";
5412
5413     //srcwin = window.open("", "CodeView2", "");
5414     srcwin = window.open("", "", "");
5415     srcwin.document.write("<span id='gsh'>\n");
5416
5417     src = document.getElementById("gsh");
5418     srcwin.document.write("<"+style>\n");
5419     srcwin.document.write("textareat{tab-size:4;}\n");
5420     srcwin.document.write("textareat{-o-tab-size:4;}\n");
5421     srcwin.document.write("textareat{-moz-tab-size:4;}\n");
5422     srcwin.document.write("</style>\n");
5423     srcwin.document.write("<h2>\n");
5424     srcwin.document.write("<"+span onclick='window.close();>Close</span> | \n");
5425     //srcwin.document.write("<"+span onclick='html_stop();>Run</span>\n");
5426     srcwin.document.write("</h2>\n");
5427     srcwin.document.write("<textarea id='gsh-src-src' cols=100 rows=60>");
5428     srcwin.document.write("/<"+html>\n");
5429     srcwin.document.write("<"+span id='gsh'>");
5430     srcwin.document.write(src.innerHTML);
5431     srcwin.document.write("<"+/span><"+/html>\n");
5432     srcwin.document.write("</"+textareat>\n");
5433
5434     document.getElementById('gsh-banner').style.backgroundImage = banner;
5435     document.getElementById('gsh-footer').style.backgroundImage = footer
5436
5437     sty = document.getElementById("gsh-style-def");
5438     srcwin.document.write("<"+style>\n");
5439     srcwin.document.write(sty.innerHTML);
5440     srcwin.document.write("<"+/style>\n");
5441
5442     run = document.getElementById("gsh-script");
5443     srcwin.document.write("<"+script>\n");
5444     srcwin.document.write(run.innerHTML);
5445     srcwin.document.write("<"+/script>\n");
5446
5447     srcwin.document.write("<"+/span><"+/html>\n"); // gsh span
5448     srcwin.document.close();
5449     srcwin.focus();
5450 }
5451 </script>
5452 -->
5453 *//<br></span></details></html>
5454
```