

```

1 //<html><details open><summary>GShell-0.2.7-HtmlArchive</summary>
2 /*<span id="gsh">
3 <meta charset="UTF-8">
4 <meta name="viewport" content="width=device-width, initial-scale=1.0">
5 <link rel="icon" id="gsh-iconurl" href=""><!-- place holder -->
6 <title>GShell-0.2.4 by SatoxITS</title>
7 <header id="gsh-banner" height="100px" onclick="shiftBG();" style="">
8 <div align="right"><note>GShell version 0.2.7 // 2020-08-31 // 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">
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 = "0.2.7"
129     DATE = "2020-08-31"
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 == '@' || ext == 'd' {
368                 // !N@ .. 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     }
378 }

```

```

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 // lnspl [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-- lnspl %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-- lnspl %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 // rewrite contents if specified
755 func (gsh*GshContext)XGrep(path string, rexpv[]string)(int){
756     file, err := os.OpenFile(path,os.O_RDONLY,0)
757     if err != nil {
758         fmt.Printf("--E-- grep %v (%v)\n",path,err)
759         return -1
760     }
761     defer file.Close()
762     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path, rexpv) }
763     //reader := bufio.NewReaderSize(file,LINESIZE)
764     reader := bufio.NewReaderSize(file,80)
765     li := 0
766     found := 0
767     for li = 0; ; li++ {
768         line, err := reader.ReadString('\n')
769         if len(line) <= 0 {
770             break
771         }
772         if 150 < len(line) {
773             // maybe binary
774             break;
775         }
776         if err != nil {
777             break
778         }
779         if 0 <= strings.Index(string(line),rexpv[0]) {
780             found += 1
781             fmt.Printf("%s:%d: %s",path,li,line)
782         }
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
816     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
817         dir,
818         fusage.Files,
819         fusage.Dirs,
820         fusage.SymLink,
821         fusage.HLinks,
822         float64(fusage.Size)/1000000.0,bsume);
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("(%8dBlk) %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         abtime(nanos),
1132         abtime(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 [jos.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(argl[0:ax]))
1377         if debug {
1378             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1379                 -1+len(argv), sb, si, string(argl[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("[%d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1563     }else
1564     if 1000000 < elps {
1565         return fmt.Sprintf("[%d.%03dms]",elps/1000000,(elps%1000000)/1000)
1566     }else{
1567         return fmt.Sprintf("[%d.%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)/1000000)
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 absi(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             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1852                 fname,local,err)
1853         }
1854         fmt.Printf(Elapsed(Start)+"--In- PUT %v (/%)\n",dsize,bsize)
1855         fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1856         fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1857         fileRelay("RecvPUT",clnt,out,dsize,bsize)
1858         res = fmt.Sprintf("200 PUT done\r\n")
1859     default:
1860         res = fmt.Sprintf("400 What? %v",req)
1861     }
1862     swcc,serr := clnt.Write([]byte(res))
1863     if serr != nil {
1864         fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1865     }else{
1866         fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1867     }
1868     aconn.Close();
1869     clnt.Close();
1870 }
1871 sconn.Close();
1872 }
1873 func (gsh*GshContext)RexecClient(argv []string)(int,string){
1874     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("(%d)%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[0: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 res := ""
2030 if count == 0 {
2031     res = "(nil)\r\n"
2032 }else{
2033     res = string(res[:count])
2034 }
2035 if err != nil {
2036     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,res)
2037 }else{
2038     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
2039 }
2040 serv.Close()
2041 //conn.Close()
2042
2043 var stat string
2044 var rcode int
2045 fmt.Sscanf(res,"%d %s",&rcode,&stat)
2046 //fmt.Printf("--D--- Client: %v (%v)",rcode,stat)
2047 return rcode,res
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("Uar=%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     }
2657     }
2658     }
2659     }
2660     }
2661     }
2662     }
2663     }
2664     }
2665     }
2666     }
2667     }
2668     }
2669     }
2670     }
2671     }
2672     }
2673     }
2674     }
2675     }
2676     }
2677     }
2678     }
2679     }
2680     }
2681     }
2682     }
2683     }
2684     }
2685     }
2686     }
2687     }
2688     }
2689     }
2690     }
2691     }
2692     }
2693     }
2694     }
2695     }
2696     }
2697     }
2698     }
2699     }
2700     }
2701     }
2702     }
2703     }
2704     }
2705     }
2706     }
2707     }
2708     }
2709     }
2710     }
2711     }
2712     }
2713     }
2714     }
2715     }
2716     }
2717     }
2718     }
2719     }
2720     }
2721     }
2722     }
2723     }
2724     }
2725     }
2726     }
2727     }
2728     }
2729     }
2730     }
2731     }
2732     }
2733     }
2734     }
2735     }
2736     }
2737     }
2738     }
2739     }
2740     }
2741     }
2742     }
2743     }
2744     }
2745     }
2746     }
2747     }
2748     }
2749     }

```

```

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[j]string){
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[j]string,optv[j]string,list[l]string){
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[j]string){
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([]string,8)
2982         n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
2983         fmt.Printf("--D-- Scanf ->(%v) n=%d err=(%v)\n",v,n,err)
2984         gsh.iValues = v
2985     }
2986 }
2987 func (gsh*GshContext)Printv(argv[j]string){
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.printf(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" || cmd == "d":
3133         xDic(argv)
3134 case cmd == "dump":
3135     gshCtx.Dump(argv)
3136 case cmd == "echo" || cmd == "e":
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(argv)
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 == "genq":
3169     gshCtx.Enq(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" || cmd == "nlsp":
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 == "rexecd" || cmd == "rexd":
3196     gshCtx.RexecServer(argv)
3197 case cmd == "rexec" || cmd == "rex":
3198     gshCtx.RexecClient(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.Rexec(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         }else{
3289             //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3290             // stat.Rdev%0xFF,stat.Rdev);
3291             //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev%0xFF);
3292             return int(stat.Rdev & 0xFF)
3293         }
3294     }
3295     return 0
3296 }
3297 func (gshCtx *GshContext) ttyfile() string {
3298     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3299     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3300         fmt.Sprintf("%02d",gshCtx.TerminalId)
3301         //strconv.Itoa(gshCtx.TerminalId)
3302     //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
3303     return ttyfile
3304 }
3305 func (gshCtx *GshContext) ttyline>(*os.File){
3306     file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3307     if err != nil {
3308         fmt.Printf("--F-- cannot open %s (%s)\n",gshCtx.ttyfile(),err)
3309         return file;
3310     }
3311     return file
3312 }
3313 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3314     if( skipping ){
3315         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3316         line, _, _ := reader.ReadLine()
3317         return string(line)
3318     }else
3319     if true {
3320         return xgetline(hix,prevline,gshCtx)
3321     }
3322     /*
3323     else
3324     if( with_exgetline && gshCtx.GetLine != "" ){
3325         //var xhix int64 = int64(hix); // cast
3326         newenv := os.Environ()
3327         newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3328
3329         tty := gshCtx.ttyline()
3330         tty.WriteString(prevline)
3331         Pa := os.ProcAttr {
3332             "", // start dir
3333             newenv, //os.Environ(),
3334             []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3335             nil,
3336         }
3337         //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3338         proc, err := os.StartProcess(gsh_getlinev[0],[string{"getline","getline"},&Pa)
3339         if err != nil {
3340             fmt.Printf("--F-- getline process error (%v)\n",err)
3341             // for ; ; {
3342             return "exit (getline program failed)"
3343         }
3344         //stat, err := proc.Wait()
3345         proc.Wait()
3346         buff := make([]byte,LINESIZE)
3347         count, err := tty.Read(buff)
3348         //_, err = tty.Read(buff)
3349         //fmt.Printf("--D-- getline (%d)\n",count)
3350         if err != nil {
3351             if ! (count == 0) { // && err.String() == "EOF" } {
3352                 fmt.Printf("--E-- getline error (%s)\n",err)
3353             }
3354         }else{
3355             //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
3356         }
3357         tty.Close()
3358         gline := string(buff[0:count])
3359         return gline
3360     }else
3361     /*
3362     {
3363         // if isatty {
3364         //     fmt.Printf("!%d",hix)
3365         //     fmt.Print(PROMPT)
3366         // }
3367         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3368         line, _, _ := reader.ReadLine()
3369         return string(line)
3370     }
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 // C language compatibility functions
3392 var errno = 0
3393 var stdin *os.File = os.Stdin
3394 var stdout *os.File = os.Stdout
3395 var stderr *os.File = os.Stderr
3396 var EOF = -1
3397 var NULL = 0
3398 type FILE os.File
3399 type StrBuff []byte
3400 var NULL_FP *os.File = nil
3401 var NULLSP = 0
3402 //var LINESIZE = 1024
3403
3404 func system(cmdstr string)(int){
3405     PA := syscall.ProcAttr {
3406         "", // the starting directory
3407         os.Environ(),
3408         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3409         nil,
3410     }
3411     argv := strings.Split(cmdstr, " ")
3412     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3413     if( err != nil ){
3414         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3415     }
3416     syscall.Wait4(pid,nil,0,nil)
3417
3418     /*
3419     argv := strings.Split(cmdstr, " ")
3420     fmt.Fprintf(os.Stderr, "--I-- system(%v)\n", argv)
3421     //cmd := exec.Command(argv[0],...)
3422     cmd := exec.Command(argv[0],argv[1],argv[2])
3423     cmd.Stdin = strings.NewReader("output of system")
3424     var out bytes.Buffer
3425     cmd.Stdout = &out
3426     var serr bytes.Buffer
3427     cmd.Stderr = &serr
3428     err := cmd.Run()
3429     if err != nil {
3430         fmt.Fprintf(os.Stderr, "--E-- system(%v)err(%v)\n", argv, err)
3431         fmt.Printf("ERR:%s\n",serr.String())
3432     }else{
3433         fmt.Printf("%s",out.String())
3434     }
3435     */
3436     return 0
3437 }
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
3449 func getenv(name string)(string){
3450     val,got := os.LookupEnv(name)
3451     if got {
3452         return val
3453     }else{
3454         return "?"
3455     }
3456 }
3457
3458 func strcpy(dst StrBuff, src string){
3459     var i int
3460     srcb := []byte(src)
3461     for i = 0; i < len(src) && srcb[i] != 0; i++ {
3462         dst[i] = srcb[i]
3463     }
3464     dst[i] = 0
3465 }
3466
3467 func xstrcpy(dst StrBuff, src StrBuff){
3468     dst = src
3469 }
3470
3471 func strcat(dst StrBuff, src StrBuff){
3472     dst = append(dst,src...)
3473 }
3474
3475 func strdup(str StrBuff)(string){
3476     return string(str[0:strlen(str)])
3477 }
3478
3479 func sstrlen(str string)(int){
3480     return len(str)
3481 }
3482
3483 func strlen(str StrBuff)(int){
3484     var i int
3485     for i = 0; i < len(str) && str[i] != 0; i++ {
3486     }
3487     return i
3488 }
3489
3490 func sizeof(data StrBuff)(int){
3491     return len(data)
3492 }
3493
3494 func isatty(fd int)(ret int){
3495     return 1
3496 }
3497
3498 func fopen(file string,mode string)(fp*os.File){
3499     if mode == "r" {
3500         fp,err := os.Open(file)
3501         if( err != nil ){
3502             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
3503             return NULL_FP;
3504         }
3505         return fp;
3506     }else{
3507         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3508         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 xfputs(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 // dictionary 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 silent = false
3636     var dump = false
3637     var builtin = false
3638     cmd := argv[0]
3639     argv = argv[1:]
3640     opt := ""
3641     arg := ""
3642
3643     if 0 < len(argv) {
3644         arg1 := argv[0]
3645         if arg1[0] == '-' {
3646             switch arg1 {
3647                 default: fmt.Printf("--Ed-- Unknown option(%v)\n",arg1)
3648                     return
3649                 case "-b": builtin = true
3650                 case "-d": debug = true
3651                 case "-s": silent = true
3652                 case "-v": info = true
3653             }
3654             opt = arg1
3655             argv = argv[1:]
3656         }
3657     }
3658
3659     dicName := ""
3660     dicURL := ""
3661     if 0 < len(argv) {
3662         arg = argv[0]
3663         dicName = arg
3664         argv = argv[1:]
3665     }
3666     if 0 < len(argv) {
3667         dicURL = argv[0]
3668         argv = argv[1:]
3669     }
3670     if false {
3671         fprintf(stderr, "--Dd-- com(%v) opt(%v) arg(%v)\n",cmd,opt,arg)
3672     }
3673     if cmd == DIC_COM_LOAD {
3674         //dicType := ""
3675         dicBody := ""
3676         if !builtin && dicName != "" && dicURL == "" {
3677             f,err := os.Open(dicName)
3678             if err == nil {
3679                 dicURL = dicName
3680             }else{
3681                 f,err = os.Open(dicName+".html")
3682                 if err == nil {
3683                     dicURL = dicName+".html"
3684                 }else{
3685                     f,err = os.Open("gshdic-"+dicName+".html")
3686                     if err == nil {
3687                         dicURL = "gshdic-"+dicName+".html"
3688                     }
3689                 }
3690             }
3691             if err == nil {
3692                 var buf = make([]byte,128*1024)
3693                 count,err := f.Read(buf)
3694                 f.Close()
3695                 if info {
3696                     fprintf(stderr, "--Id-- ReadDic(%v,%v)\n",count,err)
3697                 }
3698                 dicBody = string(buf[0:count])
3699             }
3700         }
3701         if dicBody == "" {
3702             switch arg {
3703                 default:
3704                     dicName = "WorldDic"
3705                     dicURL = WorldDic
3706                     if info {
3707                         fprintf(stderr, "--Id-- default dictionary \"%v\"\n",
3708                             dicName);
3709                     }
3710                 case "wnn":
3711                     dicName = "WnnDic"
3712                     dicURL = WnnDic
3713                 case "sumomo":
3714                     dicName = "SumomoDic"
3715                     dicURL = SumomoDic
3716                 case "sijimi":
3717                     dicName = "SijimiDic"
3718                     dicURL = SijimiDic
3719                 case "jkl":
3720                     dicName = "JKLJaDic"
3721                     dicURL = JA_JKLDic
3722             }
3723             if debug {
3724                 fprintf(stderr, "--Id-- %v URL=%v\n\n",dicName,dicURL);
3725             }
3726             dicv := strings.Split(dicURL, ",")
3727             if debug {
3728                 fprintf(stderr, "--Id-- %v encoded data...\n",dicName)
3729                 fprintf(stderr, "Type: %v\n",dicv[0])
3730                 fprintf(stderr, "Body: %v\n",dicv[1])
3731                 fprintf(stderr, "\n")
3732             }
3733             body, _ := base64.StdEncoding.DecodeString(dicv[1])
3734             dicBody = string(body)
3735         }
3736         if info {
3737             fmt.Printf("--Id-- %v %v\n",dicName,dicURL)
3738             fmt.Printf("%s\n",dicBody)
3739         }
3740         if debug {
3741             fprintf(stderr, "--Id-- dicName %v text...\n",dicName)
3742             fprintf(stderr, "%v\n",string(dicBody))
3743         }
3744         entv := strings.Split(dicBody, "\n");
3745         if info {
3746             fprintf(stderr, "--Id-- %v scan...\n",dicName);
3747         }
3748         var added int = 0
3749         var dup int = 0

```

```

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

```

```

3875     if 0 < ix {
3876         var iv int = 0
3877         //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3878         fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3879         sval := fmt.Sprintf("%x", iv)
3880         bval := []byte(sval)
3881         dstb = append(dstb, bval...)
3882         si = si+3+ix+1
3883         continue
3884     }
3885 }
3886 if strBegins(src[si:], "%d/") {
3887     // %d/integer// s/a/b/
3888     ix := strings.Index(src[si+3:], "/")
3889     if 0 < ix {
3890         var iv int = 0
3891         fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3892         sval := fmt.Sprintf("%d", iv)
3893         bval := []byte(sval)
3894         dstb = append(dstb, bval...)
3895         si = si+3+ix+1
3896         continue
3897     }
3898 }
3899 if strBegins(src[si:], "%t") {
3900     now := time.Now()
3901     if true {
3902         date := now.Format(time.Stamp)
3903         dstb = append(dstb, []byte(date)...)
3904         si = si+3
3905     }
3906     continue
3907 }
3908 var maxlen int = 0;
3909 var len int;
3910 mi = -1;
3911 for di = 0; di < dicents; di++ {
3912     len = matchlen(src[si:], romkana[di].pat);
3913     if( maxlen < len ){
3914         maxlen = len;
3915         mi = di;
3916     }
3917 }
3918 if( 0 < maxlen ){
3919     out := romkana[mi].out;
3920     dstb = append(dstb, []byte(out)...);
3921     si += maxlen;
3922 }else{
3923     dstb = append(dstb, src[si])
3924     si += 1;
3925 }
3926 }
3927 return string(dstb)
3928 }
3929 func trans(src string)(int){
3930     dst := convs(src);
3931     xfprintf(dst, stderr);
3932     return 0;
3933 }
3934 }
3935 //----- LINEEDIT
3936 // "?" at the top of the line means searching history
3937
3938 // should be compatilbe with Telnet
3939 const (
3940     EV_MODE     = 255
3941     EV_IDLE     = 254
3942     EV_TIMEOUT  = 253
3943     GO_UP       = 252
3944     GO_DOWN     = 251
3945     GO_RIGHT    = 250
3946     GO_LEFT     = 249
3947     DEL_RIGHT   = 248
3948 )
3949
3950 // should return number of octets ready to be read immediately
3951 //fprintf(stderr, "\n--Select(%v %v)\n", err, r.Bits[0])
3952
3953
3954 var EventRecvFd = -1 // file descriptor
3955 var EventSendFd = -1
3956 const EventFdOffset = 1000000
3957 const NormalFdOffset = 100
3958
3959 func putEvent(event int, evarg int){
3960     if true {
3961         if EventRecvFd < 0 {
3962             var pv = []int{-1, -1}
3963             syscall.Pipe(pv)
3964             EventRecvFd = pv[0]
3965             EventSendFd = pv[1]
3966             //fmt.Printf("--De-- EventPipe created[%v, %v]\n", EventRecvFd, EventSendFd)
3967         }
3968     }else{
3969         if EventRecvFd < 0 {
3970             // the document differs from this spec
3971             // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L340
3972             sv, err := syscall.Socketpair(syscall.AF_UNIX, syscall.SOCK_STREAM, 0)
3973             EventRecvFd = sv[0]
3974             EventSendFd = sv[1]
3975             if err != nil {
3976                 fmt.Printf("--De-- EventSock created[%v, %v](%v)\n",
3977                     EventRecvFd, EventSendFd, err)
3978             }
3979         }
3980     }
3981     var buf = []byte{ byte(event) }
3982     n, err := syscall.Write(EventSendFd, buf)
3983     if err != nil {
3984         fmt.Printf("--De-- putEvent[%v](%3v)(%v %v)\n", EventSendFd, event, n, err)
3985     }
3986 }
3987 func ungets(str string){
3988     for _, ch := range str {
3989         putEvent(int(ch), 0)
3990     }
3991 }
3992 func (gsh*GshContext)xReplay(argv []string){
3993     hix := 0
3994     tempo := 1.0
3995     xtempo := 1.0
3996     repeat := 1
3997
3998     for _, a := range argv { // tempo
3999         if strBegins(a, "x") {

```

```

4000         fmt.Sscanf(a[1:], "%f", &xtempo)
4001         tempo = 1 / xtempo
4002         //fprintf(stderr, "--Dr-- tempo=%v%\n", a[2:], tempo);
4003     }else{
4004         if strBegins(a, "r") { // repeat
4005             fmt.Sscanf(a[1:], "%v", &repeat)
4006         }else{
4007             if strBegins(a, "!") {
4008                 fmt.Sscanf(a[1:], "%d", &hix)
4009             }else{
4010                 fmt.Sscanf(a, "%d", &hix)
4011             }
4012         }
4013         if hix == 0 || len(argv) <= 1 {
4014             hix = len(gsh.CommandHistory)-1
4015         }
4016         fmt.Printf("--Ir-- Replay(!%v x%v r%v)\n", hix, xtempo, repeat)
4017         //dumpEvents(hix)
4018         //gsh.xScanReplay(hix, false, repeat, tempo, argv)
4019         go gsh.xScanReplay(hix, true, repeat, tempo, argv)
4020     }
4021 }
4022 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4023 // 2020-0827 GShell-0.2.3
4024 func FpollInl(fp *os.File, usec int) (uintptr) {
4025     nfd := 1
4026
4027     rdv := syscall.FdSet {}
4028     fd1 := fp.Fd()
4029     bank1 := fd1/32
4030     mask1 := int32(1 << fd1)
4031     rdv.Bits[bank1] = mask1
4032
4033     fd2 := -1
4034     bank2 := -1
4035     var mask2 int32 = 0
4036
4037     if 0 <= EventRecvFd {
4038         fd2 = EventRecvFd
4039         nfd = fd2 + 1
4040         bank2 = fd2/32
4041         mask2 = int32(1 << fd2)
4042         rdv.Bits[bank2] |= mask2
4043         //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n", fd2, bank2, mask2)
4044     }
4045
4046     tout := syscall.NsecToTimeval(int64(usec*1000))
4047     //n, err := syscall.Select(nfd, &rdv, nil, nil, &tout) // spec. mismatch
4048     err := syscall.Select(nfd, &rdv, nil, nil, &tout)
4049     if err != nil {
4050         //fmt.Printf("--De-- select() err(%v)\n", err)
4051     }
4052     if err == nil {
4053         if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4054             if false {
4055                 fmt.Printf("--De-- got Event\n")
4056             }
4057             return uintptr(EventFdOffset + fd2)
4058         }else{
4059             if (rdv.Bits[bank1] & mask1) != 0 {
4060                 return uintptr(NormalFdOffset + fd1)
4061             }else{
4062                 return 1
4063             }
4064         }else{
4065             return 0
4066         }
4067     }
4068 func fgetoTimeoutl(fp *os.File, usec int) (int) {
4069     READ1:
4070     readyFd := FpollInl(fp, usec)
4071     if readyFd < 100 {
4072         return EV_TIMEOUT
4073     }
4074
4075     var buf [1]byte
4076
4077     if EventFdOffset <= readyFd {
4078         fd := int(readyFd - EventFdOffset)
4079         _, err := syscall.Read(fd, buf[0:1])
4080         if( err != nil ){
4081             return EOF;
4082         }else{
4083             if buf[0] == EV_MODE {
4084                 recvEvent(fd)
4085                 goto READ1
4086             }
4087             return int(buf[0])
4088         }
4089     }
4090
4091     _, err := fp.Read(buf[0:1])
4092     if( err != nil ){
4093         return EOF;
4094     }else{
4095         return int(buf[0])
4096     }
4097 }
4098
4099 func visibleChar(ch int) (string) {
4100     switch {
4101     case '!' <= ch && ch <= '-':
4102         return string(ch)
4103     }
4104     switch ch {
4105     case '\t': return "\s"
4106     case '\n': return "\\n"
4107     case '\r': return "\\r"
4108     case '\t': return "\\t"
4109     }
4110     switch ch {
4111     case 0x00: return "NUL"
4112     case 0x07: return "BEL"
4113     case 0x08: return "BS"
4114     case 0x0E: return "SO"
4115     case 0x0F: return "SI"
4116     case 0x1B: return "ESC"
4117     case 0x7F: return "DEL"
4118     }
4119     switch ch {
4120     case EV_IDLE: return fmt.Sprintf("IDLE")
4121     case EV_MODE: return fmt.Sprintf("MODE")
4122     }
4123     return fmt.Sprintf("%x", ch)
4124 }

```

```

4125 func recvEvent(fd int){
4126     var buf = make([]byte,1)
4127     _,_ = syscall.Read(fd,buf[0:1])
4128     if( buf[0] != 0 ){
4129         romkanmode = true
4130     }else{
4131         romkanmode = false
4132     }
4133 }
4134 func (gsh*GshContext)xScanReplay(hix int,replay bool,repeat int,tempo float64,argv[string]){
4135     var Start time.Time
4136     var events = []Event{}
4137     for _,e := range Events {
4138         if hix == 0 || e.CmdIndex == hix {
4139             events = append(events,e)
4140         }
4141     }
4142     elen := len(events)
4143     if 0 < elen {
4144         if events[elen-1].event == EV_IDLE {
4145             events = events[0:elen-1]
4146         }
4147     }
4148     for r := 0; r < repeat; r++ {
4149         for i,e := range events {
4150             nano := e.when.Nanosecond()
4151             micro := nano / 1000
4152             if Start.Second() == 0 {
4153                 Start = time.Now()
4154             }
4155             diff := time.Now().Sub(Start)
4156             if replay {
4157                 if e.event != EV_IDLE {
4158                     putEvent(e.event,0)
4159                     if e.event == EV_MODE { // event with arg
4160                         putEvent(int(e.evarg),0)
4161                     }
4162                 }
4163             }else{
4164                 fmt.Printf("#7.3fms %#-3v l%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",
4165                     float64(diff)/1000000.0,
4166                     i,
4167                     e.CmdIndex,
4168                     e.when.Format(time.Stamp),micro,
4169                     e.event,e.event,visibleChar(e.event),
4170                     float64(e.evarg)/1000000.0)
4171             }
4172             if e.event == EV_IDLE {
4173                 d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4174                 //nsleep(time.Duration(e.evarg))
4175                 nsleep(d)
4176             }
4177         }
4178     }
4179 }
4180 func dumpEvents(argv[string]){
4181     hix := 0
4182     if 1 < len(argv) {
4183         fmt.Sscanf(argv[1],"%d",&hix)
4184     }
4185     for i,e := range Events {
4186         nano := e.when.Nanosecond()
4187         micro := nano / 1000
4188         //if e.event != EV_TIMEOUT {
4189             if hix == 0 || e.CmdIndex == hix {
4190                 fmt.Printf("#%-3v l%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",i,
4191                     e.CmdIndex,
4192                     e.when.Format(time.Stamp),micro,
4193                     e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4194             }
4195             //}
4196         }
4197 }
4198 func fgetcTimeout(fp *os.File,usec int)(int){
4199     ch := fgetcTimeout1(fp,usec)
4200     if ch != EV_TIMEOUT {
4201         now := Time.Now()
4202         if 0 < len(Events) {
4203             last := Events[len(Events)-1]
4204             dura := int64(now.Sub(last.when))
4205             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4206         }
4207         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4208     }
4209     return ch
4210 }
4211 }
4212 var TtyMaxCol = 72 // to be obtained by ioctl?
4213 var EscTimeout = (100*1000)
4214 var (
4215     MODE_EditMode    bool    // vi compatible mode
4216     MODE_ShowMode    bool
4217     romkanmode       bool
4218     MODE_Recursive   bool    // recursive translation
4219     MODE_CapsLock    bool    // software CapsLock
4220     MODE_LowerLock   bool    // force lower-case character lock
4221     MODE_Viinsert    int    // visible insert mode, should be like "I" icon in X Window
4222     MODE_ViTrace     bool    // output newline before translation
4223 )
4224 type IInput struct {
4225     lno      int
4226     lastlno int
4227     pch      []int // input queue
4228     prompt   string
4229     line     string
4230     right    string
4231     inJmode  bool
4232     pinJmode bool
4233     waitingMeta string // waiting meta character
4234     LastCmd   string
4235 }
4236 func (iin*IInput)Getc(timeoutUs int)(int){
4237     ch1 := EOF
4238     ch2 := EOF
4239     ch3 := EOF
4240     if( 0 < len(iin.pch) ){ // deQ
4241         ch1 = iin.pch[0]
4242         iin.pch = iin.pch[1:]
4243     }else{
4244         ch1 = fgetcTimeout(stdin,timeoutUs);
4245     }
4246     if( ch1 == 033 ){ // escape sequence
4247         ch2 = fgetcTimeout(stdin,EscTimeout);
4248         if( ch2 == EV_TIMEOUT ){
4249             }else{

```

```

4250     ch3 = fgetcTimeout(stdin,EscTimeout);
4251     if( ch3 == EV_TIMEOUT ){
4252         iin.pch = append(iin.pch,ch2) // enQ
4253     }else{
4254         switch( ch2 ){
4255             default:
4256                 iin.pch = append(iin.pch,ch2) // enQ
4257                 iin.pch = append(iin.pch,ch3) // enQ
4258             case 'l':
4259                 switch( ch3 ){
4260                     case 'A': ch1 = GO_UP; // ^
4261                     case 'B': ch1 = GO_DOWN; // v
4262                     case 'C': ch1 = GO_RIGHT; // >
4263                     case 'D': ch1 = GO_LEFT; // <
4264                     case '3':
4265                 }
4266             ch4 := fgetcTimeout(stdin,EscTimeout);
4267             if( ch4 == '-' ){
4268                 //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4269                 ch1 = DEL_RIGHT
4270             }
4271             case '\\':
4272                 //ch4 := fgetcTimeout(stdin,EscTimeout);
4273                 //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4274                 switch( ch3 ){
4275                     case '-': ch1 = DEL_RIGHT
4276                 }
4277             }
4278         }
4279     }
4280     return ch1
4281 }
4282 }
4283 func (inn*IInput)clearline(){
4284     var i int
4285     fprintf(stderr,"\r");
4286     // should be ANSI ESC sequence
4287     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4288         fputc(' ',os.Stderr);
4289     }
4290     fprintf(stderr,"\r");
4291 }
4292 func (iin*IInput)Redraw(){
4293     redraw(iin,iin.lno,iin.line,iin.right)
4294 }
4295 func redraw(iin *IInput,lno int,line string,right string){
4296     inMeta := false
4297     showMode := ""
4298     showMeta := "" // visible Meta mode on the cursor position
4299     showLino := fmt.Sprintf("%d!",lno)
4300     InsertMark := "" // in visible insert mode
4301
4302     if MODE_EditMode {
4303     }else
4304     if 0 < len(iin.right) {
4305         InsertMark = " "
4306     }
4307
4308     if( 0 < len(iin.waitingMeta) ){
4309         inMeta = true
4310         if iin.waitingMeta[0] != 033 {
4311             showMeta = iin.waitingMeta
4312         }
4313     }
4314     if( romkanmode ){
4315         //romkanmark = " *";
4316     }else{
4317         //romkanmark = "";
4318     }
4319     if MODE_ShowMode {
4320         romkan := "---"
4321         inmeta := "-"
4322         inveri := ""
4323         if MODE_CapsLock {
4324             inmeta = "A"
4325         }
4326         if MODE_LowerLock {
4327             inmeta = "a"
4328         }
4329         if MODE_ViTrace {
4330             inveri = "v"
4331         }
4332         if MODE_EditMode {
4333             inveri = ":"
4334         }
4335     }
4336     if romkanmode {
4337         romkan = "\343\201\202"
4338         if MODE_CapsLock {
4339             inmeta = "R"
4340         }else{
4341             inmeta = "r"
4342         }
4343     }
4344     if inMeta {
4345         inmeta = "\\ "
4346     }
4347     showMode = "["+romkan+inmeta+inveri+";";
4348
4349     Pre := "\r" + showMode + showLino
4350     Output := ""
4351     Left := ""
4352     Right := ""
4353     if romkanmode {
4354         Left = convs(line)
4355         Right = InsertMark+convs(right)
4356     }else{
4357         Left = line
4358         Right = InsertMark+right
4359     }
4360     Output = Pre+Left
4361     if MODE_ViTrace {
4362         Output += iin.LastCmd
4363     }
4364     Output += showMeta+Right
4365     for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4366         Output += " "
4367     }
4368     // should be ANSI ESC sequence
4369     // not necessary just after newline
4370     Output += Pre+Left+showMeta // to set the cursor to the current input position
4371     fprintf(stderr,"%s",Output)
4372
4373     if MODE_ViTrace {
4374         if 0 < len(iin.LastCmd) {
4375             iin.LastCmd = ""

```

```

4375     fprintf(stderr, "\r\n")
4376     }
4377     }
4378 }
4379 func delHeadChar(str string)(rline string, head string){
4380     _, clen := utf8.DecodeRune([]byte(str))
4381     head = string(str[0:clen])
4382     return str[clen:], head
4383 }
4384 func delTailChar(str string)(rline string, last string){
4385     var i = 0
4386     var clen = 0
4387     for {
4388         _, siz := utf8.DecodeRune([]byte(str)[i:])
4389         if siz <= 0 { break }
4390         clen = siz
4391         i += siz
4392     }
4393     last = str[len(str)-clen:]
4394     return str[0:len(str)-clen], last
4395 }
4396
4397 // 3> for output and history
4398 // 4> for keylog?
4399 // <a name="getline">Command Line Editor</a>
4400 func xgetline(lno int, prevline string, gsh*GshContext)(string){
4401     var iin IInput
4402     iin.lastlno = lno
4403     iin.lno = lno
4404
4405     CmdIndex = len(gsh.CommandHistory)
4406     if( isatty(0) == 0 ){
4407         if( sfgets(&iin.line, LINESIZE, stdin) == NULL ){
4408             iin.line = "exit\n";
4409         }else{
4410             }
4411         return iin.line
4412     }
4413     if( true ){
4414         //var pts string;
4415         //pts = ptsname(0);
4416         //pts = ttyname(0);
4417         //fprintf(stderr, "--pts[0] = %s\n", pts?pts:"?");
4418     }
4419     if( false ){
4420         fprintf(stderr, "! ");
4421         fflush(stderr);
4422         sfgets(&iin.line, LINESIZE, stdin);
4423         return iin.line
4424     }
4425     system("/bin/stty -echo -icanon");
4426     xline := iin.xgetline(prevline, gsh)
4427     system("/bin/stty echo sane");
4428     return xline
4429 }
4430 func (iin*IInput)Translate(cmdch int){
4431     romkanmode = !romkanmode;
4432     if MODE_ViTrace {
4433         fprintf(stderr, "%v\r\n", string(cmdch));
4434     }else
4435     if( cmdch == 'J' ){
4436         fprintf(stderr, "J\r\n");
4437         iin.inJmode = true
4438     }
4439     iin.Redraw();
4440     loadDefaultDic(cmdch);
4441     iin.Redraw();
4442 }
4443 func (iin*IInput)Replace(cmdch int){
4444     iin.LastCmd = fmt.Sprintf("\\%v", string(cmdch))
4445     iin.Redraw();
4446     loadDefaultDic(cmdch);
4447     dst := convs(iin.line+iin.right);
4448     iin.line = dst
4449     iin.right = ""
4450     if( cmdch == 'I' ){
4451         fprintf(stderr, "I\r\n");
4452         iin.inJmode = true
4453     }
4454     iin.Redraw();
4455 }
4456 func (iin*IInput)xgetline(prevline string, gsh*GshContext)(string){
4457     var ch int;
4458
4459     MODE_ShowMode = false
4460     iin.Redraw();
4461     first := true
4462
4463     for cix := 0; ; cix++ {
4464         iin.pinJmode = iin.inJmode
4465         iin.inJmode = false
4466
4467         ch = iin.Getc(1000*1000)
4468
4469         if ch != EV_TIMEOUT && first {
4470             first = false
4471             mode := 0
4472             if romkanmode {
4473                 mode = 1
4474             }
4475             now := time.Now()
4476             Events = append(Events, Event{now, EV_MODE, int64(mode), CmdIndex})
4477         }
4478         if ch == 033 {
4479             MODE_ShowMode = true
4480             MODE_EditMode = !MODE_EditMode
4481             iin.Redraw();
4482             continue
4483         }
4484         if MODE_EditMode {
4485             switch ch {
4486                 case 'j': ch = GO_DOWN
4487                 case 'k': ch = GO_UP
4488                 case 'h': ch = GO_LEFT
4489                 case 'l': ch = GO_RIGHT
4490                 case 'x': ch = DEL_RIGHT
4491                 case 'a': MODE_EditMode = !MODE_EditMode
4492                 case 'i': MODE_EditMode = !MODE_EditMode
4493                 case 'i': MODE_EditMode = !MODE_EditMode
4494                 case 'i': MODE_EditMode = !MODE_EditMode
4495                 case '-':
4496                     right, head := delHeadChar(iin.right)
4497                     if len([]byte(head)) == 1 {
4498                         ch = int(head[0])
4499

```

```

4500         if( 'a' <= ch && ch <= 'z' ){
4501             ch = ch + 'A'-'a'
4502         }else
4503         if( 'A' <= ch && ch <= 'Z' ){
4504             ch = ch + 'a'-'A'
4505         }
4506         iin.right = string(ch) + right
4507     }
4508     iin.Redraw();
4509     continue
4510 }
4511 }
4512
4513 //fprintf(stderr,"A[%02X]\n",ch);
4514 if( ch == '\0' || ch == 033 ){
4515     MODE_ShowMode = true
4516     metach := ch
4517     iin.waitingMeta = string(ch)
4518     iin.Redraw();
4519     // set cursor //fprintf(stderr,"???\b\b\b")
4520     ch = fgetcTimeout(stdin,2000*1000)
4521     // reset cursor
4522     iin.waitingMeta = ""
4523
4524     cmdch := ch
4525     if( ch == EV_TIMEOUT ){
4526         if metach == 033 {
4527             continue
4528         }
4529         ch = metach
4530     }else
4531     /*
4532     if( ch == 'm' || ch == 'M' ){
4533         mch := fgetcTimeout(stdin,1000*1000)
4534         if mch == 'r' {
4535             romkanmode = true
4536         }else{
4537             romkanmode = false
4538         }
4539         continue
4540     }else
4541     */
4542     if( ch == 'k' || ch == 'K' ){
4543         MODE_Recursive = IMODE_Recursive
4544         iin.Translate(cmdch);
4545         continue
4546     }else
4547     if( ch == 'j' || ch == 'J' ){
4548         iin.Translate(cmdch);
4549         continue
4550     }else
4551     if( ch == 'i' || ch == 'I' ){
4552         iin.Replace(cmdch);
4553         continue
4554     }else
4555     if( ch == 'l' || ch == 'L' ){
4556         MODE_LowerLock = IMODE_LowerLock
4557         MODE_CapsLock = false
4558         if MODE_ViTrace {
4559             fprintf(stderr,"%v\r\n",string(cmdch));
4560         }
4561         iin.Redraw();
4562         continue
4563     }else
4564     if( ch == 'u' || ch == 'U' ){
4565         MODE_CapsLock = !MODE_CapsLock
4566         MODE_LowerLock = false
4567         if MODE_ViTrace {
4568             fprintf(stderr,"%v\r\n",string(cmdch));
4569         }
4570         iin.Redraw();
4571         continue
4572     }else
4573     if( ch == 'v' || ch == 'V' ){
4574         MODE_ViTrace = !MODE_ViTrace
4575         if MODE_ViTrace {
4576             fprintf(stderr,"%v\r\n",string(cmdch));
4577         }
4578         iin.Redraw();
4579         continue
4580     }else
4581     if( ch == 'c' || ch == 'C' ){
4582         if 0 < len(iin.line) {
4583             xline,tail := delTailChar(iin.line)
4584             if len([]byte(tail)) == 1 {
4585                 ch = int(tail[0])
4586                 if( 'a' <= ch && ch <= 'z' ){
4587                     ch = ch + 'A'-'a'
4588                 }else
4589                 if( 'A' <= ch && ch <= 'Z' ){
4590                     ch = ch + 'a'-'A'
4591                 }
4592                 iin.line = xline + string(ch)
4593             }
4594         }
4595         if MODE_ViTrace {
4596             fprintf(stderr,"%v\r\n",string(cmdch));
4597         }
4598         iin.Redraw();
4599         continue
4600     }else{
4601         iin.pch = append(iin.pch,ch) // push
4602         ch = '\0'
4603     }
4604 }
4605 switch( ch ){
4606 case 'P'-0x40: ch = GO_UP
4607 case 'N'-0x40: ch = GO_DOWN
4608 case 'B'-0x40: ch = GO_LEFT
4609 case 'F'-0x40: ch = GO_RIGHT
4610 }
4611 //fprintf(stderr,"B[%02X]\n",ch);
4612 switch( ch ){
4613 case 0:
4614     continue;
4615
4616 case '\t':
4617     iin.Replace('j');
4618     continue
4619 case 'X'-0x40:
4620     iin.Replace('j');
4621     continue
4622
4623 case EV_TIMEOUT:
4624     iin.Redraw();

```

```

4625         if iin.pinJmode {
4626             fprintf(stderr, "\\J\\r\\n")
4627             iin.inJmode = true
4628         }
4629         continue
4630     case GO_UP:
4631         if iin.lno == 1 {
4632             continue
4633         }
4634         cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
4635         if ok {
4636             iin.line = cmd
4637             iin.right = ""
4638             iin.lno = iin.lno - 1
4639         }
4640         iin.Redraw();
4641         continue
4642     case GO_DOWN:
4643         cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
4644         if ok {
4645             iin.line = cmd
4646             iin.right = ""
4647             iin.lno = iin.lno + 1
4648         }else{
4649             iin.line = ""
4650             iin.right = ""
4651             if iin.lno == iin.lastlno-1 {
4652                 iin.lno = iin.lno + 1
4653             }
4654         }
4655         iin.Redraw();
4656         continue
4657     case GO_LEFT:
4658         if 0 < len(iin.line) {
4659             xline,tail := delTailChar(iin.line)
4660             iin.line = xline
4661             iin.right = tail + iin.right
4662         }
4663         iin.Redraw();
4664         continue;
4665     case GO_RIGHT:
4666         if( 0 < len(iin.right) && iin.right[0] != 0 ){
4667             xright,head := delHeadChar(iin.right)
4668             iin.right = xright
4669             iin.line += head
4670         }
4671         iin.Redraw();
4672         continue;
4673     case EOF:
4674         goto EXIT;
4675     case 'R'-0x40: // replace
4676         dst := convs(iin.line+iin.right);
4677         iin.line = dst
4678         iin.right = ""
4679         iin.Redraw();
4680         continue;
4681     case 'T'-0x40: // just show the result
4682         readDic();
4683         romkanmode = !romkanmode;
4684         iin.Redraw();
4685         continue;
4686     case 'L'-0x40:
4687         iin.Redraw();
4688         continue
4689     case 'K'-0x40:
4690         iin.right = ""
4691         iin.Redraw();
4692         continue
4693     case 'E'-0x40:
4694         iin.line += iin.right
4695         iin.right = ""
4696         iin.Redraw();
4697         continue
4698     case 'A'-0x40:
4699         iin.right = iin.line + iin.right
4700         iin.line = ""
4701         iin.Redraw();
4702         continue
4703     case 'U'-0x40:
4704         iin.line = ""
4705         iin.right = ""
4706         iin.clearline();
4707         iin.Redraw();
4708         continue;
4709     case DEL_RIGHT:
4710         if( 0 < len(iin.right) ){
4711             iin.right,_ = delHeadChar(iin.right)
4712             iin.Redraw();
4713         }
4714         continue;
4715     case 0x7F: // BS? not DEL
4716         if( 0 < len(iin.line) ){
4717             iin.line,_ = delTailChar(iin.line)
4718             iin.Redraw();
4719         }
4720         /*
4721         else
4722             if( 0 < len(iin.right) ){
4723                 iin.right,_ = delHeadChar(iin.right)
4724                 iin.Redraw();
4725             }
4726         */
4727         continue;
4728     case 'H'-0x40:
4729         if( 0 < len(iin.line) ){
4730             iin.line,_ = delTailChar(iin.line)
4731             iin.Redraw();
4732         }
4733         continue;
4734     }
4735     if( ch == '\\n' || ch == '\\r' ){
4736         iin.line += iin.right;
4737         iin.right = ""
4738         iin.Redraw();
4739         fputc(ch,stderr);
4740         break;
4741     }
4742     if MODE_CapsLock {
4743         if 'a' <= ch && ch <= 'z' {
4744             ch = ch+'A'-'a'
4745         }
4746     }
4747     if MODE_LowerLock {
4748         if 'A' <= ch && ch <= 'Z' {
4749             ch = ch+'a'-'A'

```

```

4750     }
4751     }
4752     iin.line += string(ch);
4753     iin.Redraw();
4754 }
4755 EXIT:
4756     return iin.line + iin.right;
4757 }
4758
4759 func getline_main(){
4760     line := xgetline(0,"",nil)
4761     fprintf(stderr,"%s\n",line);
4762 /*
4763     dp = strpbrk(line,"\r\n");
4764     if( dp != NULL ){
4765         *dp = 0;
4766     }
4767
4768     if( 0 ){
4769         fprintf(stderr,"\n(%d)\n",int(strlen(line)));
4770     }
4771     if( lseek(3,0,0) == 0 ){
4772         if( romkammode ){
4773             var buf [8*1024]byte;
4774             convs(line,buf);
4775             strepy(line,buf);
4776         }
4777         write(3,line,strlen(line));
4778         ftruncate(3,lseek(3,0,SEEK_CUR));
4779         //fprintf(stderr,"outsize=%d\n",int)lseek(3,0,SEEK_END));
4780         lseek(3,0,SEEK_SET);
4781         close(3);
4782     }else{
4783         fprintf(stderr,"\r\n gotline: ");
4784         trans(line);
4785         //printf("%s\n",line);
4786         printf("\n");
4787     }
4788 */
4789 }
4790 //== end ===== getline
4791
4792 //
4793 // $USERHOME/.gsh/
4794 //     gsh-rc.txt, or gsh-configure.txt
4795 //     gsh-history.txt
4796 //     gsh-aliases.txt // should be conditional?
4797 //
4798 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
4799     homedir_found := userHomeDir()
4800     if !found {
4801         fmt.Printf("--E-- You have no UserHomeDir\n")
4802         return true
4803     }
4804     gshhome := homedir + "/" + GSH_HOME
4805     _, err2 := os.Stat(gshhome)
4806     if err2 != nil {
4807         err3 := os.Mkdir(gshhome,0700)
4808         if err3 != nil {
4809             fmt.Printf("--E-- Could not Create %s (%s)\n",
4810                 gshhome,err3)
4811             return true
4812         }
4813         fmt.Printf("--I-- Created %s\n",gshhome)
4814     }
4815     gshCtx.GshHomeDir = gshhome
4816     return false
4817 }
4818 func setupGshContext()(GshContext,bool){
4819     gshPA := syscall.ProcAttr {
4820         "", // the staring directory
4821         os.Environ(), // environ[]
4822         [uintptr(os.Stdin.Fd()),os.Stdout.Fd(),os.Stderr.Fd()]},
4823         nil, // OS specific
4824     }
4825     cwd, _ := os.Getwd()
4826     gshCtx := GshContext {
4827         cwd, // StartDir
4828         "", // GetLine
4829         [][GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
4830         gshPA,
4831         [][GCommandHistory{}], //something for invokation?
4832         GCommandHistory{}, // CmdCurrent
4833         false,
4834         [][int{}],
4835         syscall.Rusage{}},
4836         "", // GshHomeDir
4837         Ttyid(),
4838         false,
4839         false,
4840         [][PluginInfo{}],
4841         [][string{}],
4842         "",
4843         "v",
4844         ValueStack{},
4845         GServer{"",""}, // LastServer
4846         "", // RSERV
4847         cwd, // RWD
4848         CheckSum{}},
4849     }
4850     err := gshCtx.gshSetupHomedir()
4851     return gshCtx, err
4852 }
4853 func (gsh*GshContext)gshellh(gline string)(bool){
4854     ghist := gsh.CmdCurrent
4855     ghist.WorkDir, _ = os.Getwd()
4856     ghist.WorkDirX = len(gsh.ChdirHistory)-1
4857     //fmt.Printf("--D--ChdirHistory(%d)\n",len(gsh.ChdirHistory))
4858     ghist.StartAt = time.Now()
4859     rusagev1 := Getrusagev()
4860     gsh.CmdCurrent.FoundFile = [][string{}
4861     fin := gsh.tgshellh(gline)
4862     rusagev2 := Getrusagev()
4863     ghist.Rusagev = RusagevSubv(rusagev2,rusagev1)
4864     ghist.EndAt = time.Now()
4865     ghist.CmdLine = gline
4866     ghist.FoundFile = gsh.CmdCurrent.FoundFile
4867
4868     /* record it but not show in list by default
4869     if len(gline) == 0 {
4870         continue
4871     }
4872     if gline == "hi" || gline == "history" { // don't record it
4873         continue
4874     }

```

```

4875  */
4876  gsh.CommandHistory = append(gsh.CommandHistory, ghist)
4877  return fin
4878 }
4879 // <a name="main">Main loop</a>
4880 func script(gshCtxGiven *GshContext) (_ GshContext) {
4881     gshCtxBuf,err0 := setupGshContext()
4882     if err0 {
4883         return gshCtxBuf;
4884     }
4885     gshCtx := &gshCtxBuf
4886
4887     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
4888     //resmap()
4889
4890     /*
4891     if false {
4892         gsh_getlinev, with_exgetline :=
4893             which("PATH",[]string{"which","gsh-getline","-s"})
4894         if with_exgetline {
4895             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
4896             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
4897         }else{
4898             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
4899         }
4900     }
4901     */
4902
4903     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
4904     gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
4905
4906     prevline := ""
4907     skipping := false
4908     for hix := len(gshCtx.CommandHistory); ; {
4909         gline := gshCtx.getline(hix,skipping,prevline)
4910         if skipping {
4911             if strings.Index(gline,"fi") == 0 {
4912                 fmt.Printf("fi\n");
4913                 skipping = false;
4914             }else{
4915                 //fmt.Printf("%s\n",gline);
4916             }
4917             continue
4918         }
4919         if strings.Index(gline,"if") == 0 {
4920             //fmt.Printf("--D-- if start: %s\n",gline);
4921             skipping = true;
4922             continue
4923         }
4924         if false {
4925             os.Stdout.Write([]byte("gotline:"))
4926             os.Stdout.Write([]byte(gline))
4927             os.Stdout.Write([]byte("\n"))
4928         }
4929         gline = strsubst(gshCtx,gline,true)
4930         if false {
4931             fmt.Printf("fmt.Printf %%v - %v\n",gline)
4932             fmt.Printf("fmt.Printf %%s - %s\n",gline)
4933             fmt.Printf("fmt.Printf %%x - %x\n",gline)
4934             fmt.Printf("fmt.Printf %%U - %s\n",gline)
4935             fmt.Printf("Stoutt.Write -")
4936             os.Stdout.Write([]byte(gline))
4937             fmt.Printf("\n")
4938         }
4939         /*
4940         // should be cared in substitution ?
4941         if 0 < len(gline) && gline[0] == '!' {
4942             xgline, set, err := searchHistory(gshCtx,gline)
4943             if err {
4944                 continue
4945             }
4946             if set {
4947                 // set the line in command line editor
4948             }
4949             gline = xgline
4950         }
4951         */
4952         fin := gshCtx.gshelllh(gline)
4953         if fin {
4954             break;
4955         }
4956         prevline = gline;
4957         hix++;
4958     }
4959     return *gshCtx
4960 }
4961 func main() {
4962     gshCtxBuf := GshContext{}
4963     gsh := &gshCtxBuf
4964     argv := os.Args
4965     if 1 < len(argv) {
4966         if isin("version",argv){
4967             gsh.showVersion(argv)
4968             return
4969         }
4970         comx := isinX("-c",argv)
4971         if 0 < comx {
4972             gshCtxBuf,err := setupGshContext()
4973             gsh := &gshCtxBuf
4974             if !err {
4975                 gsh.gshellv(argv[comx+1:])
4976             }
4977             return
4978         }
4979     }
4980     if 1 < len(argv) && isin("-s",argv) {
4981     }else{
4982         gsh.showVersion(append(argv,[]string{"-l","-a"}...))
4983     }
4984     script(nil)
4985     //gshCtx := script(nil)
4986     //gshell(gshCtx,"time")
4987 }
4988 //</div></details>
4989 //<details id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
4990 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
4991 // - merged histories of multiple parallel gsh sessions
4992 // - alias as a function or macro
4993 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
4994 // - retrieval PATH of files by its type
4995 // - gsh as an IME with completion using history and file names as dictionaies
4996 // - gsh a scheduler in precise time of within a millisecond
4997 // - all commands have its subcomand after "---" symbol
4998 // - filename expansion by "-find" command
4999 // - history of ext code and output of each commoand

```

```

5000 // - "script" output for each command by pty-tee or telnet-tee
5001 // - $BUILTIN command in PATH to show the priority
5002 // - "?" symbol in the command (not as in arguments) shows help request
5003 // - searching command with wild card like: which ssh-*
5004 // - longformat prompt after long idle time (should dismiss by BS)
5005 // - customizing by building plugin and dynamically linking it
5006 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
5007 // - "?" symbol should be used for negation, don't wast it just for job control
5008 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
5009 // - making canonical form of command at the start adding quotation or white spaces
5010 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
5011 // - name? or name! might be usefull
5012 // - name? format - packing directory contents into a single html file using data scheme
5013 // - filepath substitution should be done by each command, especially in case of builtins
5014 // - @N substitution for the history of working directory, and @spec for more generic ones
5015 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
5016 // - GSH_PATH for plugins
5017 // - standard command output: list of data with name, size, resource usage, modified time
5018 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
5019 // - wc word-count, grep match line count, ...
5020 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
5021 // - -tailf-filename like tail -f filename, repeat close and open before read
5022 // - max.size and max.duration and timeout of (generated) data transfer
5023 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
5024 // - IME "?" at the top of the command line means searching history
5025 // - IME %d/0x10000/ %x/ffff/
5026 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
5027 // - gsh in WebAssembly
5028 // - gsh as a HTTP server of online-manual
5029 //---END--- (~-~)/ITS more</div></details>
5030
5031 <<span class="gsh-golang-data">
5032
5033 var WorldDic = <<span id="gsh-world-dic">
5034 "data:text/dic;base64,+
5035 "Ly8yTkJ1TlUUVmNC4wLWJlZjE6eSpu4ICgyMDIwLTA4MTlhKQpzZWthaSDkuJbnlYwKa28g44GT"+
5036 "Cm5uIOOCkpwpaSDjgasKY2hpIOOBQp0aSDjgaEKaGEG44GvCnNlIOOBmwpYSDjgYsKaSDj"+
5037 "gYOK";
5038 </span>
5039
5040 var WnnDic = <<span id="gsh-wnn-dic">
5041 "data:text/dic;base64,+
5042 "Pc1ldEgY2hhcn1ldD0iVVRGLTgiPgo8dGV4dGFyZWEgY29scz04MCIyYzdzPTQwPgovL2R2p"+
5043 "Y3ZlcglHU2h1bGxccc0lNRVxzZG1jdg1vbmFyeVxzZm9yXHNtdG1wbW9ccy8yXHMvMDIwLTA4MzAK"+
5044 "RlNOzWxsCUDtAGYsBArjgo/jgZCj56eBCndhdGFzaGk56eBCndhdGFzaQnnp4EK44Gg"+
5045 "44G+44GICeWQjeWJlQmVhZG1ldD0iVVRGLTgiPgo8dGV4dGFyZWEgY29scz04MCIyYzdzPTQwPgovL2R2p"+
5046 "Y3ZlcglHU2h1bGxccc0lNRVxzZG1jdg1vbmFyeVxzZm9yXHNtdG1wbW9ccy8yXHMvMDIwLTA4MzAK"+
5047 "ZQnJgYyKAGErJ44GvCm5hCeoBqgprYQnJgYsKbM8J44GvCmRlCeOBpwpzQnJgZKZVxzCWFj"+
5048 "aE8KZG1jCWRpWp1Y2hvcWVjag8KcmVwbGF5CXJlcGxheQpyZXB1YXQJcmVwZWF0CmR0CWRh"+
5049 "dGVccysnJVklsVklSVlO1VnO1VTJwp0aW9uCXRPb24KJXQJXQJLY8dG8g8YmUgYW4gYWN0"+
5050 "aW9uCXJw4Gv4dGFyZWE+CG=="
5051 </span>
5052
5053 var SumomoDic = <<span id="gsh-sumomo-dic">
5054 "data:text/dic;base64,+
5055 "Pc1ldEgY2hhcn1ldD0iVVRGLTgiPgo8dGV4dGFyZWEgY29scz04MCIyYzdzPTQwPgovL3Z1"+
5056 "cg1HU2h1bGxccc0lNRVxzZG1jdg1vbmFyeVxzZm9yXHNtdG1wbW9ccy8yXHMvMDIwLTA4MzAK"+
5057 "c3UJ44GZCm1vCeOCgpubwnjga4KdQnJgYyK2hpCeOBQp0aQnJgaEKdWNoaQnlhOUk4XRP"+
5058 "CeWgQpzd1vW8J44GZ44K44KCCn1bW9tbt21vCeOBmeOCguOCgqptb21vCeahgpwt"+
5059 "b21vbW8J5qGD44KCCiwsCeOAgQouLgnjgIIRKPC90Zkh0YXJlYT4K"
5060 </span>
5061
5062 var SijimiDic = <<span id="gsh-sijimi-dic">
5063 "data:text/dic;base64,+
5064 "Pc1ldEgY2hhcn1ldD0iVVRGLTgiPgo8dGV4dGFyZWEgY29scz04MCIyYzdzPTQwPgovL3Z1"+
5065 "cg1HU2h1bGxccc0lNRVxzZG1jdg1vbmFyeVxzZm9yXHNtdG1wbW9ccy8yXHMvMDIwLTA4MzAK"+
5066 "CnNpCeOB1wpzAGJ44GXCMppCeOBmApraQnJgB8KbMEJ44GqCmplCeOBmOCQp4eXUJ44KF"+
5067 "CnUJ44GCGm5pCeOBqprbwnjgZMKYnUJ44GZCm5uCeOCkwpubwnjga4KY2hpCeOBQp0aQnJ"+
5068 "gaEKa2EJ44GLCnJhCeOCiQosLanJgIEKLi4J44CCnHUyW5hCeS4gwp4anVlCeWnQp4bmkJ"+
5069 "5LqCmtveAnlIeKa29xCeWaiwprb3gY5YCLCm5hbmfgdXVuaXgJNzIKbmfUYwpldW5eHgJ"+
5070 "77yX77yScm5hbmfgdXVuaVgJ77yX77yScuS4g+WNgE86JhGJNzIKa29IdW5uCeWAI+WIhgp0"+
5071 "aWthcmFXCeOBEOBi+OCiQp0aWthcmEJ5YqbcCmNoaWthcmEJ5YqbcJwv4dGFyZWE+CG=="
5072 </span>
5073
5074 var JA_JKLDic = <<span id="gsh-ja-jkl-dic">
5075 "data:text/dic;base64,+
5076 "Ly92ZjJsCU15SU1fAmRyY2ptb3JzZWpKQWpKS0w0MjAyMGVwODE5KShELV4pL1NhdG94SVRT"+
5077 "CmtqamprbGtqa2tsa2psIOS41ueVjApqamtqamJ44GCMtqbAnjgYQKa2tqbAnjgYKamtq"+
5078 "amwJ44GICmVpZAGJ44GXCMppCeOBmApraQnJgB8KbMEJ44GqCmplCeOBmOCQp4eXUJ44KF"+
5079 "CeOBKQpampqAnjgZMKamtq2psCeOB1Opqamtq2wJ44GXCMpamtqbAnjgZKka2pqamts"+
5080 "CeOBmwpqamprAnjgZ0KamtCeOBnwptra2prbAnjgaEKa2pqa2wJ44GCMtqa2pqAnjgaYK"+
5081 "a2tqa2tsCeOBqApramtsCeOBqppqa2prbAnjgKa2tra2wJ44GCMpqa2psCeOBpOpra2pq"+
5082 "bAnjga4Kamtra2wJ44GvCmpqa2tqbAnjgIKampra2wJ44GCMtsCeOBuAqpa2tsCeOBuwpq"+
5083 "a2tqbAnjgB4Ka2tqa2psCeOBvwpqAnjgAKamtRa2psCeOCgOpqa2tqa2wJ44KCMtqamwJ"+
5084 "44KECmpra2pqAnjgoyKampsCeOCiApra2tsCeOCiQpamtCeOCiQpqa2pqa2wJ44KCMpqa"+
5085 "amwJ44KCMtqa2psCeOCjOpqa2psCeOCjwpramramwJ44KQCMtqamrAnjgJpKa2pqamwJ"+
5086 "44KSCmtqa2prbAnjgpMka2pqa2psCeODvApra2wJ44KbCmramprAnjgpwKa2pramtqbAnj"+
5087 "gIEK";
5088 </span>
5089
5090 </span>
5091 /*
5092 <details id="references"><summary>References</summary><div class="gsh-src">
5093 <p>
5094 <a href="https://golang.org">The Go Programming Language</a>
5095 <iframe src="https://golang.org" width="100%" height="300"></iframe>
5096
5097 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
5098 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
5099 CSS:
5100 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/Selectors">Selectors</a>
5101 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
5102 HTTP
5103 JavaScript:
5104 ...
5105 </p>
5106 </div></details>
5107 */
5108 /*
5109 <details id="html-src" onclick="frame_open();"><summary>Raw Source</summary><div>
5110
5111 <!-- h2>The full of this HTML including the Go code is here.</h2 -->
5112 <details id="gsh-whole-view"><summary>Whole file</summary>
5113 <a name="whole-src-view"></a>
5114 <span id="src-frame"></span><!-- a window to show source code -->
5115 </details>
5116
5117 <details id="gsh-style-frame" onclick="fill_CSSView()"><summary>CSS part</summary>
5118 <a name="style-src-view"></a>
5119 <span id="gsh-style-view"></span>
5120 </details>
5121
5122 <details id="gsh-script-frame" onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
5123 <a name="script-src-view"></a>
5124 <span id="gsh-script-view"></span>

```

```

5125 </details>
5126
5127 <details id="gsh-data-frame" onclick="fill_DataView()"><summary>Builtin data part</summary>
5128 <a name="gsh-data-frame"></a>
5129 <span id="gsh-data-view"></span>
5130 </details>
5131
5132 </div></details>
5133 */
5134 /*
5135 <div id="gsh-footer" style=""></div><!-- ----- END-OF-VISIBLE-PART ----- -->
5136
5137
5138 <style id="gsh-style-def">
5139 //body {display:none;}
5140 .gsh-link{color:green;}
5141 #gsh {border-width:1px;margin:0;padding:0;}
5142 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
5143 #gsh header{height:100px;}
5144 #gsh header{height:100px;background-image:url(GShell-Logo00.png);}
5145 #gsh-menu{font-size:14pt;color:#f88;}
5146 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
5147 #gsh note{color:#000;font-size:10pt;}
5148 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
5149 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
5150 #gsh summary{font-size:16pt;color:#fff;background-color:#8af;height:30px;}
5151 #gsh pre{font-size:11pt;color:#223;background-color:#fafff;}
5152 #gsh a{color:#24a;}
5153 #gsh a[name]{color:#24a;font-size:16pt;}
5154 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5155 #gsh .gsh-src{background-color:#fafff;color:#223;}
5156 #gsh-src-src{spellcheck:false}
5157 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5158 #src-frame-textarea{background-color:#fafff;color:#223;}
5159 .gsh-code {white-space:pre;font-family:monospace 11pt;}
5160 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
5161 .gsh-golang-data {display:none;}
5162 #gsh-winId {color:#000;font-size:14pt;}
5163
5164 #gsh-statement {font-size:11pt;background-color:#fff;font-family:Georgia;}
5165 #gsh-statement {color:#000;background-color:#fff !important;}
5166 #gsh-statement h2{color:#000;background-color:#fff !important;}
5167 #gsh-statement details{color:#000;background-color:#fff;font-family:Georgia;}
5168 #gsh-statement p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
5169 #gsh-statement address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
5170
5171 @media print {
5172 #gsh pre{font-size:11pt !important;}
5173 }
5174 </style>
5175
5176 <!--
5177 // Logo image should be drawn by JavaScript from a meta-font.
5178 // CSS seems not follow line-splitted URL
5179 -->
5180 <script id="gsh-data">
5181 //GshLogo="OR-ITS-more.jp.png"
5182 GshLogo="data:image/png;base64,\
5183 iVBORw0KGgoAAAANSUHEUgAAQAQAAAB/CAYAANDv3f4AAAAAXNSR0IArs4c6QAAAHh1WELm\
5184 TU0AKgAAAQABAAEAAUAAABAAAPgEbaUAAABAAABAAAGoEaMAAAABAAABAAIAAIdpAAQAAAB\
5185 AAAATgAAAAAABIAAAAAQAAAEgAAABAAQAAQADAAAAAQAABACgAAEAAAAAQAACGAAwAAE\
5186 AAAAAQAAAH8AAAAAYx1BhgAAALwSF1zAAALEwAAcMBAJgcGAAAF3RJREFUEAhtnQuFNWZ\
5187 x+tTukZ3iCgg0/jY60sb8WgMzAvn7uG4+1bISTR7YnQXQpCkCj2aNw1D2MS1RkeUA-PnoCdu\
5188 4iuJx7jrlYz50D0Gmf2VqIBEiSggCoIMMA+mu+vu//ZMD9Uldau6a2uBv91CKrg3vvd6/q\
5189 fnXvd8tB88SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5190 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5191 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5192 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5193 2eXs9H+ftsKsDhXsic2qgdE7YusS+1qaalKfnY5YsokMHwEptdKMQF+5UeEslbLYsAYU15\
5194 npDIKXEZC1PirMS3JUaUq98ocQeU6i+2kK3StuONy5reEgKJ7Qw7m0vKec2ToqOiwzoi\
5195 jBOVHCstmrB3USXEJ8Fu7DsdmFb2+u4vWVFPXbBMeZU1AE/hcKoGAb66eGOLNyh56PC\
5196 HxH2VVBKORkqh3UeKiLyaOaFONJ56OkdI6w5BwomOQlyPzi0N9DLMxPfk/60p2P/Piyovf\
5197 N8mfM+/nJWNGjn9KqOTOlVGSF2p2Rillgn31j0Vk7YsoVWmzEuVFP1RKYdfoak2LRS80\
5198 zrWocCOG6EhvgRaCj/dktj3g7dXXH4gK6NARS0zPzYerzq6S6RA2DQqfk79SKTRRXhu/+9FN\
5199 L6as88pU/PnlPn1TLQJKS73dPXSr20ur7iWPeC8QhbNnCyHUIlRryOTQvY5fVqBL7jX\
5200 +cNHjBj5gJrYdLJHy39o84D04H2Qx8THaPeFIOU+w1c+KnyhK5FGEVOWGAEbX8eXMoLY\
5201 rikbd9gHEP52Vgq14h89FUA6KjYFbbQbnzLJg4zFiesNDHCwUoeeiVQob/5CY9PDU1UeH\
5202 +zghUhn8sgOqrm0uWgurki9RpjBD4Y6uQcQd5TU0W63zD3Mhesy14V49isbdKyxhG1CPFR\
5203 UJ6toACFT79V9F58NBfDHT0MbaE74Ent+eWrRw+Lz/QT60AdB7QJUjps/OA7COoNBCEMu\
5204 tCo/CG028f1pKE1TPFV8juRasEahhVxar1guoeBPyfUdo4+0feBdyb8L4t2z9eSXFAMOC\
5205 bgGuv0glzgg6W4F392xnHhdc+Mwf3JTjnt22yC1JYBJXNU25KIKyck1exXRld6BmcevN\
5206 AJovy/VbacMeVgEP46/ZlnJjt9jx17VL53Z15Mtvap1QGLNHW5p0QqYNTQ1Z2b8n6cMG2ZV\
5207 qOoFJsdyvV0AZZdFayidvFJ35CS4jXZK9hir7e27zm6p3T8hLpkyYicJpV1Htk/DJFU4Jw1\
5208 1ImhX5IR9fzgzRkX4w/C+HQSPe+krbIyrN3qEPTNahsHaLDS2xh5Q5NCoPPVDEpgeqbm/8e\
5209 7/zdoAptag/mLKJ77UOVG0xybTdx/Ex/PTfa/i7r7Ku+cSoiCXUwrohUXF16wEV9h+ccVJ\
5210 pd/CFU42AK21UP1VK1L/sjY55PvHqr728NRvfvzuvDODGy9GoopuuhNMLNfctX48YHL2QH\
5211 f/8hKXVU/43rOg9xtq6Ytvcv1XDC3fmNDQn9nbE2le7wKE1bOK65icBu0Eghd31aW82dWkPw\
5212 hrauc6zWdkcjUZR8EUXMa71zUqWcu2nbi6evN1J19/P7eW+ioMaogF+NI31JLSf8dn9ipA\
5213 WNN4rPy9jXupEDL/HXzNzgtSvesLD2vsWHW19mu5rvvzX9fos4w/LfmqdIEpHDG1fm2UCW\
5214 gJ1y2wOENR23FEci+zyNcNjYrNyhYGAo8RojTAMrIqoCJnrW5FpTn+frTwdh4SIUv\
5215 bV1WbfffLRCR04qazRD7176/zBjKyLD5pBiZ5W14wQu7tikPBeCOpUw+Kj0sp8GNNoAZuW\
5216 iO2yuzdh9z2r2XodRQVQF150xH60vVjRKAAM46pvt+RxAJVLjw7vY9/+CeUBMk168/rPQn\
5217 mCuFkZalDFN/y18A5iwc3dkIKhsyvZuCYSVG/KHewhFDRKAMMcD8EKX+rHF12A9bt2d172\
5218 2qN0vzCYDMfEtnY7QoqXDXIKAIQ7COQzchyADWnerqN5VXtctJsdGp2OtmwWJU7A+eh7\
5219 yhYbUgm1Xf77K1DwaRyUfN42F1uXNDdVetAmL6sYc9R26TbZaw2px8nfmehz3EM+mgso1k\
5220 d3/ZnBGE1XPgUWzXg1Yc5eW5/zBgy54AwogwKfNwbqptcewV74FUBvov32gew8DLzDTMAj\
5221 aupp7t/bMXx+yw/egJGkoTksy2d+gFbb9v0vDx5B1zTOR+wFjy0p6U0XGOYNgR/gu3a3vB\
5222 Fgeua6qz2d7v88Fdv3r1dBw34SGP9i0DG9h5XWknh9KaAMyJ6dk1PzZmtD3cnu77vtw5C\
5223 h/YrG1p7Wxp/VvuRduc+wsq54ym+8zXKgyRSPRA4IKoGz1i8b6ytacEPmb9v/m09CUATz\
5224 Jow6tVnPCmKHzj+sNnpHsCjYja6cRrRMyrGkiwF415UioUliL1RW7fmlLE3z2+/GfW1LU2\
5225 Y72b6Eazkfy0PetJil5QlnJyLdrFRUz1p/3pmkug/yN9gAoGyMTf7neVivx/6CHUg1l1uh/\
5226 f9Uvo+g70g3q7zFL8xq+zW+/8FP6W6fv7XShinlayWdz2X1ULm/4uLmPwNoA5UgdcoL9\
5227 ZFA6cogozhT6GQ41NR5Doj9xuvlycy+rFbcuJvsnLkV0CefphUbICLRMv1+9KP4vngHfG6F2\
5228 NCGMSiCsnkFexd+mtfLBwuxdmFb0zQ/194225Y3TCzPQWhthG2zHraJo/yb0kdhpanZq\
5229 XwFf66/8Cb5AHcbzdpnhUjeG6YFowlgzeMmtqNCDeKzTiXVuc3LK4yVtJepu5tqSwpKXdA\
5230 ufu9mFwiG3sgnntcX76+3EXOQWzVeqSpvrZmC2afSYvY461+O4KvYVgicCugG2r2poyPtwEj\
5231 021m2JWZE0+f6K0DfTnXfW2U9x70/bqZct5z0Po10+vdpyDJcdxR234U9XCEh1oSktt3uG\
5232 AcwtK009F2Fn+gWVWdS6DcFodrAxncOFRXWUSO9K3BZXXV7Ae+qwr506/204LXgngLbrC\
5233 7HgRdvtEh2Z1MXYVgqm5ZTP5+7volRR/zJ10Ylx+8ohOzEb+CV/OTU5ic3NGfjks30MZ\
5234 FtUt1l+Yi4fCwkJzqpZyb6h1JebWpWLYxo09/j8k//WW3S32QpPhV5AmTPI1DFN20p6f\
5235 fz5yWf4HfmXD+/Buy4NVU73yEFBOK65icot+zjP+8qf4JkyitngKtb/gST0MKACq18jJPL\
5236 A4PCyXmPKOTjrE8V4HpyOsws/BsqyT2RGz6zr10gA9sBhEp46hsP2ratmoJeGrugBWB2Pw\
5237 NYD1B40STBmcmdS2E/GG2ZvrF7Uejsqyw/7A7guEH6Kyy19q3fp0QvGxt4dz+Ueg+Lmy5V\
5238 bjjy+o+b5Lsgp5Nz6nwbFFHudaYgemZy4ap1z5d1bByA3NQTc4F3RKYfOTKaUF9Xry0LwU8\
5239 dMC/H29v0v0TmV1C+izhTu27rgAebk4+8H3P5530ooyu/WHj21Zwbd7z2XLuv4f1gmQSV\
5240 2GML+6LmhorvaWqne1lyZ/gLLX+IBncN2FQ7F9Y5XQEN/qua+Hr3URAggIMTLR3G3fPyEtP\
5241 6m65oyCzcXmX9nQ2JAggbBymXSL9vzQsGbfXUBjHpbXbzM+VkueBRRIotE/Bw8ogf/LIhZY\
5242 79Tcn8b681t7DqgnQRE1EVT2z9eW5SfJ71FSzovLYfTLvqUT0b62etccBRO1Lhe568SyeT\
5243 202deugmRW757ng7dKrv19rLzt0MPBK73nA4YrdZfm+5DzsymDymaHnCLoKvP0h5FSrQ5\
5244 wC76RwU9Dkx5MU9vQXMAx+ePguLw8/dvG6U1LPvsPbpXspOniQwagElsm9gNxc0E0gljv5\
5245 7tBBBjAdHKmpdV0/q/irW1bf44t5cNKQWaq7DsuJzh16Clz8bk+1u2u78FXyWfK1q4/qY2x\
5246 tyYjX8boyWm6zwc9/Ojwz7PtuV1Lp0NQ2UoL8PKODMuluvo0TDjLyxcrNWHHEjJw5yKrkPs\
5247 2Jh14LpJcQXOyp6nMs5fysKeille0G95+WxCEj3m5mcmjNe5b+lyhZYLXgJrMdnY/HMK0S\
5248 aP7Md34PueUvz8DWDovSjzXVf/xsFe+Lpz/wjQQ9eih94ZwqVS62+CUHv31MtnjshfXorHf\
5249 wKz9g9FwITCRJw3Wh5+/ocSLZQ1z652BtVtG+wOpqXRYEwCarfrdbSgC5bD/PySxBhakPWO

```

```

5250 qz9y4L10uABB4x5we8qDsH06++b0nWjzFXyAUvYi6Ece001I7SAZkxOUgxmZB9RcaVyxX\
5251 2CMBjadTcrUwWYkrIwy4myTH9zt3R93/8X1j0ESWetyy7gFj1lodwKAmhFEA2KD6DlWne6h\
5252 H52huWwLALQHQOZUyZr6yzntLs7rGu4oYBjg4JBWJGayRhTyeX4X8\XCw+r+s9L5yc0A+W\
5253 8v0w0N2ZxXw7VzBzCEdpXpdsLkODeFwEMHj47AEaa7yXzXm+61PzUL46ch7c0Q6g/m\
5254 Wwcf9R7yXbs6Z3hhxvFvmlkjhJUBTFKbbag1QCWibuu1FPytkLhWzEag8YK0eMcj191yPly\
5255 Pw79U/55Bk75ESXMcwhwJ7935xv77q8YspwTbgs5+55hdjJne6Y8fzFyVOL2xoeLcbmWj\
5256 Ywkg5S2p1j0K5d4zgs+2LBA4Z6/1g+woaSa6yUOY1jzcCuO641llpQV0Fep1wulXUL4PFR\
5257 zD3G6wVb4JA35zeKLN1SuBb/34Rcw6B4XGgZrFlBBjbbH7tLWbGDr04b1eXkgPbbHm\
5258 NQ73IqMHZ7THWuXsv45z8FpFKMRc2LzBJBk8mU55cn4x/2rLdJQzNjKkyuu01pdqCFM\
5259 aWdrXYb9m9ZMLHmChjCvUUFKMRc2LzBJBk8mU55cn4x/2rLdJQzNjKkyuu01pdqCFM\
5260 KQp/ahfXkooVi-JfofzfmZuJyn8P7QHmhmKd4aUeTx6c7F07SUUKgy50a3wV/20C7b+scH\
5261 Ltp1tH3Yw841pGt4JWAnu7Pn5xwqjB41MabBc3Q8rFLZPCJFte05F0B8aD5eFwYfHBu\
5262 ml1qjJTHChN3eSR+42Mk5KwEtsMx35RJVtor3rmm49VMOGF8o1D191X61dvbMkgjvB\
5263 NfydX9m8WimLMLKzeSL/VzQSkDpccdycte71g/B4XkFKQaek3ML47+29fQl/gaT+VzO\
5264 pDTT0U9UwBkXUWf9MULzJvPxxu0fP00/pTeehdod/1XXGzWafuXp6eG1Lzreme2X9lbo\
5265 0xuU119f0LAKGqghafa5NVPHxjK7X0GuOMRm+JAFPeSannaKzLRhZklyB5e5idUwK1/wD7\
5266 fd+JL72vEtdPEJggWkZj6zEP/d5durtz+ZHhXkLnhs7umT011AjkyVScenp1UwLAAcAe\
5267 dgY2Sx/S+nLN0DpE1KVLD/SkUr+JL5/9VsbL75z+bYNS8Q2EuQn/Oa3x1/FJZS/Vz30EGCB\
5268 ePdtCYCRORCKr3q6vL0pof7XkVdAaVzcgJocZCZ56CYcmxZ/7CyurW2IINzK4NOC075/\
5269 yfWRK3XuyfGjgmxT/xdbpt8uSR17F11lUoFJt0mJ3J7KkfygMVsDvwpVq9RPaeh07FRV\
5270 hUL469JpwUlyYn+FX0C+Cy0VrWkXzylh/w3n7fiiBreUteVURMitjpkWRmPKZmHdZFcIM\
5271 dm1f6ew10/651HmCDD2YFE12FyCgJ38ARbQSPGXISCGUCaKrdOuyZauvgZ6zAVTfE\
5272 LLGfLXPJfjYjYtChKphr+cN+r76LoL143d451+andv9Yr4veCW99+SrTx6G/areZLkBX4W\
5273 tgyr7Wk4n+gF/FzZURIA3ky5Ulm09CE8N3HgLin15ISRnY32hsXorNTBmVWmiP9zT03\
5274 j0g8vnn35zeCGYfjCmlw2/fvicJoJyt.ieoL0XvRghMynZ1/IJtL6w3j5y8j+711dy057\
5275 xLjUm+XoF0gtrugEUTDIPfCenowAE2KAeVArG5T3tJBCQT+5rCU+U1Bzxp1PJumpRVP\
5276 4YeuZ9w9x1fVv/oppuyXp9uFyih91/XNXovNSd5dGG8C8wms31CzfrkCOUTCZSHj+wm8Q\
5277 JV7XE3M6WqjL5r6LV6687ToExtHj/4Cdw24+uzFvsJrT1lRkFoOoALTzPzFd2D12QrQ\
5278 8YV889psbshvRLDp6exvrE0J3y4q9DQPKC5ZmjJyz021LdV7y3zfl8qmsDm0PARTVWFC31\
5279 N1NQGW1jEavqOmz78D2ZveFmhdFCU86nbFBB5:Kf1FPMRHE6Fo05OatvM/d8V98km7D\
5280 C5YrseFLvslpLpx79z64erdZNYuNLK1leJdaluak7j0orr315x+YA9CbQDF/cK7JHdDb\
5281 E5sg690KMH9pdrJd6v3vgEYvbdQcusc1VM9n0/QaPP3KZlve8zWcmJk30kx+30RK0E8K1wN\
5282 lxaFhe29JqBL86FGKam6n5P9mdGP5bmtIkpmc22r7BHSKjP0kMCKtCF/KAM10EJtEjK\
5283 v7q+/OmZybn/Z5iOHT3+NwPzn2eyx7uiZ0JDM9xoyTcZBTOya+vndq3URP1jYxbmD01/au\
5284 zq4BeyGslmpghdLIXcmLWskzBgwa940stveB+sf714IiK3oi05Mxod+r9/I2Vx809Pec3\
5285 xp7QYU8JTGmcat0+NeY/99v3xhb+21bh3cnot1jdfCznzkeapsDN/vjdg4XPcnb8+W9p\
5286 9zaduKz2Q3f6ev05lytqto30pzK9E5sHOY+FXE5150r6xr5HkDFMGAAadK3Y0A9ydyfJd\
5287 pp5k1Nq6grnY13DFyKI5h14oOKj1azehBj9NtWTFBAGv1uIawS2xVtAhBs50dfpseEaP\
5288 mRf1LXOm8XmXnY/fBy6av2fyt5Skwno2mPFSF3sgCf304UGGSj/wI548wVlFvVb4720b\
5289 Xx/MwclF9zXPQMbMx5IAfjIhYjhsR7BKMF8GmT+D3CdJF2qod1vNN33d60xw7hyf\
5290 koSVf0EpkZfegJWQtd1d70c6dnp1H7zi0z9330LHWYJULrEhZ7ptxeVe9XWH+3Jd0mestO\
5291 lEWe1C5J8EajZNR0adga71evOR2LBSccV820u5Ue1JbspxVgEeuscjRkKjLW0VSSUInTmW\
5292 LaycXhPswtKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5293 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5294 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5295 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5296 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5297 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5298 m3Bw0ncAAAAASUVRK5CYII=";
5299
5300 GshIcon="data:image/png;base64,\
5301 iVBORw0KGooAAANSUHEUGAAAKWAAAB/CAYAAABYmyLZAAAXNSR0Iars4c6QAAAHlWE1m\
5302 TU0AKgAAAAGAAEAAEAAUAAAABAAAAPgEBAUAAAABAAAARgEoAAAMAAAABAAAIAAIpAAQAAAAB\
5303 AAAATgAAAAAAAABIAAAAQAAAEgAAABAAGAAQAAQAAAAQAAACgAAEAAAAQAAAYgAAWAE\
5304 AAAAQAAAH8AAAARACT6tZwAAAA1wSfLzAAALEAAACMBAJgcGAAADQRJREFUeAhtn09vFNUd\
5305 x9/b21+zIYKCI1K1amW1/jjB6CKstFEFth1IqPwdstQoqEunttW2nFg01YtIatinZ0\
5306 amdAQY6jIyoX17kgglarVv74b3BAQPKbVajj3e3r94wCJpe93csmobj784kd/v7723+3nF\
5307 fFv+nxA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
5308 SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
5309 XiEu65tAhEwAd8Imp2wLTXtdTadyMzrT+42pZrSrd3pQvPksMtrrhgNc8fEwHXfUap+2yH\
5310 ZUASIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
5311 hf7WveE6PyD+oM6JmXwzxn6NREVCgS4SQXwhL18xtFFcuWghX7AMRgIKQAIKQAIKQAIKQAI\
5312 Nbcgm30981Ho0S8sa5044oUmz+EzCEAE/FWHXfnj9sOfld/YP80ZnJKRALVawJEGG4C/BGSe\
5313 rgK0AMB/d3uCJ1WJsYVnsIy0KA08FevYnmsYJYR5F3cmUmL6v/FwDD1QAV4S7EBpZyQ/\
5314 65pVScodz46qncHyZlLzEdtLk7M51Ayw2ymTmXNda8LcplJlWw0i0lU1/3sD05692zqBP\
5315 2DBH8ZDXp+28K1icYgJ9Fve6Eh5OVcInOVegOKFL1K7qgKvWbUnKnF0odS814tKfyWGPt\
5316 0lC2e8+3+ra1p1162LEVYnPs6SQfapwSgmV0Kf8pDwGkzleSbaWPFudreUyYUvEPhwH9\
5317 faWfI9Qot4C27yYoroBVF9CrE/2gnEo/ElPa4DqHalosCUT4rPjzseCLGN56Z10QvRtE\
5318 rHdXjvnnSHvmsYPTq6UEZEgEJS2EWmpJ4sk83SVQat/zSelLuz5aem1hZrVkdphAVH0\
5319 F6R5ZaZf8510gmVortLseXG/oTLB9s++5h8u0ihusYfZ19f1Glp2cNSyt1IA2//w0078aEK\
5320 IX87zhymPTkTb3oc1bXxa/DKX3bYpoeHh686jgcSEXCugvXALy+CVN0S08mm19U0OH+0h\
5321 zFN7phGbb3Ck0oJ1F09zr7rGvN341QNRtg4570TS1hkYsJSuok6478h1pRof614D4mU7N\
5322 4t2xw1BFIu1BE6G0w2+T9JpFNk7WUbrms5WgpjCTK13801uLcApeIwCLADaUXqEoc1Ys4\
5323 LB703kKUEt04panz0+9y0NN9ANsdFQmN4oxSnokzgn+foCANUSNmm3unjXgAOvHzuc6+ \
5324 CGpZDuFdWGMGrf0n8BezLJXDYscHa+vntqfDT10QH1kCVeJ9jsVjebVqEts0sV1/ERXV9FE\
5325 74arv8+Eyc/v6mf7Xamn05Kqnr60Ylem/+qGTOA6dXNt28weCCamh8Moubur8I5Bb1bk6EC\
5326 Rbhvm2brm7h195Jv11hSPpwyEn9SxhL6JNE7K1+UwQNK2HcmG5M6VJZwU1U1Y19TUXe+ \
5327 sB0ngishovWt/2Fw5q6wVhVwYt4r/Quk2WP20gihfSnYqgqYaZ5tBav0/Pdh1DX8FuW\
5328 lfbpLH6f7fbb7Vc6FkUXFvHs0ZYSjzclTK3TtJWR1jeCr0HtS1rJCXhIu09N1xfvGk3\
5329 5wZuDRKXN211qV08pLmXu0BCa2paVgJdel20+Suop3S11+3idYU2NxcnCDUVAFJ2K0847e\
5330 Gp4eUaUP7R/74/WD77y76+a3c574a/FyENPhy9Yb/cvZPn3oYfvtY3PCJPJ0FAGkAet3\
5331 7T4wC6jEqpKBy0eAKDXFG7FKWKh72Wx453a+vBksbwt1a7r0hPrY9M9yotQldvotJd2/1\
5332 Tr/7DA8W5jK8WVTLKfums4CsRv411A+Y8t/zd1L0nolVcTe7fDhK3+TazxSTJL12K6\
5333 C8v2lGcfhKsrNeUxNQ9Uv0K390MfZTUYZa8pA05RyNejKA0/hu0tNw+cc5Y9264CNL\
5334 TyPH01R+3pL9VMMidCpG6pLLTtasnpJRC0+BiH0q9KGNrXmH4drYNeZuJfzvKb1NQXZ\
5335 eQd16tyHZZ6MWC212h9Ye9+0/wJ2ajrQ+hFTFPFKytZ0qjpvsvrX70z6ZobWY1J0ePsfN3\
5336 cSNYwSfS13RQTKH17ky7cCT+GEaor2odzvHgmI/O9rVtAhpulzsy0kf99UCZDB121J\
5337 2yDwtdLSVHX9Fdrt1h1f0qMKMF/29W2y7k7ZDUuz1butncUdLMkyR600L450Y2RS1uy8E\
5338 213vcxGbegY2F3BH6gAT7f3yc0VArNdIoMx/886D1mVSB1TZPT58DnaCMHxwPmaH0MmbfM\
5339 vhdsgJNGjXhR00u841F/WeBp4PALZG1gtD1Zu7VBNBRp9q/ubABEYVYK/ulF8EXgsvC\
5340 V9eGEnbQ/DSv0RwSrxRiQB34E0x/ssYpD3Wk9owbTpeZp++jFTSodyoAzC6xr/ofj5QrDY\
5341 BnasW4zhsv+2rDrY5gZQAVdp5We1t/CQSumZyW6HgmGfJRo/y4aaU+7GfEyl+LS0KwC+PC\
5342 AjzxwVwLdL+BY07RA6IHTuc8jclEpNNZ5zqZ4PS8K09H9p552S3TmJngu8zUPTN+OL/C\
5343 dc2P4UFAhmsfn47H6R12VnwjzrZ5LuflwLsBs0YF72KosQJYjzn2FL0a0GKg46b8+BXyQ\
5344 TKvWenYqepTgZ2ftH6ghg26/j8AePKnoB059jzLh9L+084E59SUQhk165VwG6P3njyDW\
5345 85zr1S001+qAbRR6Tso+r2Bmqxv2xrc0ssSMqI/fCFY7LPdZ21Jzr5KK+C5EdLh61xYTWL\
5346 V/nm4/cmBCWn+XmNWee48EznelWaoF+EKYUro1IDOGFL3PawpZRGFP1nIhtCOXYQ5CLPQW\
5347 RGJ4fb1+LEuY3Vnc8BzF4Kv1szzeFVNVs6qj3Qv++Y0t29BUzGwJrjgWZD11oBJwXzE18VIX\
5348 KEPL9fdsBxp/2Xs6gXKM3dfClatfd8adBN1u0UNh1aFwg6Bw93sevqj1HMULPw/b14a6npg\
5349 pksWlwr06FYNm+3VmlU6MwfbD3KwyWtXwJ2z2Ue04jSj+6WUyJbTL1LpSv1okE327S/NJwX\
5350 MgJ+21W6vtW1T14Ty/bUnDxmS71u9baqa2OYX5DbUX1z9BRpGevDrDHJ51k3m3z394VgSdp\
5351 qZbnk1kQbbVhBteH1610Vv/zsZaaEFLr+NOBXC9Y90Xa7q7BbtQ6tbuV/oYiPhu8xhZ4AR7\
5352 0lMaou3Q4pY2WwWct1a17Ndi3bXo2P7v2p70cmmEpycew8L4Q6770Evh+htZPNEd+NFy/W2\
5353 9LRATH+5EJ/vQ5fE1LwStTREM44Au5+Q3T6A8rsqdmx7Z+/GB47u1290Uw9JX4AnJ711S\
5354 16Y+1x8fsm6YcrRoXul4K1ysH6Be9110YHs791/4cxvbnH2jWB1j1XXxecYQUZ005WDUJ\
5355 Y6MFG2XRCb6w7rTjP5NO7ZuP3v91rmdNn4F5eSBK0HOTab6aStQot7/beUGSubEmhrC27B1\
5356 0YqHs10JvclY04fxKoz+kqw+oaJDVESgVer9PehP+SrXwnkMNLm6VpQnUikXz+mPveQgV\
5357 h4F8J1j9WOrt+64Stf500WEZd2G5tCd/FZS/VXH3nagrQUL+4B2j8m8Ss/FmI9D3McBjwo\
5358 kRn3KZuzqzpsZKZU1cbOCRjmfWhQ1aBxM1gdsU1315d3JLr9ywoVm9Np1kTie/CQYldWZV2\
5359 8/qpxnKkvclbdeIDD0Usc+RxsD1pnmXW78YVM6HGDc2f2gZnJ+8xzSRBvQ4zrhEw9\
5360 H92687JSOHPzRdII7AAPQwzK17LsplurBj0QpXyYb/8dmn2//11/qnago2Awqf/38+WE1\
5361 I4af+505EXMARKAoI2CCP2xvJNV+LMZ78Lk3V2TLWVt2n9w4/+6JgdkJPLq7b1TADkw1p\
5362 InH+QSI+HieW5SRPuvengV20d6NfK0t0lFjdj/ikUsatCEBEIABEIAEIAEIAEIAEIAEIAE\
5363 EIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
5364 EIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
5365 EIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
5366 EIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
5367
5368 ITSMOREQR="data:image/png;base64,\
5369 iVBORw0KGooAAANSUHEUGAAAGAAABVAAQAAADYAAAB1BMVEX///9BAEPhqDaAAAB\
5370 Hk1EQV04jdxTsa2EMawGYCMX7s1CKVgJXvACBe7CARASXda1LAWgS4HwM5zEVs+mvSgS+2BQ\
5371 8gcB4BdHyZv8zMSaUBHNM+KAd4QCLDpDn8ogT4UpPGci2j18IGF3eLwPwAhkVWYwecv\
5372 UEBDXaB0X2anJueVDoZnK1qassPCKjc4nW3E1SfWqY6jU/vAKPhg0ALSpHve8Jt0dkWDMwr\
5373 YMGSSuPYWHAr19k0tkv2sb3sdw2rUCqW88g4Rp1A9s1TPv9cTPlNRD4XFKIn8XaQCWT6Lzq\
5374 Z08dhw/4+U2Gzq158gbqVmkfr1N6YXK8Q1D00mLGM7vPERA8AL9vwb0iFpS0L33fsYtrrL\

```

```
5375 S9wiqDzznhUI38v5n783/gBuUs2eLg1c8gAAAAABJRu5ErkJggg="";
5376
5377 </script>
5378
5379 <script id="gsh-script">
5380 //document.getElementById('gsh-iconurl').href = GshIcon
5381 //document.getElementById('gsh-iconurl').href = GshLogo
5382 document.getElementById('gsh-iconurl').href = ITSmoreQR
5383
5384 // id of GShell HTML elements
5385 var E_BANNER = "gsh-banner" // banner element in HTML
5386 var E_FOOTER = "gsh-footer" // footer element in HTML
5387 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
5388 var E_GOCODE = "gsh-gocode" // Golang code of GShell
5389 var E_TODO = "gsh-todo" // TODO of GShell
5390 var E_DICT = "gsh-dict" // Dictionary of GShell
5391
5392 function bannerElem(){ return document.getElementById(E_BANNER); }
5393 function bannerStyleFunc(){ return bannerElem().style; }
5394 var bannerStyle = bannerStyleFunc();
5395 bannerStyle.backgroundImage = "url("+GshLogo+")";
5396
5397 function footerElem(){ return document.getElementById(E_FOOTER); }
5398 function footerStyle(){ return footerElem().style; }
5399 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
5400 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
5401
5402 function html_fold(e){
5403     if( e.innerHTML == "Fold" ){
5404         e.innerHTML = "Unfold"
5405         document.getElementById('gsh-menu-exit').innerHTML=""
5406         document.getElementById('gsh-statement').open=false
5407         document.getElementById('html-src').open=false
5408         document.getElementById(E_GINDEX).open=false
5409         document.getElementById(E_GOCODE).open=false
5410         document.getElementById(E_TODO).open=false
5411         document.getElementById('references').open=false
5412     }else{
5413         e.innerHTML = "Fold"
5414         document.getElementById('gsh-statement').open=true
5415         document.getElementById(E_GINDEX).open=true
5416         document.getElementById(E_GOCODE).open=true
5417         document.getElementById(E_TODO).open=true
5418         document.getElementById('references').open=true
5419     }
5420 }
5421 function html_pure(e){
5422     if( e.innerHTML == "Pure" ){
5423         document.getElementById('gsh').style.display=true
5424         //document.style.display = false
5425         e.innerHTML = "Unpure"
5426     }else{
5427         document.getElementById('gsh').style.display=false
5428         //document.style.display = true
5429         e.innerHTML = "Pure"
5430     }
5431 }
5432
5433 var bannerIsStopping = false
5434 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5435 function shiftBG(){
5436     bannerIsStopping = !bannerIsStopping
5437     bannerStyle.backgroundPosition = "0 0";
5438 }
5439 // status should be inherited on Window Fork(), so use the status in DOM
5440 function html_stop(e,toggle){
5441     if( toggle ){
5442         if( e.innerHTML == "Stop" ){
5443             bannerIsStopping = true
5444             e.innerHTML = "Start"
5445         }else{
5446             bannerIsStopping = false
5447             e.innerHTML = "Stop"
5448         }
5449     }else{
5450         // update JavaScript variable from DOM status
5451         if( e.innerHTML == "Stop" ){ // shown if it's running
5452             bannerIsStopping = false
5453         }else{
5454             bannerIsStopping = true
5455         }
5456     }
5457 }
5458 html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
5459 //html_stop(bannerElem(),false) // onInit.
5460
5461 //https://www.w3schools.com/jsref/met_win_setinterval.asp
5462 function shiftBanner(){
5463     var now = new Date().getTime();
5464     //console.log("now="+now%10)
5465     if( !bannerIsStopping ){
5466         bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
5467     }
5468 }
5469 setInterval(shiftBanner,10); // onInit.
5470
5471 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
5472 // from embedded html to standalone page
5473 var MyChildren = 0
5474 function html_fork(){
5475     MyChildren += 1
5476     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5477     newwin = window.open("",WinId,"");
5478     src = document.getElementById("gsh");
5479     newwin.document.write("<"+<"+html>\n");
5480     newwin.document.write("<"+<span id="gsh">");
5481     newwin.document.write(src.innerHTML);
5482     newwin.document.write("<"+</span>"+</html>\n"); // gsh span
5483     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5484     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5485     newwin.document.close();
5486     newwin.focus();
5487 }
5488 function html_close(){
5489     window.close()
5490 }
5491 function win_jump(win){
5492     //win = window.top;
5493     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window/opener
5494     if( win == null ){
5495         console.log("jump to window.opener("+win+") (Error)\n")
5496     }else{
5497         console.log("jump to window.opener("+win+")\n")
5498         win.focus();
5499     }
5500 }
```

```

5500 }
5501
5502 // source code view
5503 function frame_close(){
5504     srcframe = document.getElementById("src-frame");
5505     srcframe.innerHTML = "";
5506     //srcframe.style.cols = 1;
5507     srcframe.style.rows = 1;
5508     srcframe.style.height = 0;
5509     srcframe.style.display = false;
5510     src = document.getElementById("src-frame-textarea");
5511     src.innerHTML = ""
5512     //src.cols = 0
5513     src.rows = 0
5514     src.display = false
5515     //alert("--closed--")
5516 }
5517 //<!-- | <span onclick="html_view();">Source</span> -->
5518 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
5519 //<!--| <span>Download</span> -->
5520 function frame_open(){
5521     oldsrc = document.getElementById("GENSRC");
5522     if (oldsrc != null){
5523         //alert("--I--(erasing old text)")
5524         oldsrc.innerHTML = "";
5525         return
5526     }else{
5527         //alert("--I--(no old text)")
5528     }
5529     banner = document.getElementById('gsh-banner').style.backgroundImage;
5530     footer = document.getElementById('gsh-footer').style.backgroundImage;
5531     document.getElementById('gsh-banner').style.backgroundImage = "";
5532     document.getElementById('gsh-banner').style.backgroundPosition = "";
5533     document.getElementById('gsh-footer').style.backgroundImage = "";
5534
5535     src = document.getElementById("gsh");
5536     srcframe = document.getElementById("src-frame");
5537     srcframe.innerHTML = ""
5538     + "<"+<cite id="GENSRC">\n"
5539     + "<"+<style>\n"
5540     + "#GENSRC textarea{tab-size:4;}\n"
5541     + "#GENSRC textarea(-o-tab-size:4;)\n"
5542     + "#GENSRC textarea(-moz-tab-size:4;)\n"
5543     + "#GENSRC textarea(spellcheck:false;)\n"
5544     + "</"+<style>\n"
5545     + "<"+<textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">"
5546     + "/*<"+<html>\n" // lost preamble text
5547     + "<"+<span id="gsh">" // lost preamble text
5548     + src.innerHTML
5549     + "<"+</span>+<"/>\n" // lost trail text
5550     + "<"+<textarea>\n"
5551     + "</"+<cite><!-- GENSRC -->\n";
5552
5553     //srcframe.style.cols = 80;
5554     //srcframe.style.rows = 80;
5555
5556     document.getElementById('gsh-banner').style.backgroundImage = banner;
5557     document.getElementById('gsh-footer').style.backgroundImage = footer;
5558 }
5559 function fill_CSSView(){
5560     part = document.getElementById('gsh-style-def')
5561     view = document.getElementById('gsh-style-view')
5562     view.innerHTML = ""
5563     + "<"+<textarea cols=100 rows=20 class="gsh-code">"
5564     + part.innerHTML
5565     + "<"+</textarea>"
5566 }
5567 function fill_JavaScriptView(){
5568     jspart = document.getElementById('gsh-script')
5569     view = document.getElementById('gsh-script-view')
5570     view.innerHTML = ""
5571     + "<"+<textarea cols=100 rows=20 class="gsh-code">"
5572     + jspart.innerHTML
5573     + "<"+</textarea>"
5574 }
5575 function fill_DataView(){
5576     part = document.getElementById('gsh-data')
5577     view = document.getElementById('gsh-data-view')
5578     view.innerHTML = ""
5579     + "<"+<textarea cols=100 rows=20 class="gsh-code">"
5580     + part.innerHTML
5581     + "<"+</textarea>"
5582 }
5583 function jumpto_StyleView(){
5584     jsview = document.getElementById('html-src')
5585     jsview.open = true
5586     jsview = document.getElementById('gsh-style-frame')
5587     jsview.open = true
5588     fill_CSSView()
5589 }
5590 function jumpto_JavaScriptView(){
5591     jsview = document.getElementById('html-src')
5592     jsview.open = true
5593     jsview = document.getElementById('gsh-script-frame')
5594     jsview.open = true
5595     fill_JavaScriptView()
5596 }
5597 function jumpto_DataView(){
5598     jsview = document.getElementById('html-src')
5599     jsview.open = true
5600     jsview = document.getElementById('gsh-data-frame')
5601     jsview.open = true
5602     fill_DataView()
5603 }
5604 function jumpto_WholeView(){
5605     jsview = document.getElementById('html-src')
5606     jsview.open = true
5607     jsview = document.getElementById('gsh-whole-view')
5608     jsview.open = true
5609     frame_open()
5610 }
5611 function html_view(){
5612     html_stop();
5613
5614     banner = document.getElementById('gsh-banner').style.backgroundImage;
5615     footer = document.getElementById('gsh-footer').style.backgroundImage;
5616     document.getElementById('gsh-banner').style.backgroundImage = "";
5617     document.getElementById('gsh-banner').style.backgroundPosition = "";
5618     document.getElementById('gsh-footer').style.backgroundImage = "";
5619
5620     //srcwin = window.open("", "CodeView2", "");
5621     srcwin = window.open("", "", "");
5622     srcwin.document.write("<span id="gsh">\n");
5623
5624     src = document.getElementById("gsh");

```

```
5625 srcwin.document.write("<+style>\n");
5626 srcwin.document.write("textarea{tab-size:4;}\n");
5627 srcwin.document.write("textarea{-o-tab-size:4;}\n");
5628 srcwin.document.write("textarea{-moz-tab-size:4;}\n");
5629 srcwin.document.write("</style>\n");
5630 srcwin.document.write("<h2>\n");
5631 srcwin.document.write("<+span onclick=\"window.close();\">Close</span> | \n");
5632 //srcwin.document.write("<+span onclick=\"html_stop();\">Run</span>\n");
5633 srcwin.document.write("</h2>\n");
5634 srcwin.document.write("<textarea id=\"gsh-src-src\" cols=100 rows=60>");
5635 srcwin.document.write("/<+html>\n");
5636 srcwin.document.write("<+span id=\"gsh\">");
5637 srcwin.document.write(src.innerHTML);
5638 srcwin.document.write("</+span><+html>\n");
5639 srcwin.document.write("</+textarea>\n");
5640
5641 document.getElementById('gsh-banner').style.backgroundImage = banner;
5642 document.getElementById('gsh-footer').style.backgroundImage = footer
5643
5644 sty = document.getElementById("gsh-style-def");
5645 srcwin.document.write("<+style>\n");
5646 srcwin.document.write(sty.innerHTML);
5647 srcwin.document.write("<+style>\n");
5648
5649 run = document.getElementById("gsh-script");
5650 srcwin.document.write("<+script>\n");
5651 srcwin.document.write(run.innerHTML);
5652 srcwin.document.write("<+script>\n");
5653
5654 srcwin.document.write("<+span><+html>\n"); // gsh span
5655 srcwin.document.close();
5656 srcwin.focus();
5657 }
5658 </script>
5659 -->
5660 *//<br></span></details></html>
5661
```